Understanding Programs

Wolfgang Schreiner
Wolfgang.Schreiner@risc.uni-linz.ac.at

Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria
http://www.risc.uni-linz.ac.at

July 28, 2008

Abstract

This work-in-progress document describes our understanding of the semantics of programs written in imperative languages, of the specification of program behaviors, and of the rules for verifying that programs behave as specified. The presentation is based on the formal modeling of programs as state relations; it may serve as a foundation for computer-supported program reasoning.

Contents

Chapter 1

Introduction

In this "work in progress" report, we present the main ideas for a program reasoning calculus which is based on the semantics of commands as state relations: the execution of a command in a pre-state may result in zero, one, or more possible post-states; the semantics of a command is thus defined by describing the relationship between the command's pre- and poststates as a logical formula.

The core idea of the corresponding program reasoning calculus is to lift the "commands as relations" principle from the meta-level (the definition of the semantics) to the object-level (the judgements of the calculus): a command/program implementation is translated to a predicate logic formula I that captures the program semantics; the specification of the command/program given by the user is also such a formula S; the implementation is correct with respect to the specification, if $I \Rightarrow S$ holds.

We believe that, independent of the actual verification, the translation of commands to logical formulas may give (after appropriate simplification) crucial insight into the behavior of a program by pushing through "the syntactic surface" of a program and disclosing its "semantic essence"; this is similar to Schmidt's approach to denotational program semantics [13] (which however uses a functional model rooted in Scott's domain theory). For this purpose, the calculus is settled in classical predicate logic (in contrast to other approaches based on e.g. dynamic logic [2]); this is the logic that (if any) most software developers are familiar with.

The idea of programs as state relations is not new: it is the core idea of the Lamport's "Temporal Logical of Actions" [10] where the individual actions of a process are described by formulas relating pre- to post-states; Boute's "Calculational Semantics" [3] defines program behavior by program equations; related approaches are Hehner's "Practical Theory of Programming" [5] and Hoare and

Jifeng's "Unifying Theories of Programming" [6]. Calculi for program refinement [1, 12, 11, 4] allow specifications as first-order language constructs at the same level as program commands with which they may be freely intermixed.

While the present report builds upon these ideas, it has a different focus. Most of the calculi described above work on simple "while languages" that have clean and elegant calculi but neglect "messy" constructs that would complicate the calculus. Our goal, however, is to model the full richness of program structures including

- local variable declarations (and thus commands with different scopes),
- commands that break the control flow (`continue`, `break`, `return`),
- commands that raise and handle exceptions (`throw`, `try ... catch`),
- expressions that raise exceptions (`1 / 0`).

Furthermore, our calculus includes program procedures ("methods") with static scoping; it supports modular reasoning about programs on the basis of method specifications (rather than on the basis of method implementations). A core motivation of our work was to understand in depth the semantics of modern "behavioral interface specification languages" such as JML [7] or Spec# [14] (which build upon earlier specification languages such as VDM [8]) as the basis of software systems for specifying and verifying computer programs; the method specifications in the present paper are derived from these. The current version of the calculus handles most aspects of imperative programming languages with the major exception of datatypes and pointer/reference semantics semantics (programs operate on mathematical values). It does also not address object-oriented features (object methods, inheritance, overriding) or concurrency.

Since our language model is much closer to real programming languages, the rules are frequently considerably more complicated than those in the calculi presented above. However, we wanted to deal with the current programming reality "as it is" in contrast to what one might think it "should be". We also wanted to stay as close to the source language as possible and avoid translations to simple core languages (such as performed in ESC/Java2 [9]) since these tend to obfuscate the relationship between the program text accessible to the user and the ultimately constructed semantic interpretation which is used for reasoning/verification.

While relational frameworks are good for modeling "partial correctness" (no terminating computation exhibits a wrong result), they have problems with modeling "total correctness" (every computation terminates). There have been various attempts to embed "termination" into the relational structure, e.g. including "non-termination states" ($\perp$) into the domains of the relations or by simply demanding

that for correct programs all computations must terminate. We find these approaches not satisfactory and therefore treat "termination" as an issue orthogonal to (partial) correctness: every program/command is, in addition to a state relation R, specified by an accompanying state condition C: only if C is satisfied in a pre-state, the command is required to terminate (in some post-state allowed by R); the only connection between C and R is that that for every prestate on which C holds, R must allow some post-state (otherwise, the specification is inconsistent).

The present report describes "work in progress" towards the envisioned calculus; it is written from front to back in chronological order and thus documents our increase in understanding over time. It starts with a simple programming language and corresponding reasoning calculus whose soundness is proved with respect to the semantics of the language (completeness is not addressed in this report). Proofs have been elaborated in reasonable depth relative to numerous lemmas (listed in the appendix) which have mostly not been proved. The programming language has been repeatedly extended and the corresponding calculus has been further refined and modified; sometimes critical proofs are repeated, sometimes previously derived correctness assumptions are (if plausible) taken over to the new version of the calculus. In some parts (e.g. semantics of undefined program expressions), alternative versions are discussed.

As should become clear from above description, the report is *not* a refined presentation of the envisioned calculus; it has however built up the elements of our understanding. As a next step, we will summarize in a short reference paper a consistent form of the calculus that can serve as the basis of further implementation in a computer-assisted program exploration and reasoning framework.

Chapter 2

States and Commands

In this chapter, we lay the foundation of our treatment on program reasoning. We introduce the core idea of considering programs as relations between system states, introduce the syntax of a command language for manipulating such states, give this language a formal semantics, introduce a language for formally specifying properties of such programs, present a calculus for verifying such specifications and show that this calculus is sound with respect to the semantics of programs and specifications. On this basis, we will in later chapters extend the command language to a full programming language with loops, methods, recursion, exceptions, and expressions with side-effects.

2.1 Programs and their Meaning

Our goal is to describe the behavior of programs whose core principle of operation is to modify the *state* of a system (*imperative programs* as opposed to e.g. *functional programs*). The most important part of a state is typically the *store* which holds the values of *variables* that may be read and written by the program. We thus assume the existence of a set *Variable* of program variables and a non-empty set *Value* of variable values and define

$$\begin{aligned} \mathit{Store} &:= \mathit{Variable} \to \mathit{Value} \\ \mathit{State} &:= \mathit{Store} \end{aligned}$$

(later *State* will be redefined to capture additional features than just the store). If we assume $\mathit{Variable} = \{x, y\}$ and $\mathit{Value} = \mathbb{N}$, then a possible state is $s = [x \mapsto 2, y \mapsto 3]$ which holds in variable x the value 2 and in variable y the value 3.

Consequently, we will assume that $\mathbb{B} \subseteq \mathit{Value}$ i.e. that *Value* contains the Boolean values TRUE and FALSE.

If a program starts execution in a *prestate* s and terminates after some while, it leaves the system in a *poststate* s'. For instance, the execution of a program with single command

```
x = x+y
```

in the prestate $s = [x \mapsto 2, y \mapsto 3]$ yields the poststate $s = [x \mapsto 5, y \mapsto 3]$. For a given program, the poststate is often uniquely determined by the prestate (as in above example), but this need not be the case. Take the program

```
var z;
x = x+y*z;
```

which introduces a temporary variable z which is not initialized (and thus may have an arbitrary value from $\mathit{Value} = \mathbb{N}$). If this program is executed in the prestate $s = [x \mapsto 2, y \mapsto 3]$, we now about the poststate s' only $s' = [x \mapsto 2 + 3 \cdot z, y \mapsto 3]$ for some value $z \in \mathbb{N}$. Possible poststates are therefore $[x \mapsto 2, y \mapsto 3]$, $[x \mapsto 5, y \mapsto 3]$,$[x \mapsto 8, y \mapsto 3]$, ...; however $[x \mapsto 3, y \mapsto 3]$ is not a possible poststate.

In general, a program thus describes a *relation* on states; we say that the *semantics* (meaning) of a program is that of a relation on states. Consequently we define

$$\mathit{StateRelation} := \mathbb{P}(\mathit{State} \times \mathit{State})$$

as the set of possible program semantics. Given a set "Program" of possible programs, our goal is now to define a valuation function

$$[\![\_]\!] : \text{Program} \to \mathit{StateRelation}$$

which maps every program $P \in \text{Program}$ to its semantics $[\![P]\!] \in \mathit{StateRelation}$ i.e. to a relation on states.

2.2 The Syntax of Programs

Our first task is to define the set "Program" of possible programs, or in other words, the (abstract) *syntax* of the programs whose semantics we are going to describe. Figure 2.1 shows the definition of "Program" which is based on three

Command Language: Abstract Syntax

$P \in$ Program,
$C \in$ Command,
$E \in$ Expression,
$I \in$ Identifier.

$P ::= C.$
$C ::= I = E \mid \texttt{var}\ I\texttt{;}\ C \mid \texttt{var}\ I\texttt{=}E\texttt{;}\ C \mid C_1\texttt{;}C_2$
$\quad \mid \texttt{if (}E\texttt{)}\ C \mid \texttt{if (}E\texttt{)}\ C_1\ \texttt{else}\ C_2.$
$E ::= \ldots$
$I ::= \ldots$

Figure 2.1: A Command Language

other sets "Command", "Expression", and "Identifier"; these sets are called *syntactic domains* because their elements are (abstract) syntax trees.

Each syntactic domain is introduced by an enumeration of all possible forms that an element of this domain may have. In our language, every $P \in$ Program has only one choice: it is a $C \in$ Command. However for every $C' \in$ Command there exist six possibilities:

I = E C' may be an *assignment* composed of some $I \in$ Identifier and some $E \in$ Expression .

`var` I `;` C C' may be a *declaration block* composed of some $I \in$ Identifier and some $C \in$ Command.

`var` I = E `;` C may be a *definition block* composed of some $I \in$ Identifier, some $E \in$ Expression and some $C \in$ Command.

C_1 `;` C_2 C' may be a *command sequence* composed of some $C_1, C_2 \in$ Command.

`if (E)` C C' may be a *one-sided conditional* composed of some $E \in$ Expression and some $C \in$ Command.

`if (E)` C_1 `else` C_2 C' may be a *two-sided conditional* composed of some $E \in$ Expression and some $C_1, C_2 \in$ Command.

The definition of "Command" is inductive in the sense that it contains some "atomic" elements whose components are not from "Command" ($I = E$) as well

as some "compound" elements that already contain elements from "Command" (e.g. `var` *I*`;` *C*). The meaning of such an inductive definition is taken as the smallest set that contains the atomic elements and all compound elements that can be constructed from other elements of the set.

The inductive definition of syntactic domains is accompanied by a corresponding proving principle: to show that a property P holds for every element $C \in$ Command, it suffices to show that P holds for every possible form of the element. This however may be shown under the additional assumption that P already holds for those components of C that are elements of "Command". For example, for showing that P holds for $C =$ `if (`E`)` C_1 `else` C_2, we may already assume that P holds for C_1 and for C_2.

Figure 2.1 does not elaborate the composition of the elements of the domains "Expression" and "Identifier"; at the moment we are primarily interested in the definition of the semantics of programs respectively their constituting commands.

2.3 The Semantics of Programs

To define the meaning of programs we have to define the meaning of every element of every syntactic domain of the language in which the program is written. For this purpose we introduce for every syntactic domain a valuation function that maps every element of this domain to an element of a corresponding *semantic domain* (a set that holds the meanings of the syntactic elements). As a convention, we denote every valuation function by the name $[\![\_]\!]$; in every application $[\![T]\!]$, the type of the abstract syntax tree T makes clear which valuation function is applied.

For the language of Figure 2.1, we use valuation functions of the following types:

$$\begin{aligned} &\mathit{StateRelation} := \mathbb{P}(\mathit{State} \times \mathit{State}) \\ &\mathit{StateFunction} := \mathit{State} \to \mathit{Value} \end{aligned}$$

$$\begin{aligned} &[\![\_]\!] : \text{Program} \to \mathit{StateRelation} \\ &[\![\_]\!] : \text{Command} \to \mathit{StateRelation} \\ &[\![\_]\!] : \text{Expression} \to \mathit{StateFunction} \\ &[\![\_]\!] : \text{Identifier} \to \mathit{Variable} \end{aligned}$$

Each valuation function is described by a collection of equations, one equation for each possible form of the syntactic argument to which it may be applied. In the case of programs, we have just one form:

$$\begin{aligned} &[\![\_]\!] : \text{Program} \to \mathit{StateRelation} \\ &[\![C]\!] = [\![C]\!] \end{aligned}$$

This equation looks strange because both sides are syntactically equal. However, the left side refers to the valuation function on programs applied to some value $P \in \text{Program}$ (which can have in our language only the form $P = C$), while the right side refers to the valuation function on commands. The former function is thus defined in terms of the later one.

If the result of a valuation function is a relation or a function, we generally prefer to write the definition in a format that makes clear what the result of the application of this relation/function when applied to specific arguments is. For instance, we write above definition as

$$[\![\_]\!] : \text{Program} \to \mathit{StateRelation}$$
$$[\![C]\!](s,s') \Leftrightarrow [\![C]\!](s,s')$$

which states that the meaning of a program $P = C$ is a state relation which holds for two states s and s' if and only if the state relation of command C holds for s and s'.

The definition of the valuation function for commands will make use of the following auxiliary notions:

$$\mathit{read} : \mathit{State} \times \text{Identifier} \to \mathit{Value}$$
$$\mathit{read}(s,I) = s([\![I]\!])$$

$$\mathit{write} : \mathit{State} \times \text{Identifier} \times \mathit{Value} \to \mathit{State}$$
$$\mathit{write}(s,I,v) = s[[\![I]\!] \mapsto v]$$

$$\mathit{writes}(s,I_1,v_1,\ldots,I_n,v_n) \equiv s[[\![I_1]\!] \mapsto v_1]\ldots[[\![I_n]\!] \mapsto v_n]$$

$$s_0 \text{ EQUALS } s_1 \equiv$$
$$\quad \forall I \in \text{Identifier} : \mathit{read}(s_0,I) = \mathit{read}(s_1,I)$$

$$s_0 = s_1 \text{ AT } I_1,\ldots,I_n \equiv$$
$$\quad \forall I \in \text{Identifier} : I = I_1 \vee \ldots \vee I = I_n \Rightarrow \mathit{read}(s_0,I) = \mathit{read}(s_1,I)$$

$$s_0 = s_1 \text{ EXCEPT } I_1,\ldots,I_n \equiv$$
$$\quad \forall I \in \text{Identifier} : I \neq I_1 \wedge \ldots \wedge I \neq I_n \Rightarrow \mathit{read}(s_0,I) = \mathit{read}(s_1,I)$$

$\mathit{read}(s,I)$ is the value of the variable denoted by I that is held in s (remember that a state maps variables to values and that an identifier denotes a variable). The reason that we distinguish between an identifier I (which may appear in a program) and a variable $[\![I]\!]$ (by which a store may be accessed) is that there is not necessarily a one-to-one relationship between both; in general different identifiers

may denote the same variable (i.e. storage location), i.e. one identifier may be an *alias* of another one.

$write(s,I,e)$ denotes the store that is equal to s except that the variable denoted by identifier I is mapped to value v. $writes(s,I_1,v_1,\ldots,I_n,v_n)$ denotes the store that maps the variables denoted by the identifiers $I_1,\ldots,I_n$ to the values $v_1,\ldots,v_n$. The proposition s_0 EQUALS s_1 states that s_0 and s_1 hold the same values for all variables that can be denoted by identifiers. The proposition $s_0 = s_1$ AT $I_1,\ldots,I_n$ states that s_0 and s_1 hold the same values for the variables denoted by identifiers $I_1,\ldots,I_n$. The proposition $s_0 = s_1$ EXCEPT $I_1,\ldots,I_n$ states that s_0 and s_1 are identical except that they may hold different values for the variables denoted by $I_1,\ldots,I_n$. The semantic domains and the corresponding operations are summarized in Figure 2.2.

The definition of the valuation function for commands consists of six equations, one for each possible kind of command (all are summarized in Figure 2.3).

I = E The poststore of an assignment is identical to the prestate except that the variable denoted by the identifier I is mapped the value of expression E in the prestate:

$$\begin{array}{l}[\![I = E]\!](s,s') \Leftrightarrow \\ \quad s' = write(s,I,[\![E]\!](s))\end{array}$$

`var` I; C If a declaration block is executed in a prestate s, the command C is executed in a state identical to s except that I may have a different value; the execution of the command yields a state which is identical to the poststate s' except that s' holds the same value for I as s:

$$\begin{array}{l}[\![\texttt{var}\ I\texttt{;}\ C]\!](s,s') \Leftrightarrow \\ \quad \exists s_0,s_1 \in State : \\ \qquad s_0 = s \text{ EXCEPT } I \wedge [\![C]\!](s_0,s_1) \wedge \\ \qquad\quad s' = write(s_1,I,read(s,I))\end{array}$$

`var` I=E; C The execution of a definition block differs from that of a declaration block only in that the the state in which C is executed holds for I the value of E:

$$\begin{array}{l}[\![\texttt{var}\ I\texttt{=}E\texttt{;}\ C]\!](s,s') \Leftrightarrow \\ \quad \exists s_0,s_1 \in State : \\ \qquad s_0 = write(s,I,[\![E]\!](s)) \wedge [\![C]\!](s_0,s_1) \wedge \\ \qquad\quad s' = write(s_1,I,read(s,I))\end{array}$$

C_1 ; C_2 If a command sequence is executed in a prestate s, command C_1 is executed in that state yielding some state s_0 in which command C_2 is executed yielding the overall poststate s':

Command Language: Semantic Domains and Operations

$Variable := \ldots$
$Value := \ldots$
$Store := Variable \to Value$
$State := Store$
$StateRelation := \mathbb{P}(State \times State)$
$StateFunction := State \to Value$

$read : State \times \text{Identifier} \to Value$
$read(s,I) = s([\![I]\!])$

$write : State \times \text{Identifier} \times Value \to State$
$write(s,I,v) = s[\,[\![I]\!] \mapsto v\,]$

$writes(s,I_1,v_1,\ldots,I_n,v_n) \equiv s[\,[\![I_1]\!] \mapsto v_1\,]\ldots[\,[\![I_n]\!] \mapsto v_n\,]$

s_0 EQUALS $s_1 \equiv$
$\quad \forall I \in \text{Identifier} : read(s_0,I) = read(s_1,I)$

$s_0 = s_1$ AT $I_1,\ldots,I_n \equiv$
$\quad \forall I \in \text{Identifier} : I = I_1 \vee \ldots \vee I = I_n \Rightarrow read(s_0,I) = read(s_1,I)$

$s_0 = s_1$ EXCEPT $I_1,\ldots,I_n \Leftrightarrow$
$\quad \forall I \in \text{Identifier} : I \neq I_1 \wedge \ldots \wedge I \neq I_n \Rightarrow read(s_0,I) = read(s_1,I)$

Figure 2.2: The Semantic Domains of the Command Language

Command Language: Valuation Functions

$[\![\_]\!] : \textbf{Program} \to \textit{StateRelation}$
$[\![C]\!] = [\![C]\!]$

$[\![\_]\!] : \textbf{Command} \to \textit{StateRelation}$
$[\![I = E]\!](s,s') \Leftrightarrow$
  $s' = \mathit{write}(s, I, [\![E]\!](s))$
$[\![\texttt{var } I\texttt{; } C]\!](s,s') \Leftrightarrow$
  $\exists s_0, s_1 \in \mathit{State}:$
    $s_0 = s \text{ EXCEPT } I \wedge [\![C]\!](s_0, s_1) \wedge$
      $s' = \mathit{write}(s_1, I, \mathit{read}(s, I))$
$[\![\texttt{var } I\texttt{=}E\texttt{; } C]\!](s,s') \Leftrightarrow$
  $\exists s_0, s_1 \in \mathit{State}:$
    $s_0 = \mathit{write}(s, I, [\![E]\!](s)) \wedge [\![C]\!](s_0, s_1) \wedge$
      $s' = \mathit{write}(s_1, I, \mathit{read}(s, I))$
$[\![C_1\texttt{; } C_2]\!](s,s') \Leftrightarrow$
  $\exists s_0 \in \mathit{State} : [\![C_1]\!](s, s_0) \wedge [\![C_2]\!](s_0, s')$
$[\![\texttt{if (}E\texttt{) } C]\!](s,s') \Leftrightarrow$
  $\text{IF } [\![E]\!](s) = \text{TRUE THEN } [\![C]\!](s,s') \text{ ELSE } s' = s$
$[\![\texttt{if (}E\texttt{) } C_1 \texttt{ else } C_2]\!](s,s') \Leftrightarrow$
  $\text{IF } [\![E]\!](s) = \text{TRUE THEN } [\![C_1]\!](s,s') \text{ ELSE } [\![C_2]\!](s,s')$

$[\![\_]\!] : \textbf{Expression} \to \textit{StateFunction}$
...

$[\![\_]\!] : \textbf{Identifier} \to \textit{Variable}$
...

Figure 2.3: The Valuation Functions of the Command Language

$$[\![C_1 \texttt{;} C_2]\!](s,s') \Leftrightarrow \\ \quad \exists s_0 \in \mathit{State} : [\![C_1]\!](s,s_0) \wedge [\![C_2]\!](s_0,s')$$

`if (E) C` If a one-sided conditional is executed in a prestate s, then there are two possibilities: if the value of E in s is TRUE, then C is executed in that state yielding the overall poststate s'; otherwise, the state remains unchanged, i.e. s' is identical to s:

$$[\![\texttt{if (}E\texttt{)}\ C]\!](s,s') \Leftrightarrow \\ \quad \text{IF } [\![E]\!](s) = \text{TRUE THEN } [\![C]\!](s,s') \text{ ELSE } s' = s$$

`if (E)` $\mathbf{C_1}$ `else` $\mathbf{C_2}$ If a two-sided conditional is executed, there are two possibilities: if the value of E in s is TRUE, then C_1 is executed in that state yielding the overall poststate s'; otherwise, C_2 is executed in that state and yields s':

$$[\![\texttt{if (}E\texttt{)}\ C_1 \texttt{ else } C_2]\!](s,s') \Leftrightarrow \\ \quad \text{IF } [\![E]\!](s) = \text{TRUE THEN } [\![C_1]\!](s,s') \text{ ELSE } [\![C_2]\!](s,s')$$

As we did not describe the construction of the domains “Expression” and “Identifier”, also the valuation functions for expressions and identifiers are omitted.

2.4 The Specification of Programs

Having defined a programming language, we are now going to introduce a language which allows us to specify the behavior of programs. The essential phrases of this language are *formulas* that describe the relationship between the prestate of a command and its poststate. For this purpose, a formula may contain two kinds of variables: plain variables such as `x` and `y` that refer to the values of program variables in the prestate and primed variables such as `x'` and `y'` that refer to the values of program variables in the poststate. For example, the behavior of the assignment

```
x = x+y
```

can be described by the formula

```
x' = x+y
```

which states that the value of program variable `x` in the poststate equals the sum of the values of `x` and `y` in the prestate. However, there is a catch: above formula does not say anything about the variable `y` which thus might have different values in both states. The behavior of the command is therefore better described as

```
x' = x+y AND y' = y
```

which also states that the value of `y` remains unchanged (the conjunction operator `AND` is used to state that both parts of the formula are true). However, this formula is still not sufficient because the store holds many other variables. Rather than stating for each individual variable that its value remains unchanged, we can write

```
x' = x+y AND writesonly x
```

to denote that the prestate and the poststate may only differ in their values for the variable `x`. This formula is a precise description of the behavior of the assignment `x = x+y`.

Apart from the two kinds of variables denoting the values of program variables in different states, the formulas may also make use of a third kind of variables tagged with the prefix `$`. Such variables, e.g. `$x` or `$y`, denote mathematical variables that receive their denotation not from a program state but from the environment of the formulas in which they occur. For instance, the behavior of the program

```
var y;
x = x+y;
```

can be described the formula

```
(EXISTS $y: x' = x+$y) AND writesonly x
```

In this formula the quantifier `EXISTS` introduces a mathematical variable `$y`. The formula states that there exists some unknown value named `$y` such that the value of the program variable `x` in the poststate equals the sum of the value of `x` in the prestate and this unknown value.

Based on the ideas outlined above, Figure 2.4 describes the complete syntax of the language of formulas that we will use to describe program behaviors. The construction of the syntactic domain "Formula" of *formulas* is based on the domain "Term" of *terms*, the domain "Predicate" of *predicates* and the domain "Function" of *functions*; the later two domains are not further specified. The meanings of these four domains are defined by the following valuation functions that map the syntactic phrases to the semantic domains introduced in Figure 2.5:

$[\![\_]\!]$: Formula $\rightarrow$ *Environment* $\rightarrow$ *StateRelation*
$[\![\_]\!]$: Term $\rightarrow$ *Environment* $\rightarrow$ *StateFunction*
$[\![\_]\!]$: Predicate $\rightarrow$ *Predicate*
$[\![\_]\!]$: Function $\rightarrow$ *Function*

Formula Language: Abstract Syntax

$F \in$ Formula
$T \in$ Term
$p \in$ Predicate
$f \in$ Function

$F ::=$ `TRUE` | `FALSE`
| $p\,(T_1,\ldots,T_n)$ | $T_1 = T_2$ | T_1 `/=` T_2
| `readsonly` | `writesonly` $I_1,\ldots,I_n$
| `!` F | F_1 `AND` F_2 | F_1 `OR` F_2 | F_1 `=>` F_2 | F_1 `<=>` F_2
| F_1 `XOR` F_2 | `IF` F `THEN` F_1 `ELSE` F_2 |
| `FORALL` `$`$I_1,\ldots,$`$I_n$`:` $F$ | `EXISTS` `$`$I_1,\ldots,$`$I_n$`:` F
| `LET` `$`I_1=$T_1,\ldots,$`$I_n$=$T_n$ `IN` F
$T ::= I$ | I' | `$`I | $f\,(T_1,\ldots,T_n)$
| `IF` F `THEN` T_1 `ELSE` T_2
| `LET` `$`I_1=$T_1,\ldots,$`$I_n$=$T_n$ `IN` $T$ | `SUCH` `$`$I$`:` F
$p ::= \ldots$
$f ::= \ldots$

Figure 2.4: A Formula Language

Formula Language: Semantic Domains

ValueEnv := Identifier $\rightarrow$ *Value*
Environment := *ValueEnv*
Predicate := $\mathbb{P}(\mathit{Value}^*)$
Function := $\mathit{Value}^* \rightarrow \mathit{Value}$
BinaryStateFunction := $(\mathit{State} \times \mathit{State}) \rightarrow \mathit{Value}$

Figure 2.5: The Semantic Domains of the Formula Language

Formula Language: Valuation Functions (Formulas)

$[\![\_]\!] : \textbf{Formula} \rightarrow \textit{Environment} \rightarrow \textit{StateRelation}$

$[\![\texttt{TRUE}]\!](e)(s,s') \Leftrightarrow \text{TRUE}$
$[\![\texttt{FALSE}]\!](e)(s,s') \Leftrightarrow \text{FALSE}$
$[\![p\,(T_1,\ldots,T_n)]\!](e)(s,s') \Leftrightarrow [\![p]\!]([\![T_1]\!](e)(s,s'),\ldots,[\![T_n]\!](e)(s,s'))$
$[\![T_1 = T_2]\!](e)(s,s') \Leftrightarrow [\![T_1]\!](e)(s,s') = [\![T_2]\!](e)(s,s')$
$[\![T_1 \texttt{ /= } T_2]\!](e)(s,s') \Leftrightarrow [\![T_1]\!](e)(s,s') \neq [\![T_2]\!](e)(s,s')$
$[\![\texttt{readsonly}]\!](e)(s,s') \Leftrightarrow s = s'$
$[\![\texttt{writesonly}\ I_1,\ldots,I_n]\!](e)(s,s') \Leftrightarrow s = s' \text{ EXCEPT } I_1,\ldots,I_n$
$[\![\texttt{!}\,F]\!](e)(s,s') \Leftrightarrow \neg [\![F]\!](e)(s,s')$
$[\![F_1 \texttt{ AND } F_2]\!](e)(s,s') \Leftrightarrow [\![F_1]\!](e)(s,s') \wedge [\![F_2]\!](e)(s,s')$
$[\![F_1 \texttt{ OR } F_2]\!](e)(s,s') \Leftrightarrow [\![F_1]\!](e)(s,s') \vee [\![F_2]\!](e)(s,s')$
$[\![F_1 \texttt{ => } F_2]\!](e)(s,s') \Leftrightarrow [\![F_1]\!](e)(s,s') \Rightarrow [\![F_2]\!](e)(s,s')$
$[\![F_1 \texttt{ <=> } F_2]\!](e)(s,s') \Leftrightarrow [\![F_1]\!](e)(s,s') \Leftrightarrow [\![F_2]\!](e)(s,s')$
$[\![F_1 \texttt{ XOR } F_2]\!](e)(s,s') \Leftrightarrow [\![F_1]\!](e)(s,s') \not\Leftrightarrow [\![F_2]\!](e)(s,s')$
$[\![\texttt{IF } F \texttt{ THEN } F_1 \texttt{ ELSE } F_2]\!](e)(s,s') \Leftrightarrow$
  IF $[\![F]\!](e)(s,s')$ THEN $[\![F_1]\!](e)(s,s')$ ELSE $[\![F_2]\!](e)(s,s')$
$[\![\texttt{FORALL \$}I_1,\ldots,\texttt{\$}I_n \texttt{:}\ F]\!](e)(s,s') \Leftrightarrow$
  $\forall v_1,\ldots,v_n \in \mathit{Value} : [\![F]\!](e[I_1 \mapsto v_1,\ldots,I_n \mapsto v_n])(s,s')$
$[\![\texttt{EXISTS \$}I_1,\ldots,\texttt{\$}I_n \texttt{:}\ F]\!](e)(s,s') \Leftrightarrow$
  $\exists v_1,\ldots,v_n \in \mathit{Value} : [\![F]\!](e[I_1 \mapsto v_1,\ldots,I_n \mapsto v_n])(s,s')$
$[\![\texttt{LET \$}I_1\texttt{=}T_1,\ldots,\texttt{\$}I_n\texttt{=}T_n \texttt{ IN } F]\!](e)(s,s') \Leftrightarrow$
  LET
    $e_1 = e[I_1 \mapsto [\![T_1]\!](e)(s,s')]$
    $\ldots$
    $e_n = e_{n-1}[I_n \mapsto [\![T_n]\!](s,s',e_{n-1})]$
  IN $[\![F]\!](s,s',e_n)$

$[\![\_]\!] : \textbf{Predicate} \rightarrow \textit{Predicate}$
$\ldots$

Figure 2.6: The Valuation Functions of the Formula Language (Formulas)

Formula Language: Valuation Functions (Terms)

$[\![\_]\!] : \textit{Term} \to \textit{Environment} \to \textit{BinaryStateFunction}$

$[\![I]\!](e)(s,s') = \textit{read}(s,I)$

$[\![I']\!](e)(s,s') = \textit{read}(s',I)$

$[\![\$I]\!](e)(s,s') = e(I)$

$[\![f\,(T_1,\ldots,T_n)]\!](e)(s,s') = [\![f]\!]([\![T_1]\!](e)(s,s'),\ldots,[\![T_n]\!](e)(s,s'))$

$[\![\texttt{IF}\ F\ \texttt{THEN}\ T_1\ \texttt{ELSE}\ T_2]\!](e)(s,s') =$

IF $[\![F]\!](e)(s,s')$ THEN $[\![T_1]\!](e)(s,s')$ ELSE $[\![T_2]\!](e)(s,s')$

$[\![\texttt{LET}\ \$I_1{=}T_1,\ldots,\$I_n{=}T_n\ \texttt{IN}\ T]\!](e)(s,s') \Leftrightarrow$

LET

$e_1 = e[I_1 \mapsto [\![T_1]\!](e)(s,s')]$

$\ldots$

$e_n = e_{n-1}[I_n \mapsto [\![T_n]\!](s,s',e_{n-1})]$

IN $[\![T]\!](s,s',e_n)$

$[\![\texttt{SUCH}\ \$I\texttt{:}\ F]\!](e)(s,s') = \text{SUCH}\ v \in \textit{Value} : [\![F]\!](s,s',e[I \mapsto v])$

$[\![\_]\!] : \textbf{Function} \to \textit{Function}$

$\ldots$

Figure 2.7: The Valuation Functions of the Formula Language (Terms)

According to these declarations, a formula denotes a mapping from a value environment to a state relation (the meaning of a program command) while a term denotes a function from an environment to a state function (the meaning of a program expression). A predicate denotes a relation on value sequences and a function denotes a map from value sequences to values.

The definitions of the valuation functions for the various kinds of formulas and terms are given in Figures 2.6 and 2.7; some further explanation is given below.

Formulas Most kinds of formulas are known in classical predicate logic and take their standard interpretation. The exceptions are:

`readsonly` This formula states that all program variables have the same value in the prestate and in the poststate.

`writesonly` $\mathbf{I_1}, \ldots, \mathbf{I_n}$ This formula states that only the program variables $I_1, \ldots, I_n$ may have different values in the pres-state and in the post-state (while all other program variables have the same value).

`unchanged` $\mathbf{I_1}, \ldots, \mathbf{I_n}$ This formula states that the program variables denoted by $I_1, \ldots, I_n$ have the same values in the pres-state and in the post-state (while all other variables may have different values).

Please note how the quantifiers `ALL`, `EX`, and `LET` introduce mathematical variables by defining their interpretations in the environment in which the body formula is evaluated..

Terms Most kinds of formulas are known in classical predicate logic and take their standard interpretation. The exceptions are:

I A plain identifier denotes the value of the corresponding program variable in the pre-state.

I′ A primed identifier denotes the value of the corresponding program variable in the post-state.

`$I` An identifier qualified by the prefix `$` denotes the value of the corresponding mathematical variable in the environment of the formula.

Similar to the formula quantifiers, the term quantifiers `LET` and `SUCH` introduce mathematical variables by defining their interpretations in the environment in which the body term is evaluated.

2.5 The Verification of Programs

Using the command language and the formula language introduced in the previous sections, our goal is now to develop a verification calculus which consists of rules for deriving *judgements* of the form

$$C : F$$

where C is a command and F is a formula. Such a judgement can be read as "command C implements specification F" or vice versa as "formula F describes the behavior of command C". The semantic interpretation of this judgement is

$$\forall s, s' \in \mathit{State}, e \in \mathit{Environment} : [\![C]\!](s, s') \Rightarrow [\![F]\!](e)(s, s')$$

i.e. only such pairs of states are related by C that are also related by F (for any environment e). We will later show that the presented calculus is (under some additional assumption) *sound* in the sense that only such judgements can be derived for which this interpretation is true.

Every *(derivation) rule* of the calculus has form

$$\frac{\begin{array}{l}\mathit{premise}_1 \\ \ldots \\ \mathit{premise}_n\end{array}}{\mathit{conclusion}}$$

Such a rule states that the judgement *conclusion* can be derived, if the judgements $\mathit{premise}_1, \ldots, \mathit{premise}_n$ can be derived, again by the application of the rules of the calculus. The derivation of a judgement can thus be depicted as the construction of a *derivation tree*

$$\dfrac{\dfrac{\dfrac{L_{\ldots}}{\ldots}\quad\dfrac{L_{\ldots}}{\ldots}}{J_{11}\ \ldots\ J_{1m_1}}}{J_1}\quad\ldots\quad\dfrac{\dfrac{\dfrac{L_{\ldots}}{\ldots}\quad\dfrac{L_{\ldots}}{\ldots}}{J_{n1}\ \ldots\ J_{nm_n}}}{J_n}}{J}$$

whose root J is the judgement to be derived and where each node $\dfrac{J_{i1}, \ldots, J_{im_i}}{J_i}$ corresponds to the application of some rule whose conclusion matches J_i and whose premises match $J_{i1}, \ldots, J_{im_i}$. The leaves $L_{\ldots}$ of this tree are judgements that match *axioms*, i.e. derivation rules that have no premises.

Verification Calculus: Definitions

$[F]_{I_1,\ldots,I_n} \equiv (F)$ AND writesonly $I_1,\ldots,I_n$

$\llcorner \simeq \lrcorner : \mathbb{P}(\text{Expression} \times \text{Term})$
$E \simeq T \Leftrightarrow$
T has no free variables $\wedge$
T has no primed program variables $\wedge$
$\forall s, s' \in \mathit{Store}, e \in \mathit{Environment} : [\![E]\!](s) = [\![T]\!](e)(s,s')$

$\llcorner \simeq \lrcorner : \mathbb{P}(\text{Expression} \times \text{Formula})$
$E \simeq F \Leftrightarrow$
F has no free variables $\wedge$
F has no primed program variables $\wedge$
$\forall s, s' \in \mathit{Store}, e \in \mathit{Environment} :$
$[\![E]\!](s) = \text{TRUE} \Leftrightarrow [\![F]\!](e)(s,s')$

Verification Calculus: Judgements

$C : F \Leftrightarrow$
$\forall s, s' \in \mathit{State}, e \in \mathit{Environment} : [\![C]\!](s,s') \Rightarrow [\![F]\!](e)(s,s')$

Figure 2.8: The Verification Calculus of the Command Language (Part 1 of 3)

Verification Calculus: Rules

$$\frac{C : [F]_{I_1,\ldots,I_n} \quad \text{p is a permutation of } \{1,\ldots,n\}}{C : [F]_{I_{p(1)},\ldots,I_{p(n)}}}$$

$$\frac{C : [F]_{I_1,\ldots,I_n} \quad I \neq I_1 \wedge \ldots \wedge I \neq I_n}{C : [F \texttt{ AND } I' = I]_{I_1,\ldots,I_n,I}}$$

$$\frac{E \simeq T}{I = E : [I' = T]_I}$$

$$\frac{C : [F]_{I_1,\ldots,I_n,I} \quad I_a \neq I_b \quad \$I_a \text{ and } \$I_b \text{ do not occur in } F}{\texttt{var } I\texttt{; } C : \; [\texttt{EXISTS } \$I_a\texttt{, } \$I_b : F[\$I_a/I, \$I_b/I']]_{I_1,\ldots,I_n}}$$

$$\frac{C : [F]_{I_1,\ldots,I_n,I} \quad I_a \neq I_b \quad \$I_a \text{ and } \$I_b \text{ do not occur in } F \quad E \simeq T}{\texttt{var } I\texttt{=}E\texttt{; } C : \; [\texttt{EXISTS } \$I_a\texttt{, } \$I_b : \$I_a\texttt{=}T \texttt{ AND } F[\$I_a/I, \$I_b/I']]_{I_1,\ldots,I_n}}$$

Figure 2.9: The Verification Calculus of the Command Language (Part 2 of 3)

Verification Calculus: Rules (Continued)

$$\frac{C_1 : [F_1]_{I_1,\ldots,I_n} \quad C_2 : [F_2]_{I_1,\ldots,I_n} \quad \$I_1,\ldots,\$I_n \text{ do not occur in } F_1 \text{ and } F_2}{C_1 ; C_2 : \\ \quad [\texttt{EXISTS } \$I_1,\ldots,\$I_n : \\ \quad\quad F_1[\$I_1/I_1\texttt{'},\ldots,\$I_n/I_n\texttt{'}] \texttt{ AND } F_2[\$I_1/I_1,\ldots,\$I_n/I_n]]_{I_1,\ldots,I_n}}$$

$$\frac{C : [F]_{I_1,\ldots,I_n} \quad E \simeq F_0}{\texttt{if (}E\texttt{) } C : [\texttt{IF } F_0 \texttt{ THEN } F \texttt{ ELSE readsonly}]_{I_1,\ldots,I_n}}$$

$$\frac{C_1 : [F_1]_{I_1,\ldots,I_n} \quad C_2 : [F_2]_{I_1,\ldots,I_n} \quad E \simeq F_0}{\texttt{if (}E\texttt{) } C_1 \texttt{ else } C_2 : [\texttt{IF } F_0 \texttt{ THEN } F_1 \texttt{ ELSE } F_2]_{I_1,\ldots,I_n}}$$

Figure 2.10: The Verification Calculus of the Command Language (Part 2 of 3)

Figures 2.8, 2.9, and 2.10 presents the verification calculus by listing the form of judgements, their interpretation, and the rules for deriving judgements. The individual rules will be explained later; for the moment it suffices to note that only judgements of the form

$$C : [F]_{I_1,\ldots,I_n}$$

can be derived where the formula $[F]_{I_1,\ldots,I_n}$ is a syntactic abbreviation of the formula (F) `AND writesonly` $I_1,\ldots,I_n$. The set $\{I_1,\ldots,I_n\}$ is called the *specification frame*; it describes which variables the command may change. In the presented calculus, the frame is thus an inherent part of every specification.

Some premises in the rules have form $E \simeq T$ respectively $E \simeq F$ where E is an expression of the programming language and T respectively F is a term respectively formula of the formula language that is closed and only refers to the prestate. According to the definition of $\simeq$ given in Figure 2.8, this means that both E and T denote the same value in any state respectively that E yields `TRUE` in a state if and only if F is true in that state (since T respectively F are closed, their values are not influenced by the environment respectively poststate).

The main reason that we distinguish between program expressions and mathematical terms is that in general they are of different expressive power and may also

differ in their interpretations of operations. For example, the program expression `x+y` typically denotes the addition of two computer words using 32 bit arithmetic; the identical mathematical term typically denotes plain addition. An appropriate translation to a mathematical term then actually is

```
SUCH $z: LET $n=2^32 IN
  -$n/2<=$z AND $z<$n/2 AND $n|(x+y-$z)
```

The individual rules of the verification calculus will be further explained in the following section. For the moment it suffices to note that the rules allow a bottom-up construction of the specification of a command from the specifications of its subcommands starting with the specifications derived from the atomic commands. For instance, to construct the specification of a program

```
y = y+1; if (x < y) x = x+y else y = x+1
```

we first construct the specifications of the atomic commands

$$\texttt{y=y+1}:[\texttt{y'=y+1}]_{\texttt{y}}$$
$$\texttt{x=x+y}:[\texttt{x'=x+y}]_{\texttt{x}}$$
$$\texttt{y=x+1}:[\texttt{y'=x+1}]_{\texttt{y}}$$

The specifications can be rewritten to include a common frame

$$\texttt{y=y+1}:[\texttt{y'=y+1 AND x'=x}]_{\texttt{x,y}}$$
$$\texttt{x=x+y}:[\texttt{x'=x+y AND y'=y}]_{\texttt{x,y}}$$
$$\texttt{y=x+1}:[\texttt{y'=x+1 AND x'=x}]_{\texttt{x,y}}$$

The specifications of the conditional branches can be composed to the specification of the conditional statement itself:

```
if (x < y) x = x+y else y = x+1:
  [IF x < y
     THEN x'=x+y AND y'=y
     ELSE y'=x+1 AND x'=x]x,y
```

This specification can be composed with the specification of `y=y+1` to the specification of the program

```
y = y+1; if (x < y) x = x+y else y = x+1:
  [EXISTS $x, $y:
     $y=y+1 AND $x=x AND
     IF $x < $y
       THEN x'=$x+$y AND y'=$y
       ELSE y'=$x+1 AND x'=$x]x,y
```

which can be further simplified by conventional logical reasoning to

```
y = y+1; if (x < y) x = x+y else y = x+1 :
  [IF x < y+1
     THEN x'=x+(y+1) AND y'=y+1
     ELSE y'=x+1 AND x'=x]x,y
```

This makes the state relation expressed by the program very explicit.

One should also note that by the rules of the calculus only *closed* specification formulas can be derived i.e. only formulas that do not not have any occurrence of a mathematical variable $I outside the scope of the quantifiers FORALL $I, EXISTS $I, LET $I=..., or SUCH $I.

Lemma (Closed Specifications) If a judgement $C : F$ can be derived by the rules of the verification calculus of the command language, then F is closed, i.e. does not contain free occurrences of mathematical variables.

Proof The proof of the lemma proceeds by induction on the derivation of $C : F$. All derived specification formulas have form $[F]_{I_1,\ldots,I_n}$ which abbreviates F AND writesonly $I_1,\ldots,I_n$. Since the writesonly formula does not refer to any mathematical variables, we can focus on the first part of the formula.

We now distinguish which rule matches the last step of this derivation:

Permutation The formula F of the conclusion is derived from a premise from which, by the induction hypothesis, we can conclude that F is closed.

Frame Extension The formula F AND $I' = I$ can only have mathematical variables in F. F is derived from a premise from which, by the induction hypothesis, we can conclude that F is closed.

Assignment The formula $I'=T$ can only have mathematical variables in T which is derived from the premise $E \simeq T$; by the definition of $\simeq$, T is closed.

Variable Declaration The formula EXISTS $\$I_a$, $\$I_b : F[\$I_a/I, \$I_b/I']$ can only have free occurrences of mathematical variables that are also free in F. F is derived from a premise from which, by the induction hypothesis, we can conclude that F is closed.

Variable Definition Formula EXISTS $\$I_a$, $\$I_b$: $\$I_a=T$ AND $F[\$I_a/I, \$I_b/I']$ can only have free occurrences of mathematical variables that are also free

in T or in F; T is derived from the premise $E \simeq T$; by the definition of $\simeq$, T is closed. F is derived from a premise from which, by the induction hypothesis, we can conclude that F is closed.

Command Sequence The specification formula

$$\texttt{EXISTS } \$I_1, \ldots, \$I_n : \\ F_1[\$I_1/I_1\texttt{'}, \ldots, \$I_n/I_n\texttt{'}] \texttt{ AND } F_2[\$I_1/I_1, \ldots, \$I_n/I_n]$$

can only have free occurrences of mathematical variables that are also free in F_1 or in F_2. Both formulas are derived from premises from which, by the induction hypothesis, we can conclude that the formulas are closed.

One-Sided Conditional The formula `IF` T `THEN` F `ELSE readsonly` can only have free occurrences of mathematical variables that are also free in T or in F; T is derived from the premise $E \simeq T$; by the definition of $\simeq$, T is closed. F is derived from a premise; by induction hypothesis, F is closed.

Two-Sided Conditional The formula `IF` T `THEN` F_1 `ELSE` F_2 can only have free occurrences of mathematical variables that are also free in T or in F_1 or in F_2. T is derived from the premise $E \simeq T$; by the definition of $\simeq$, T is closed. F_1 and F_2 are derived from premises from which, by the induction hypothesis, we can conclude that they are closed. □

2.6 Soundness of the Verification Calculus

We are now going to state that the presented calculus is indeed sound with respect to the intended interpretation of its judgements. However, the soundness is stated relative to the property

$$\mathit{DifferentVariables} :\Leftrightarrow \forall I_1, I_2 \in \text{Identifier} : I_1 \neq I_2 \Rightarrow [\![I_1]\!] \neq [\![I_2]\!]$$

which says that different program identifiers denote different storage variables. The truth of *DifferentVariables* depends on the definition of the valuation function $[\![\_]\!] : \text{Identifier} \to \mathit{Variable}$ (which we have deliberately left open).

Theorem (Soundness of the Verification Calculus) Under the assumption denoted by *DifferentVariables*, the following is true: if a judgement $C : F$ can be derived from the rules of the verification calculus of the command language, then

$$\forall s, s' \in \mathit{State}, e \in \mathit{Environment} : [\![C]\!](s, s') \Rightarrow [\![F]\!](e)(s, s')$$

In other words, if different identifiers denote different variables, then the derivation of $C : F$ implies that command C implements specification F.

Proof We assume

(1) *DifferentVariables*

Take C and F such that $C : F$ can be derived and take arbitrary $s, s' \in State, e \in Environment$. We prove

$$[\![C]\!](s,s') \Rightarrow [\![F]\!](e)(s,s')$$

by induction on the derivation of $C : F$. The following subsections cover all cases for the last step of such a derivation; each subsection essentially shows the soundness of one derivation rule, i.e. that the interpretation of its conclusion is true under the assumption that the interpretations of its premises are true. □

The subsequent proofs will make use of various lemmas stated in Appendix B that describe properties of states/stores and of syntactic phrases (formulas and terms).

2.6.1 Frame Permutation

$$\frac{C : [F]_{I_1,\ldots,I_n} \qquad \text{p is a permutation of } \{1,\ldots,n\}}{C : [F]_{I_{p(1)},\ldots,I_{p(n)}}}$$

This rule states that the identifiers of a frame may be permuted in an arbitrary way. For example, from the judgement

$$\texttt{x=x+y} : [\texttt{x'=x+y}]_{\texttt{x,y}}$$

also the following judgement can be derived:

$$\texttt{x=x+y} : [\texttt{x'=x+y}]_{\texttt{y,x}}$$

Soundness Proof We have to show

(a) $[\![C]\!](s,s') \Rightarrow [\![[F \texttt{ AND } I' = I]_{I_{p(1)},\ldots,I_{p(n)}}]\!](e)(s,s')$

Assume

(2) $[\![C]\!](s,s')$

By the definition of $[\_]_\_$, we have to show

(b) $[\![F \texttt{ AND writesonly } I_{p(1)}, \ldots, I_{p(n)}]\!](e)(s,s')$

By the definition of $[\![\_]\!]$, it suffices to show

(b.1) $[\![F]\!](e)(s,s')$

(b.2) $s = s' \text{ EXCEPT } I_{p(1)}, \ldots, I_{p(n)}$

From the premises we know

(3) $\forall s,s' \in \mathit{State}, e \in \mathit{Environment} : [\![C]\!](s,s') \Rightarrow [\![[F]_{I_1,\ldots,I_n}]\!](e)(s,s')$

(4) p is a permutation of $\{1,\ldots,n\}$

From (2), (3), and the definition of $[\_]_{\_}$ we know

(5) $[\![F \texttt{ AND writesonly } I_1, \ldots, I_n]\!](e)(s,s')$

By the definition of $[\![\_]\!]$, we thus know

(6) $[\![F]\!](e)(s,s')$

(7) $s = s' \text{ EXCEPT } I_1, \ldots, I_n$

From (6), we know (b.1). From (4), (7), and (SVE), we also know (b.2). □

2.6.2 Frame Extension

$$\frac{C : [F]_{I_1,\ldots,I_n} \quad I \neq I_1 \wedge \ldots \wedge I \neq I_n}{C : [F \texttt{ AND } I' = I]_{I_1,\ldots,I_n,I}}$$

This rule states that the frame of a specification may be extended by an identifier I not present in the frame, if the condition $I' = I$ is added to the specification formula. For example, from the judgement

$$\texttt{x=x+y} : [\texttt{x'=x+y}]_{\texttt{x,y}}$$

also the following judgement can be derived:

$$\texttt{x=x+y} : [\texttt{x'=x+y AND z'=z}]_{\texttt{x,y,z}}$$

Soundness Proof We have to show

(a) $[\![C]\!](s,s') \Rightarrow [\![[F \texttt{ AND } I' = I]_{I_1,\ldots,I_n,I}]\!](e)(s,s')$

Assume

(2) $[\![C]\!](s,s')$

By the definition of $[\_]_\_$, we have to show

(b) $[\![(F \texttt{ AND } I' = I) \texttt{ AND writesonly } I_1,\ldots,I_n,I]\!](e)(s,s')$

By the definition of $[\![\ ]\!]$, it suffices to show

(b.1) $[\![F]\!](e)(s,s')$

(b.2) $read(s',I) = read(s,I)$

(b.3) $s = s' \text{ EXCEPT } I_1,\ldots,I_n,I$

From the premises we know

(3) $\forall s,s' \in State, e \in Environment : [\![C]\!](s,s') \Rightarrow [\![[F]_{I_1,\ldots,I_n}]\!](e)(s,s')$

(4) $I \neq I_1 \wedge \ldots \wedge I \neq I_n$

From (2), (3), and the definition of $[\_]_\_$ we know

(5) $[\![F \texttt{ AND writesonly } I_1,\ldots,I_n]\!](e)(s,s')$

By the definition of $[\![\_]\!]$, we thus know

(6) $[\![F]\!](e)(s,s')$

(7) $s = s' \text{ EXCEPT } I_1,\ldots,I_n$

From (6), we know (b.1). From (4), (7), and (RSE), we know (b.2). From (7) and (AVE), we also know (b.3). □

2.6.3 Assignment

$$\frac{E \simeq T}{I = E : [I' = T]_I}$$

This rule describes the construction of a specification for the assignment of the value a program expression E to a variable I. It states that the value of I in the poststate equals the value of T where T is a term of the formula language describing the value of E. For instance, we may have

$$\texttt{x=x+y} : [\texttt{x'=ADD32(x,y)}]_{\texttt{x}}$$

if the operator + in the programming language and and the operator `ADD32` in the formula language denote the same function.

Soundness Proof We have to show

(a) $[\![I = E]\!](s,s') \Rightarrow [\![[I' = T]_I]\!](e)(s,s')$

Assume

(2) $[\![I = E]\!](s,s')$

By the definitions of $[\_]_\_$ and $[\![\_]\!]$, we have to show

(a.1) $read(s',I) = [\![T]\!](e)(s,s')$

(a.2) $s = s'$ EXCEPT I

From (2), we know by the definition of $[\![\_]\!]$

(3) $s' = write(s,I,[\![E]\!](s))$

From the premise and the definition of $\simeq$, we know

(4) $[\![E]\!](s) = [\![T]\!](e)(s,s')$

From (3) and (4), we know

(5) $s' = write(s,I,[\![T]\!](e)(s,s'))$

From (5) and (RW1), we know (a.1). From *DifferentVariables*, (3), and (WS), we know (a.2). □

2.6.4 Variable Declaration

$$\frac{\begin{array}{l} C : [F]_{I_1,\ldots,I_n,I} \\ I_a \neq I_b \\ \texttt{\$}I_a \text{ and } \texttt{\$}I_b \text{ do not occur in } F \end{array}}{\begin{array}{l} \texttt{var } I\texttt{; } C : \\ \quad [\texttt{EXISTS \$}I_a\texttt{, \$}I_b : F[\texttt{\$}I_a/I, \texttt{\$}I_b/I']]_{I_1,\ldots,I_n} \end{array}}$$

This rule shows how the specification of a declaration of a program variable I can be constructed from the specification of its base command: in the specification formula, any occurrence I respectively I' referring to the prestate/poststate value of I has to be replaced by an occurrence of a mathematical variable $\texttt{\$}I_a$ respectively $\texttt{\$}I_b$ that is existentially quantified. For instance, from the judgement

```
y=y*y; x=x+y : [y'=y*y AND x'=x+y']x,y
```

the following judgement can be derived:

```
var y; y=y*y; x=x+y :
  [EXISTS $y0,$y1:
     $y1=$y0*$y0 AND x'=x+$y1]x
```

Soundness Proof We have to show

(a) $[\![\texttt{var}\ I\texttt{;}\ C]\!](s,s') \Rightarrow$
$\quad [\![[\texttt{EXISTS}\ \$I_a\texttt{,}\ \$I_b : F[\$I_a/I, \$I_b/I']]_{I_1,\ldots,I_n}]\!](e)(s,s')$

Assume

(2) $[\![\texttt{var}\ I\texttt{;}\ C]\!](s,s')$

By the definitions of $[\_]_\_$ and $[\![\_]\!]$, we have to show

(a.1) $[\![\texttt{EXISTS}\ \$I_a\texttt{,}\ \$I_b : F[\$I_a/I, \$I_b/I']]\!](e)(s,s')$
(a.2) $s = s'$ EXCEPT $I_1,\ldots,I_n$

From (2) and the definition of $[\![\_]\!]$, we know

(3) $\exists s_0, s_1 \in \mathit{State}:$
$\quad s_0 = s$ EXCEPT $I \wedge [\![C]\!](s_0,s_1) \wedge s' = \mathit{write}(s_1, I, \mathit{read}(s,I))$

Let $s_0, s_1 \in \mathit{State}$ be such that

(4) $s_0 = s$ EXCEPT I
(5) $[\![C]\!](s_0,s_1)$
(6) $s' = \mathit{write}(s_1, I, \mathit{read}(s,I))$

From the premises, we know

(7) $[\![C]\!](s_0,s_1) \Rightarrow [\![[F]_{I_1,\ldots,I_n,I}]\!](e)(s_0,s_1)$
(8) $I_a \neq I_b$
(8a) $\$I_a$ and $\$I_b$ do not occur in F

and by Lemma "Closed Specifications"

(9) F has no free variables

From (5), (7), and the definitions of $[\![\_]\!]$ and $[\_]_\_$, we know

(10) $[\![F]\!](e)(s_0,s_1)$
(11) $s_1 = s_0$ EXCEPT $I_1,\ldots,I_n,I$

From *DifferentVariables*, (6), and (WS), we know

(12) $s' = s_1$ EXCEPT I

To show (a.1), we have by the definition of $[\![\_]\!]$ to show

(a.1.a) $\exists v_a, v_b \in \mathit{Value} : [\![F[\$I_a/I, \$I_b/I']]\!](e[I_a \mapsto v_a, I_b \mapsto v_b])(s, s')$

It thus suffices to show

(a.1.b) $[\![F[\$I_a/I, \$I_b/I']]\!](e[I_a \mapsto \mathit{read}(s_0, I), I_b \mapsto \mathit{read}(s_1, I)])(s, s')$

This is true from (4), (8), (8a), (10), (12), (PMVF1), and (PMVF2).

From (6) and (RW1), we know

(13) $\mathit{read}(s', I) = \mathit{read}(s, I)$

From (4) and (AVE), we know

(14) $s_0 = s$ EXCEPT $I_1, \ldots, I_n, I$

From (12) and (AVE), we know

(15) $s' = s_1$ EXCEPT $I_1, \ldots, I_n, I$

From (11), (14), (15), and (TRE), we know

(16) $s = s'$ EXCEPT $I_1, \ldots, I_n, I$

From (13), (16), and (RVE), we know (a.2). □

2.6.5 Variable Definition

$$\frac{\begin{array}{l} C : [\,F\,]_{I_1,\ldots,I_n,I} \\ I_a \neq I_b \\ \$I_a \text{ and } \$I_b \text{ do not occur in } F \\ E \simeq T \end{array}}{\begin{array}{l} \texttt{var}\ I\texttt{=}E\texttt{;}\ C : \\ \quad [\texttt{EXISTS}\ \$I_a\texttt{,}\ \$I_b : \$I_a\texttt{=}T\ \texttt{AND}\ F[\$I_a/I, \$I_b/I']]_{I_1,\ldots,I_n} \end{array}}$$

This rule shows how the specification of a definition of a program variable I by the value of a program expression E can be constructed from the specification of its base command: in the specification formula, any occurrence I respectively I' referring to the prestate/poststate value of I has to be replaced by an occurrence of a mathematical variable $\$I_a$ that is defined by a mathematical term T with the same value as E respectively by a mathematical variable $\$I_b$ that is existentially quantified. For instance, from the judgements

```
y/2 ≃ DIV32(y,2)
y=y*y; x=x+y : [y'=y*y AND x'=x+y']x,y
```

the following judgement can be derived:

```
var y = z/2; y=y*y; x=x+y:
  [EXISTS $y0,$y1: $y0 = DIV32(y,2) AND
     $y1=$y0*$y0 AND x'=x+$y1]x
```

Soundness Proof We have to show

(a) $[\![\texttt{var}\ I\texttt{=}E\texttt{;}\ C]\!](s,s') \Rightarrow$
$[\![[\texttt{EXISTS}\ \$I_a\texttt{,}\ \$I_b : \$I_a\texttt{=}T\ \texttt{AND}\ F[\$I_a/I, \$I_b/I']]_{I_1,\ldots,I_n}]\!](e)(s,s')$

Assume

(2) $[\![\texttt{var}\ I\texttt{=}E\texttt{;}\ C]\!](s,s')$

By the definitions of $[\_]_\_$ and $[\![\_]\!]$, we have to show

(a.1) $[\![\texttt{EXISTS}\ \$I_a\texttt{,}\ \$I_b : \$I_a\texttt{=}T\ \texttt{AND}\ F[\$I_a/I, \$I_b/I']]\!](e)(s,s')$

(a.2) $s = s'$ EXCEPT $I_1,\ldots,I_n$

From (2) and the definition of $[\![\_]\!]$, we know

(3) $\exists s_0, s_1 \in \mathit{State} :$
$s_0 = \mathit{write}(s,I,[\![E]\!](s)) \wedge [\![C]\!](s_0,s_1) \wedge s' = \mathit{write}(s_1,I,\mathit{read}(s,I))$

Let $s_0, s_1 \in \mathit{State}$ such that

(4) $s_0 = \mathit{write}(s,I,[\![E]\!](s))$

(5) $[\![C]\!](s_0,s_1)$

(6) $s' = \mathit{write}(s_1,I,\mathit{read}(s,I))$

From the premises, we know

(7) $[\![C]\!](s_0,s_1) \Rightarrow [\![[F]_{I_1,\ldots,I_n,I}]\!](e)(s_0,s_1)$

(8) $I_a \neq I_b$

(8a) $\$I_a$ and $\$I_b$ do not occur in F

(9) $E \simeq T$

and by Lemma “Closed Specifications”

(10) F has no free variables

From (5), (7), and the definitions of $[\![\_]\!]$ and $[\_]\_$, we know

(11) $[\![F]\!](e)(s_0, s_1)$

(12) $s_1 = s_0$ EXCEPT $I_1, \ldots, I_n, I$

From *DifferentVariables*, (4), (6), and (WS), we know

(13) $s_0 = s$ EXCEPT I

(14) $s' = s_1$ EXCEPT I

To show (a.1), we have by the definition of $[\![\_]\!]$ to show

(a.1.a)
$$\begin{array}{l} \exists v_a, v_b \in \mathit{Value} : \\ \quad v_a = [\![T]\!](e)(s, s') \wedge \\ \quad [\![F[\$I_a/I, \$I_b/I']]\!](e[I_a \mapsto v_a, I_b \mapsto v_b])(s, s') \end{array}$$

From (8), (8a), (11), (13), (14), (PMVF1), and (PMVF2), we know

(15) $[\![F[\$I_a/I, \$I_b/I']]\!](e[I_a \mapsto \mathit{read}(s_0, I), I_b \mapsto \mathit{read}(s_1, I)])(s, s')$

From (4) and (RW1), we know

(16) $\mathit{read}(s_0, I) = [\![E]\!](s)$

From (9) and the definition of $\simeq$, we know

(17) $[\![E]\!](s) = [\![T]\!](e)(s, s')$

From (15), (16), and (17) we know

(18) $[\![F[\$I_a/I, \$I_b/I']]\!](e[I_a \mapsto [\![T]\!](e)(s, s'), I_b \mapsto \mathit{read}(s_1, I)])(s, s')$

and thus (a.1.a).

From (6) and (RW1), we know

(19) $\mathit{read}(s', I) = \mathit{read}(s, I)$

From (13) and (AVE), we know

(20) $s_0 = s$ EXCEPT $I_1, \ldots, I_n, I$

From (14) and (AVE), we know

(21) $s' = s_1$ EXCEPT $I_1, \ldots, I_n, I$

From (12), (20), (21), and (TRE), we know

(22) $s = s'$ EXCEPT $I_1, \ldots, I_n, I$

From (19), (22), and (RVE), we know (a.2). □

2.6.6 Command Sequence

$$\frac{\begin{array}{l} C_1 : [F_1]_{I_1,\ldots,I_n} \\ C_2 : [F_2]_{I_1,\ldots,I_n} \\ \$I_1,\ldots,\$I_n \text{ do not occur in } F_1 \text{ and } F_2 \end{array}}{\begin{array}{l} C_1\,\texttt{;}\,C_2 : \\ \quad [\texttt{EXISTS } \$I_1,\ldots,\$I_n : \\ \quad\quad F_1[\$I_1/I_1\texttt{'},\ldots,\$I_n/I_n\texttt{'}]\ \texttt{AND}\ F_2[\$I_1/I_1,\ldots,\$I_n/I_n]]_{I_1,\ldots,I_n} \end{array}}$$

This rule shows how the specification of a sequence of commands C_1 and C_2 can be constructed from the specifications of the individual commands (provided that these specifications have the same frame $I_1,\ldots,I_n$) by introducing mathematical variables $\$I_1,\ldots,\$I_n$ that describe the values of these variables in the intermediate state (the one after the execution of C_1 and before the execution of C_2). For example, from the judgements

```
x=x+1 : [x'=x+1 AND y'=y]x,y
y=x+y : [y'=x+y AND x'=x]x,y
```

the following judgement can be derived

```
x=x+1; y=x+y :
  [EXISTS $x, $y:
     $x=x+1 AND $y=y AND
     y'=$x+$y AND x'=$x]x,y
```

which can be further simplified by the semantics of `EXISTS` to

```
x=x+1; y=x+y : [x'=x+1 AND y'=(x+1)+y]x,y
```

Soundness Proof We have to show

$$\begin{array}{ll} & [\![C_1\,\texttt{;}\,C_2]\!](s,s') \Rightarrow \\ \text{(a)} & \quad [\![\,[\texttt{EXISTS } \$I_1,\ldots,\$I_n : \\ & \quad\quad F_1[\$I_1/I_1\texttt{'},\ldots,\$I_n/I_n\texttt{'}]\ \texttt{AND} \\ & \quad\quad F_2[\$I_1/I_1,\ldots,\$I_n/I_n]]_{I_1,\ldots,I_n}\,]\!](e)(s,s') \end{array}$$

Assume

(2) $[\![C_1\,\texttt{;}\,C_2]\!](s,s')$

By the definitions of $[\,\_\,]_{\_}$ and $[\![\,\_\,]\!]$, we have to show

(a.1) $[\![\texttt{EXISTS } \$I_1, \ldots, \$I_n : F_1[\$I_1/I_1', \ldots, \$I_n/I_n'] \texttt{ AND } F_2[\$I_1/I_1, \ldots, \$I_n/I_n]]\!](e)(s,s')$

(a.2) $s = s' \text{ EXCEPT } I_1, \ldots, I_n$

From (2) and the definition of $[\![\_]\!]$, we know

(3) $\exists s_0 \in \mathit{State} : [\![C_1]\!](s,s_0) \wedge [\![C_2]\!](s_0,s')$

Let $s_0 \in \mathit{State}$ such that

(4) $[\![C_1]\!](s,s_0)$

(5) $[\![C_2]\!](s_0,s')$

From the premises, we know

(6) $[\![C_1]\!](s,s_0) \Rightarrow [\![[F_1]_{I_1,\ldots,I_n}]\!](e)(s,s_0)$

(7) $[\![C_2]\!](s_0,s') \Rightarrow [\![[F_2]_{I_1,\ldots,I_n}]\!](e)(s_0,s')$

(7a) $\$I_1, \ldots, \$I_n$ do not occur in F_1 and F_2

From (4), (5), (6), (7), and the definitions of $[\![\_]\!]$ and $[\_]_{\_}$, we know

(8) $[\![F_1]\!](e)(s,s_0)$

(9) $[\![F_2]\!](e)(s_0,s')$

(10) $s = s_0 \text{ EXCEPT } I_1, \ldots, I_n$

(11) $s_0 = s' \text{ EXCEPT } I_1, \ldots, I_n$

From (10) and (11), and (TRE), we know (a.2). To show (a.1), we have by the definition of $[\![\_]\!]$ to show

(a.1.a) $\exists v_1, \ldots, v_n \in \mathit{Value} :$
$[\![F_1[\$I_1/I_1', \ldots, \$I_n/I_n']]\!](e[I_1 \mapsto v_1, \ldots, I_n \mapsto v_n])(s,s') \wedge$
$[\![F_2[\$I_1/I_1, \ldots, \$I_n/I_n]]\!](e[I_1 \mapsto v_1, \ldots, I_n \mapsto v_n])(s,s')$

It thus suffices to show

(a.1.a.1) $[\![F_1[\$I_1/I_1', \ldots, \$I_n/I_n']]\!](e[I_1 \mapsto \mathit{read}(s_0,I_1), \ldots, I_n \mapsto \mathit{read}(s_0,I_n)])(s,s')$

(a.1.a.2) $[\![F_2[\$I_1/I_1, \ldots, \$I_n/I_n]]\!](e[I_1 \mapsto \mathit{read}(s_0,I_1), \ldots, I_n \mapsto \mathit{read}(s_0,I_n)])(s,s')$

From (7a), (8), (11), and (PMVF2), we know (a.1.a.1). From (7a), (9), (10), and (PMVF1), we know (a.1.a.2). □

2.6.7 One-Sided Conditional

$$\frac{C : [F]_{I_1,\ldots,I_n} \qquad E \simeq F_0}{\texttt{if (}E\texttt{) } C : [\texttt{IF } F_0 \texttt{ THEN } F \texttt{ ELSE readsonly}]_{I_1,\ldots,I_n}}$$

This rule shows how the specification of a one-sided conditional can be constructed from the specification of its branch. For instance from the judgement

```
y = y+x:[y'=y+x AND x'=x]x,y
```

the following judgement may be derived:

```
if (x > 0) y = y+x:
  [IF x > 0
     THEN y' = y+x AND x'=x
     ELSE readsonly]x,y
```

Soundness Proof We have to show

(a) $[\![\texttt{if (}E\texttt{) } C]\!](s,s') \Rightarrow [\![[\texttt{IF } F_0 \texttt{ THEN } F \texttt{ ELSE readsonly}]_{I_1,\ldots,I_n}]\!](e)(s,s')$

Assume

(2) $[\![\texttt{if (}E\texttt{) } C]\!](s,s')$

By the definitions of $[\_]_\_$ and $[\![\_]\!]$, we have to show

(a.1) IF $[\![F_0]\!](e)(s,s')$ THEN $[\![F]\!](e)(s,s')$ ELSE $s = s'$

(a.2) $s = s'$ EXCEPT $I_1,\ldots,I_n$

From (2) and the definition of $[\![\_]\!]$, we know

(3) IF $[\![E]\!](s) =$ TRUE THEN $[\![C]\!](s,s')$ ELSE $s' = s$

From the premises and the definition of $\simeq$, we know

(4) $[\![C]\!](s,s') \Rightarrow [\![[F]_{I_1,\ldots,I_n}]\!](e)(s,s')$

(5) $[\![E]\!](s) =$ TRUE $\Leftrightarrow$ $[\![F_0]\!](e)(s,s')$

From (4) and the definitions of $[\_]_\_$ and $[\![\_]\!]$, we know

(6) $[\![C]\!](s,s') \Rightarrow [\![F]\!](e)(s,s')$

(7) $[\![C]\!](s,s') \Rightarrow s = s'$ EXCEPT $I_1,\ldots,I_n$

From (3), (5), and (6), we know (a.1). From (3), (7), (AVE), and (REE), we know (a.2). □

2.6.8 Two-Sided Conditional

$$\frac{C_1 : [F_1]_{I_1,\ldots,I_n} \quad C_2 : [F_2]_{I_1,\ldots,I_n} \quad E \simeq F_0}{\texttt{if (}E\texttt{)}\ C_1\ \texttt{else}\ C_2 : [\texttt{IF}\ F_0\ \texttt{THEN}\ F_1\ \texttt{ELSE}\ F_2]_{I_1,\ldots,I_n}}$$

This rule shows how the specification of a two-sided conditional can be constructed from the specifications of its branches. For instance from the judgements

```
y = y+x:[y'=y+x AND x'=x]x,y
x = y-x:[x'=y-x AND y'=y]x,y
```

the following judgement may be derived:

```
if (x > 0) y = y+x else y = y-x:
  [IF x > 0
     THEN y' = y+x AND x'=x
     ELSE x' = y-x AND y'=y]x,y
```

Soundness Proof We have to show

(a) $[\![\texttt{if (}E\texttt{)}\ C_1\ \texttt{else}\ C_2]\!](s,s') \Rightarrow$
$[\![[\texttt{IF}\ F_0\ \texttt{THEN}\ F_1\ \texttt{ELSE}\ F_2]_{I_1,\ldots,I_n}]\!](e)(s,s')$

Assume

(2) $[\![\texttt{if (}E\texttt{)}\ C_1\ \texttt{else}\ C_2]\!](s,s')$

By the definitions of $[\_]_\_$ and $[\![\_]\!]$, we have to show

(a.1) IF $[\![F_0]\!](e)(s,s')$THEN $[\![F_1]\!](e)(s,s')$ ELSE $[\![F_2]\!](e)(s,s')$

(a.2) $s = s'$ EXCEPT $I_1,\ldots,I_n$

From (2) and the definition of $[\![\_]\!]$, we know

(3) IF $[\![E]\!](s) =$ TRUE THEN $[\![C_1]\!](s,s')$ ELSE $[\![C_2]\!](s,s')$

From the premises and the definition of $\simeq$, we know

(4) $[\![C_1]\!](s,s') \Rightarrow [\![[F_1]_{I_1,\ldots,I_n}]\!](e)(s,s')$

(5) $[\![C_2]\!](s,s') \Rightarrow [\![[F_2]_{I_1,\ldots,I_n}]\!](e)(s,s')$

(6) $[\![E]\!](s) = \text{TRUE} \Leftrightarrow [\![F_0]\!](e)(s, s')$

From (4) and the definitions of $[\,\_\,]_{\_}$ and $[\![\,\_\,]\!]$, we know

(7) $[\![C_1]\!](s, s') \Rightarrow [\![F_1]\!](e)(s, s')$

(8) $[\![C_1]\!](s, s') \Rightarrow\ s = s' \text{ EXCEPT } I_1, \ldots, I_n$

Likewise, from (5) and the definitions of $[\,\_\,]_{\_}$ and $[\![\,\_\,]\!]$, we know

(9) $[\![C_2]\!](s, s') \Rightarrow [\![F_2]\!](e)(s, s')$

(10) $[\![C_2]\!](s, s') \Rightarrow\ s = s' \text{ EXCEPT } I_1, \ldots, I_n$

From (3), (6), (7), and (9), we know (a.1). From (3), (8), and (10), we know (a.2). □

Chapter 3

Loops and Non-Termination

In this chapter, we extend the command language of Chapter 2 by a loop command. We give the syntax and semantics of this command and corresponding verification rules whose soundness we prove. In contrast to the previously introduced commands, a loop does (due to possible non-termination) not necessarily yield a poststate for every prestate. We therefore extend our notion of program specifications by the description of the termination behavior of a program and introduce a calculus for verifying the termination of programs.

3.1 Loops and their Semantics

We extend the command language of Figure 2.1 by the following command:

`while (E) C` A command may be a *(while) loop* composed of an expression E and a command C.

The command shall have the classical interpretation: if the evaluation of E yields true, the command C is executed and we execute the while loop again; otherwise, the command has no effect i.e. it leaves the state unchanged.

Above definition implies that the relationship between a prestate s and a poststate s' of this command is provided by a *sequence* of states such that

- the first state in the sequence is s,
- for every state in the sequence the evaluation of E yields true except for the last one,

Command Language with Loops
An extension of the language of Figure 2.1.

Abstract Syntax

$C ::= \ldots \mid \texttt{while (}E\texttt{) }C.$

Semantic Operations

finiteExecution :
$\quad \mathbb{P}(\mathbb{N} \times State^{\infty} \times State \times StateFunction \times StateRelation)$
$finiteExecution(k,t,s,E,C) \Leftrightarrow$
$\quad t(0) = s \wedge \forall i \in \mathbb{N}_k : E(t(i)) = \text{TRUE} \wedge C(t(i),t(i+1))$

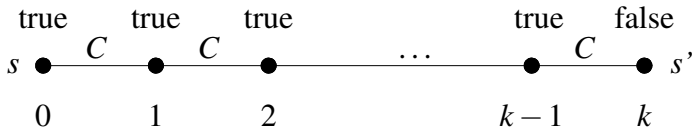
Valuation Function

$[\![\texttt{while (}E\texttt{) }C]\!](s,s') \Leftrightarrow$
$\quad \exists k \in \mathbb{N}, t \in State^{\infty} :$
$\quad\quad finiteExecution(k,t,s,[\![E]\!],[\![C]\!]) \wedge$
$\quad\quad [\![E]\!](t(k)) \neq \text{TRUE} \wedge t(k) = s'$

Figure 3.1: The Command Language with Loops

- the relationship between every successive pair of states is established by the execution of C, and
- the last state in the sequence is s'.

This relationship is illustrated by the following picture



and formalized by the following definition of the state relation which demands the existence of a finite state sequence t with the properties described above:

$[\![\texttt{while (}E\texttt{) }C]\!](s,s') \Leftrightarrow$
$\quad \exists k \in \mathbb{N}, t \in State^{\infty} :$
$\quad\quad t(0) = s \wedge \forall i \in \mathbb{N}_k : E(t(i)) = \text{TRUE} \wedge C(t(i),t(i+1)) \wedge$
$\quad\quad [\![E]\!](t(k)) \neq \text{TRUE} \wedge t(k) = s'$

Figure 3.1 summarizes the syntax and semantics of loops (with the help of a predicate *finiteExecution* that will be reused later).

Verification Calculus: Definitions

$$
\begin{array}{l}
\mathit{Invariant}(G,H,F)_{I_1,\ldots,I_n} \equiv \\
\quad G \text{ has no free variables} \wedge \\
\quad \$I_1,\ldots,\$I_n \text{ do not occur in } G,H, \text{ and } F \wedge \\
\quad \forall e \in \mathit{Environment}, s, s' \in \mathit{Store} : \\
\qquad [\![\texttt{FORALL } \$I_1,\ldots,\$I_n : \\
\qquad\quad (G[\$I_1/I_1\texttt{'},\ldots,\$I_n/I_n\texttt{'}] \texttt{ AND } H[\$I_1/I_1,\ldots,\$I_n/I_n] \\
\qquad\qquad \texttt{AND } F[\$I_1/I_1,\ldots,\$I_n/I_n]) \texttt{ => } G)]\!](e)(s,s')
\end{array}
$$

Verification Calculus: Rules

$$
\frac{\begin{array}{l} C : [F]_{I_1,\ldots,I_n} \\ E \simeq H \end{array}}{\texttt{while (}E\texttt{) } C : [\,\texttt{!}H[I_1\texttt{'}/I_1,\ldots,I_n\texttt{'}/I_n]\,]_{I_1,\ldots,I_n}}
$$

$$
\frac{\begin{array}{l} C : [F]_{I_1,\ldots,I_n} \\ E \simeq H \\ \mathit{Invariant}(G,H,F)_{I_1,\ldots,I_n} \end{array}}{\begin{array}{l} \texttt{while (}E\texttt{) } C : [\,\texttt{!}H[I_1\texttt{'}/I_1,\ldots,I_n\texttt{'}/I_n] \texttt{ AND} \\ \quad (G[I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}] \texttt{ => } G)\,]_{I_1,\ldots,I_n} \end{array}}
$$

Figure 3.2: The Verification Calculus for Loops

3.2 The Verification of Loops

Figure 3.2 gives two rules for the verification of the loop command that extend the verification calculus presented in the previous chapter. Also by these rules (which will be explained in the following subsections), only closed specification formulas can be derived.

Lemma (Closed Specifications) If $C : F$ can be derived by the rules of the verification calculus for loops, then F is closed.

Proof By induction on the structure of the derivation of $C : F$. For the last step of the derivation, we have two additional cases:

Basic Rule The formula $\texttt{!}H[I_1\texttt{'}/I_1,\ldots,I_n\texttt{'}/I_n]$ can only have free variables that are also free in H. Because of the premise $E \simeq H$ and the definition of $\simeq$, H does not have free variables.

Invariant Rule The formula

$$!H[I_1\texttt{'}/I_1,\ldots,I_n\texttt{'}/I_n]\ \texttt{AND}\ (G[I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]\ \texttt{=>}\ G)$$

can only have free variables that are also free in H or in G. Because of the premise $E \simeq H$ and the definition of $\simeq$, H does not have such variables. Because of the premise $Invariant(G,H,F)_{I_1,\ldots,I_n}$ and the definition of *Invariant*, G does also not have such variables. □

3.2.1 Basic Rule

As for a calculus for verifying respectively/deriving the specification of a loop, the only property which is immediately clear from the information explicit in the command is that in a terminal state the loop condition does not hold (any more). We thus can give the following rule:

$$\frac{\begin{array}{l}C:[F]_{I_1,\ldots,I_n}\\ E \simeq H\end{array}}{\texttt{while (}E\texttt{) }C:[\,!H[I_1\texttt{'}/I_1,\ldots,I_n\texttt{'}/I_n]]_{I_1,\ldots,I_n}}$$

This rule derives as a specification the negation of the termination condition (i.e. the negation of a mathematical formula H that corresponds to the program expression E, see the definition of $\simeq$ in Figure 3.1) where all occurrences of potentially changed program variables are replaced by their primed counterparts. For instance, if we can derive the judgements

$$\texttt{s=s+i; i=i+1}:[\texttt{s'=s+i AND i'=i+1}]_{\texttt{S,i}}$$
$$\texttt{i<n} \simeq \texttt{LESS(i,n)}$$

then we can also derive

$$\texttt{while (i<n) \{s=s+i; i=i+1\}}:[\,\texttt{!(LESS(i,n)}\,]_{\texttt{S,i}}$$

This judgement is not really very impressive since the specification does not say anything about the variable `s` changed in the body of the loop. The rule therefore only allows to derive very weak specifications but, on the other hand, its soundness (in the sense of Section 2.6) can be easily proved.

Soundness Proof We have to show

(a) $[\![\texttt{while (}E\texttt{) }C]\!](s,s') \Rightarrow [\![[\,!H[I_1\texttt{'}/I_1,\ldots,I_n\texttt{'}/I_n]]_{I_1,\ldots,I_n}]\!](e)(s,s')$

Assume

(2) $[\![\texttt{while (}E\texttt{) }C]\!](s,s')$

By the definitions of $[\![\_]\!]_\_$ and $[\![\_]\!]$, we have to show

(a.1) $\neg[\![H[I_1\texttt{'}/I_1,\ldots,I_n\texttt{'}/I_n]]\!](e)(s,s')$

(a.2) $s = s'$ EXCEPT $I_1,\ldots,I_n$

From (2) and the definition of $[\![\_]\!]$, we know

$$\begin{array}{ll} & \exists k \in \mathbb{N}, t \in \mathit{State}^\infty : \\ (3) & \quad \mathit{finiteExecution}(k,t,s,[\![E]\!],[\![C]\!]) \wedge \\ & \quad [\![E]\!](t(k)) \neq \text{TRUE} \wedge t(k) = s' \end{array}$$

Let $k \in \mathbb{N}, t \in \mathit{State}^\infty$ such that

(4) $\mathit{finiteExecution}(k,t,s,[\![E]\!],[\![C]\!])$

(5) $[\![E]\!](t(k)) \neq \text{TRUE}$

(6) $t(k) = s'$

From (4) and the definition of *finiteExecution*, we know

(7) $t(0) = s$

(8) $\forall i \in \mathbb{N}_k : [\![E]\!](t(i)) = \text{TRUE} \wedge [\![C]\!](t(i),t(i+1))$

From the premises, we know

(9) $\forall e \in \mathit{Environment}, s, s' \in \mathit{State} : [\![C]\!](s,s') \Rightarrow [\![[F]_{I_1,\ldots,I_n}]\!](e)(s,s')$

(10) $E \simeq H$

From (8) and (9), we know

(11) $\forall i \in \mathbb{N}_k : [\![[F]_{I_1,\ldots,I_n}]\!](e)(t(i),t(i+1))$

From (11), we know by the definitions of $[\![\_]\!]$ and $[\_]_\_$

(12) $\forall i \in \mathbb{N}_k : [\![F]\!](e)(t(i),t(i+1))$

(13) $\forall i \in \mathbb{N}_k : t(i) = t(i+1)$ EXCEPT $I_1,\ldots,I_n$

From (6), (7), (13) and (TRE), we know (a.2).

From (5) and (6), we know

(14) $[\![E]\!](s') \neq \text{TRUE}$

From (10) and the definition of $\simeq$, we know

(15) $[\![E]\!](s') = \text{TRUE} \Leftrightarrow [\![H]\!](e)(s',s')$

From (14), and (15), we know

(16) $\neg[\![H]\!](e)(s',s')$

We define

(17) $s'' := writes(s, I_1, read(s', I_1), \ldots, I_n, read(s', I_n))$

From (17) and (WSE), we know

(18) $s'' = s \text{ EXCEPT } I_1, \ldots, I_n$

From (18), (a.2), and (TRE), we know

(19) $s'' = s' \text{ EXCEPT } I_1, \ldots, I_n$

From *DifferentVariables*, (17), and (RWE), we know

(20) $read(s'', I_1) = read(s', I_1) \wedge \ldots \wedge read(s'', I_n) = read(s', I_n)$

From (19), (20), (RVE), and (NEQ), we know

(21) $s'' \text{ EQUALS } s'$

From (REE) and (NEQ), we know

(22) $s' \text{ EQUALS } s'$

From (16), (21), (22), and (ESF), we know

(24) $\neg[\![H]\!](e)(s'',s')$

From (17), (24), and (PPVF1), we know (a.1). □

3.2.2 Invariant Rule

The basic loop verification rule is unsatisfactory because it does not relate the poststate of the loop to its prestate. Our goal is thus to derive from the loop expression E (respectively its formula counterpart H) and from the loop body C (respectively its specification formula $[\,F\,]_{I_1,\ldots,I_n}$) a formula G that provides such a relationship. If we comprise this construction under the predicate name $\mathit{Invariant}(G,T,F)_{I_1,\ldots,I_n}$ (which will be explained later), we get the following rule which results in a judgement that is apparently stronger than the one obtained by the basic rule:

$$\frac{\begin{array}{l} C : [\,F\,]_{I_1,\ldots,I_n} \\ E \simeq H \\ \mathit{Invariant}(G,H,F)_{I_1,\ldots,I_n} \end{array}}{\begin{array}{l} \texttt{while (}E\texttt{) } C : [\,!H[I_1\texttt{'}/I_1,\ldots,I_n\texttt{'}/I_n]\ \texttt{AND} \\ \quad (G[I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]\ \texttt{=>}\ G)\,]_{I_1,\ldots,I_n} \end{array}}$$

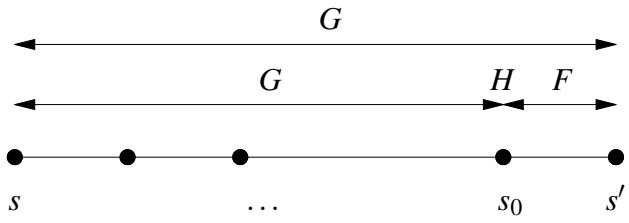
Rather than giving an explicit construction of G (which is hardly possible), the predicate *Invariant* describes the properties from which we can conclude that G is indeed a relation between the prestate of the loop and its poststate (provided that it is also relation between the prestate and itself). The core idea is that G must describe the relationship between the prestate of the loop and the poststate after *an arbitrary number of iterations* i.e. that G remains *invariant* through the execution of the loop. This can be shown by some sort of induction:

Induction Base We show that G describes the relationship between the prestate s and the poststate $s' = s$ (i.e. the poststate after 0 iterations). This induction base is captured by the premise

$$G[I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]$$

in the derived specification where all references $I_1\texttt{'},\ldots,I_n\texttt{'}$ to the poststate values of potentially modified variables are replaced by references $I_1,\ldots,I_n$ to their prestate values.

Induction Step Assuming that G describes the relationship between the prestate s and the state s_0 after k iterations, we show that G also describes the relationship between s and the state s' after $k+1$ iterations; since one additional iteration is performed, we may assume that the loop condition H holds in s_0 and that the specification F of the loop body relates s_0 to s'. This obligation is illustrated by the following figure:



This induction step is captured by the predicate *Invariant* which introduces mathematical variables $\$I_1, \ldots, \$I_n$ to describe the values of the potentially modified program variables in the intermediate state s_0:

$$
\begin{array}{l}
\mathit{Invariant}(G,H,F)_{I_1,\ldots,I_n} \equiv \\
\quad G \text{ has no free variables} \wedge \\
\quad \forall e \in \mathit{Environment}, s, s' \in \mathit{Store} : \\
\quad\quad [\![\texttt{FORALL } \$I_1, \ldots, \$I_n : \\
\quad\quad\quad (G[\$I_1/I_1\texttt{'}, \ldots, \$I_n/I_n\texttt{'}] \texttt{ AND } H[\$I_1/I_1, \ldots, \$I_n/I_n] \\
\quad\quad\quad\quad \texttt{AND } F[\$I_1/I_1, \ldots, \$I_n/I_n] \texttt{ => } G)]\!](e)(s,s')
\end{array}
$$

For example, for the loop

```
while (i<n) {s=s+i; i=i+1}
```

with judgements

```
s=s+i; i=i+1 :[s'=s+i AND i'=i+1]_{s,i}
i<n ≃ i<n
```

a suitable invariant might be

```
nat(i) AND i<=n AND nat(s) =>
  nat(i') AND i'<=n AND s'=s+sum(i,i')
```

where `nat(`i`)` shall express that i is a natural number and `sum(`i`,` j`)` shall denote the sum of all natural numbers greater than or equal to i and less than j.

It is then necessary to show the validity of the induction step

```
FORALL $i, $s:
  ((nat(i) AND i<=n AND nat(s) =>
     nat($i) AND $i<=n AND $s=s+sum(i,$i)) AND
  $i < n AND
  (s'=$s+$i AND i'=$i+1 AND writesonly s,i)) =>
    (nat(i) AND i<=n AND nat(s) =>
      nat(i') AND i'<=n AND s'=s+sum(i,i'))
```

Since the induction base

```
nat(i) AND i<=n AND nat(s) =>
  nat(i) AND i<=n AND s=s+sum(i,i)
```

is valid in every state, the loop specification

```
(nat(i) AND i<=n AND nat(s) =>
  nat(i') AND i'<=n AND s'=s+sum(i,i'))
AND !(i<n)
```

can be derived.

Soundness Proof We have to show

$$\text{(a)} \quad \begin{array}{l} [\![\texttt{while (}E\texttt{) }C]\!](s,s') \Rightarrow \\ \quad [\![[\,\texttt{!}H[I_1\texttt{'}/I_1,\ldots,I_n\texttt{'}/I_n]\ \texttt{AND} \\ \qquad (G[I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]\ \texttt{=>}\ G)\,]_{I_1,\ldots,I_n}]\!](e)(s,s') \end{array}$$

Assume

(2) $[\![\texttt{while (}E\texttt{) }C]\!](s,s')$

By the definitions of $[\,\_\,]_{\_}$ and $[\![\,\_\,]\!]$, we have to show

(a.1) $\neg[\![H[I_1\texttt{'}/I_1,\ldots,I_n\texttt{'}/I_n]]\!](e)(s,s')$

(a.2) $[\![G[I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]]\!](e)(s,s') \Rightarrow [\![G]\!](e)(s,s')$

(a.3) $s = s'$ EXCEPT $I_1,\ldots,I_n$

The proofs of (a.1) and (a.3) proceed as shown in the soundness proof of the basic version of the rule. We are now going to show (a.2).

We assume

(3) $[\![G[I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]]\!](e)(s,s')$

and show

(a.2.a) $[\![G]\!](e)(s,s')$

From (2) and the definitions of $[\![\,\_\,]\!]$ and *finiteExecution*, we know that there exist some $k \in \mathbb{N}, t \in State^{\infty}$ such that

(4) $t(0) = s$

(5) $\forall i \in \mathbb{N}_k : [\![E]\!](t(i)) = \text{TRUE} \wedge [\![C]\!](t(i), t(i+1))$

(6) $[\![E]\!](t(k)) \neq \text{TRUE}$

(7) $t(k) = s'$

From the premises, we know

(8) $\forall e \in \mathit{Environment}, s, s' \in \mathit{State} : [\![C]\!](s, s') \Rightarrow [\![[F]_{I_1,\ldots,I_n}]\!](e)(s, s')$

(9) $E \simeq H$

(10) $\mathit{Invariant}(G, T, F)_{I_1,\ldots,I_n}$

From (5) and (8), we know by the definitions of $[\![\_]\!]$ and $[\_]_\_$

(11) $\forall i \in \mathbb{N}_k : [\![F]\!](e)(t(i), t(i+1))$

(12) $\forall i \in \mathbb{N}_k : t(i) = t(i+1)$ EXCEPT $I_1, \ldots, I_n$

From (10) and the definition of *Invariant*, we know

(13) G has no free variables

(13a) $\$I_1, \ldots, \$I_n$ do not occur in $G, H,$ and F

(14)
$$\begin{array}{l} \forall e \in \mathit{Environment}, s, s' \in \mathit{Store} : \\ \quad [\![\texttt{FORALL } \$I_1, \ldots, \$I_n : \\ \qquad (G[\$I_1/I_1\,', \ldots, \$I_n/I_n\,'] \texttt{ AND } H[\$I_1/I_1, \ldots, \$I_n/I_n] \\ \qquad\quad \texttt{AND } F[\$I_1/I_1, \ldots, \$I_n/I_n]) \texttt{ => } G)]\!](e)(s, s') \end{array}$$

We are now going to show

(a.2.b) $\forall i \in \mathbb{N}_{k+1} : [\![G]\!](e)(s, t(i))$

from which with (7) we know (a.2.a). The proof proceeds by induction on i with bound k.

Induction Base We show

(a.2.b.1) $[\![G]\!](e)(s, t(0))$

By (4), it suffices to show

(a.2.b.1.a) $[\![G]\!](e)(s, s)$

We define

(15) $s'' := writes(s', I_1, read(s, I_1), \ldots, I_n, read(s, I_n))$

From (3), (15), and (PPVF1), we know

(16) $[\![G]\!](e)(s, s'')$

From (15) and (WSE), we know

(17) $s'' = s'$ EXCEPT $I_1, \ldots, I_n$

From (a.3), (17), and (TRE), we know

(18) $s'' = s$ EXCEPT $I_1, \ldots, I_n$

From (15) and (RWE), we know

(19) $read(s'', I_1) = read(s, I_1) \wedge \ldots \wedge read(s'', I_n) = read(s, I_n)$

From (18), (19), (RVE), and (NEQ), we know

(20) s'' EQUALS s

From (REE) and (NEQ), we know

(21) s EQUALS s

From (16), (20), (21), and (PPVF2), we know (a.2.b.1.a).

Induction Step Take arbitrary $i \in \mathbb{N}_k$. We assume

(15) $[\![G]\!](e)(s, t(i))$

and show

(a.2.b.2) $[\![G]\!](e)(s, t(i+1))$

From (14) and the definition of $[\![\_]\!]$, we know

$$
\begin{array}{ll}
 & \forall v_1, \ldots, v_n \in Value : \\
 & \quad [\![G[\$I_1/I_1{}', \ldots, \$I_n/I_n{}']]\!](e[I_1 \mapsto v_1, \ldots, I_n \mapsto v_n])(s, t(i+1)) \wedge \\
(16) & \quad [\![H[\$I_1/I_1, \ldots, \$I_n/I_n]]\!](e[I_1 \mapsto v_1, \ldots, I_n \mapsto v_n])(s, t(i+1)) \wedge \\
 & \quad [\![F[\$I_1/I_1, \ldots, \$I_n/I_n]]\!](e[I_1 \mapsto v_1, \ldots, I_n \mapsto v_n])(s, t(i+1)) \Rightarrow \\
 & \qquad [\![G]\!](e[I_1 \mapsto v_1, \ldots, I_n \mapsto v_n])(s, t(i+1))
\end{array}
$$

To prove (a.2.b.2), by (13) and (MVF), it suffices to prove

(a.2.b.2.a) $[\![G]\!](e[I_1 \mapsto read(t(i),I_1),\ldots,I_n \mapsto read(t(i),I_n)])(s,t(i+1))$

To prove this, by (16), it suffices to prove

(a.2.b.2.a.1) $[\![G[\$I_1/I_1{}',\ldots,\$I_n/I_n{}']]\!]$
$(e[I_1 \mapsto read(t(i),I_1),\ldots,I_n \mapsto read(t(i),I_n)])(s,t(i+1))$

(a.2.b.2.a.2) $[\![H[\$I_1/I_1,\ldots,\$I_n/I_n]]\!]$
$(e[I_1 \mapsto read(t(i),I_1),\ldots,I_n \mapsto read(t(i),I_n)])(s,t(i+1))$

(a.2.b.2.a.3) $[\![F[\$I_1/I_1,\ldots,\$I_n/I_n]]\!]$
$(e[I_1 \mapsto read(t(i),I_1),\ldots,I_n \mapsto read(t(i),I_n)])(s,t(i+1))$

From (12), we know

(17) $t(i) = t(i+1)$ EXCEPT $I_1,\ldots,I_n$

From (13a), (15), (17), and (PMVF2), we know (a.2.b.2.a.1).

From (5), (9), and the definition of $\simeq$, we know

(18) $[\![H]\!](e)(t(i),t(i+1))$

From (4), (12), and (TRE), we know

(19) $s = t(i)$ EXCEPT $I_1,\ldots,I_n$

From (13a), (18), (19), and (PMVF1), we know (a.2.b.2.a.2).

From (11), we know

(20) $[\![F]\!](e)(t(i),t(i+1))$

From (13a), (19), (20), and (PMVF1), we know (a.2.b.2.a.3). □

3.3 The Problem of Non-Termination

The loop command introduces a problem that we have not encountered so far: a command does not necessarily relate every prestate to some poststate i.e. the command may not terminate for some prestates. For example, the program

```
while (x>0) x=x-i;
```

Command Language: Termination Conditions

Definitions

$StateCondition := \mathbb{P}(State)$

$infiniteExecution :$
$\quad \mathbb{P}(State^{\infty} \times State \times StateFunction \times StateRelation)$
$infiniteExecution(t,s,E,C) \Leftrightarrow$
$\quad t(0) = s \wedge \forall i \in \mathbb{N} : E(t(i)) = \text{TRUE} \wedge C(t(i),t(i+1))$

Valuation Functions

$[\![\_]\!]_{\mathrm{T}} : \textbf{Program} \rightarrow \textbf{\textit{StateCondition}}$
$[\![C]\!]_{\mathrm{T}} = [\![C]\!]_{\mathrm{T}}$

$[\![\_]\!]_{\mathrm{T}} : \textbf{Command} \rightarrow \textbf{\textit{StateCondition}}$
$[\![I = E]\!]_{\mathrm{T}}(s) \Leftrightarrow \text{TRUE}$
$[\![\texttt{var } I\texttt{; } C]\!]_{\mathrm{T}}(s) \Leftrightarrow \forall v \in Value : [\![C]\!]_{\mathrm{T}}(write(s,I,v))$
$[\![\texttt{var } I\texttt{=}E\texttt{; } C]\!]_{\mathrm{T}}(s) \Leftrightarrow [\![C]\!]_{\mathrm{T}}(write(s,I,[\![E]\!](s)))$
$[\![C_1\texttt{; } C_2]\!]_{\mathrm{T}}(s) \Leftrightarrow$
$\quad [\![C_1]\!]_{\mathrm{T}}(s) \wedge \forall s' \in State : [\![C_1]\!](s,s') \Rightarrow [\![C_2]\!]_{\mathrm{T}}(s')$
$[\![\texttt{if (}E\texttt{) } C]\!]_{\mathrm{T}}(s) \Leftrightarrow$
$\quad [\![E]\!](s) = \text{TRUE} \Rightarrow [\![C]\!]_{\mathrm{T}}(s)$
$[\![\texttt{if (}E\texttt{) } C_1 \texttt{ else } C_2]\!]_{\mathrm{T}}(s) \Leftrightarrow$
$\quad \text{IF } [\![E]\!](s) = \text{TRUE THEN } [\![C_1]\!]_{\mathrm{T}}(s) \text{ ELSE } [\![C_2]\!]_{\mathrm{T}}(s)$
$[\![\texttt{while (}E\texttt{) } C]\!]_{\mathrm{T}}(s) \Leftrightarrow$
$\quad \forall t \in State^{\infty}, k \in \mathbb{N} :$
$\qquad \neg infiniteExecution(t,s,[\![E]\!],[\![C]\!]) \wedge$
$\qquad (finiteExecution(k,t,s,[\![E]\!],[\![C]\!]) \Rightarrow$
$\qquad\quad [\![E]\!](t(k)) = \text{TRUE} \Rightarrow [\![C]\!]_{\mathrm{T}}(t(k)))$

Figure 3.3: The Termination Conditions of the Command Language

only terminates for a prestate s with $read(s, \mathtt{i}) > 0$. To make explicit for which prestates a program/command yields a poststate, we introduce in Figure 3.3 valuation functions

$$[\![\_]\!]_{\mathrm{T}} : \text{Program} \to \mathit{StateCondition}$$
$$[\![\_]\!]_{\mathrm{T}} : \text{Command} \to \mathit{StateCondition}$$

with the properties stated below.

Theorem (Termination Condition) If the termination condition of a program respectively command is true on a prestate s, then the program respectively command relates s to some poststate s':

$$\forall P \in \text{Program}, s \in \mathit{State} : [\![P]\!]_{\mathrm{T}}(s) \Rightarrow \exists s' \in \mathit{State} : [\![P]\!](s, s')$$
$$\forall C \in \text{Command}, s \in \mathit{State} : [\![C]\!]_{\mathrm{T}}(s) \Rightarrow \exists s' \in \mathit{State} : [\![C]\!](s, s')$$

Proof The first part of the theorem is immediately clear by the definition of $[\![\_]\!]_{\mathrm{T}} : \text{Program} \to \mathit{StateCondition}$ and by the second part of the theorem. We are now going to show this second part i.e.

(a) $\forall C \in \text{Command}, s \in \mathit{State} : [\![C]\!]_{\mathrm{T}}(s) \Rightarrow \exists s' \in \mathit{State} : [\![C]\!](s, s')$

Take arbitrary $C_0 \in \text{Command}, s \in \mathit{State}$ and assume

(1) $[\![C_0]\!]_{\mathrm{T}}(s)$

We show

(b) $\exists s' \in \mathit{State} : [\![C_0]\!](s, s')$

We proceed by induction on the structure of C_0.

I = E From the definition of $[\![\_]\!]$, we know

(2) $[\![I = E]\!](s, write(s, I, [\![E]\!](s)))$

and thus (b).

`var` I; C From (1) and the definition of $[\![\_]\!]_{\mathrm{T}}$, we know

(2) $\forall v \in \mathit{Value} : [\![C]\!]_{\mathrm{T}}(write(s, I, v))$

From the definition of $[\![\_]\!]$, it suffices to show

(c) $\exists s_0, s_1, s' \in State :$
$s_0 = s \text{ EXCEPT } I \wedge [\![C]\!](s_0, s_1) \wedge s' = write(s_1, I, read(s, I))$

which can be simplified to

(d) $\exists s_0, s_1 \in State : s_0 = s \text{ EXCEPT } I \wedge [\![C]\!](s_0, s_1)$

From (ID), we know

(3) $s = write(s, I, s(I))$

From (2) and (3), we know

(4) $[\![C]\!]_{\mathrm{T}}(s)$

From (4) and the induction hypothesis, we know form some $s_1 \in State$

(5) $[\![C]\!](s, s_1)$

To show (d), it suffices to show

(d.1) $s = s \text{ EXCEPT } I$
(d.2) $[\![C]\!](s, s_1)$

From (RE), we know (c.1). From (5), we know (d.2).

`var` I=E; C From (1) and the definition of $[\![\_]\!]_{\mathrm{T}}$, we know

(2) $[\![C]\!]_{\mathrm{T}}(write(s, I, [\![E]\!](s)))$

From the definition of $[\![\_]\!]$, it suffices to show

(c) $\exists s_0, s_1, s' \in State :$
$s_0 = write(s, I, [\![E]\!](s)) \wedge [\![C]\!](s_0, s_1) \wedge$
$s' = write(s_1, I, read(s, I))$

which can be simplified to

(d) $\exists s_1 \in State : [\![C]\!](write(s, I, [\![E]\!](s)), s_1)$

From (2) and the induction hypothesis, we know (d).

C_1 ; C_2 From (1) and the definition of $[\![\_]\!]_{\mathrm{T}}$, we know

(2) $[\![C_1]\!]_{\mathrm{T}}(s)$
(3) $\forall s' \in State : [\![C_1]\!](s, s') \Rightarrow [\![C_2]\!]_{\mathrm{T}}(s')$

From the definition of $[\![\_]\!]$, it suffices to show

(c) $\exists s_0, s' \in State : [\![C_1]\!](s, s_0) \wedge [\![C_2]\!](s_0, s')$

From (2) and the induction hypothesis, we know for some $s_0 \in State$

(4) $[\![C_1]\!](s, s_0)$

From (3) and (4), we know

(5) $[\![C_2]\!]_{\mathrm{T}}(s_0)$

From (5) and the induction hypothesis, we know for some $s' \in \mathit{State}$

(6) $[\![C_2]\!](s_0, s')$

From (4) and (6), we know (c).

`if (E) C` From (1) and the definition of $[\![\_]\!]_{\mathrm{T}}$, we know

(2) $[\![E]\!](s) = \text{TRUE} \Rightarrow [\![C]\!]_{\mathrm{T}}(s)$

From the definition of $[\![\_]\!]$, it suffices to show

(c) $\exists s' \in \mathit{State} : \text{IF } [\![E]\!](s) = \text{TRUE THEN } [\![C]\!](s, s') \text{ ELSE } s' = s$

which can be rewritten to

(d)
$$\begin{array}{l}\text{IF } [\![E]\!](s) = \text{TRUE} \\ \quad \text{THEN } (\exists s' \in \mathit{State} : [\![C]\!](s, s')) \\ \quad \text{ELSE } (\exists s' \in \mathit{State} : s' = s)\end{array}$$

which can be further simplified to

(e) $[\![E]\!](s) = \text{TRUE} \Rightarrow \exists s' \in \mathit{State} : [\![C]\!](s, s')$

Assume

(3) $[\![E]\!](s) = \text{TRUE}$

We have to show

(f) $\exists s' \in \mathit{State} : [\![C]\!](s, s')$

From (2) and (3), we know

(4) $[\![C]\!]_{\mathrm{T}}(s)$

From (4) and the induction hypothesis, we know (f).

`if (E) C`$_1$ `else` C_2 From (1) and the definition of $[\![\_]\!]_{\mathrm{T}}$, we know

(2) $\text{IF } [\![E]\!](s) = \text{TRUE THEN } [\![C_1]\!]_{\mathrm{T}}(s) \text{ ELSE } [\![C_2]\!]_{\mathrm{T}}(s)$

From the definition of $[\![\_]\!]$, it suffices to show

(c) $\exists s' \in \mathit{State} : \text{IF } [\![E]\!](s) = \text{TRUE THEN } [\![C_1]\!](s, s') \text{ ELSE } [\![C_2]\!](s, s')$

which can be rewritten to

$$\begin{array}{ll} & \text{IF } [\![E]\!](s) = \text{TRUE} \\ \text{(d)} & \quad \text{THEN } (\exists s' \in \mathit{State} : [\![C_1]\!](s,s')) \\ & \quad \text{ELSE } (\exists s' \in \mathit{State} : [\![C_2]\!](s,s')) \end{array}$$

1. Case $[\![E]\!](s) = \text{TRUE}$: We have to show
   (e.1) $\exists s' \in \mathit{State} : [\![C_1]\!](s,s')$
   From (2) and the case condition, we know
   (4) $[\![C_1]\!]_{\mathrm{T}}(s)$
   From (4) and the induction hypothesis, we know (e.1).
2. Case $\mathit{semanticsE}(s) \neq \text{TRUE}$: We have to show
   (e.2) $\exists s' \in \mathit{State} : [\![C_2]\!](s,s')$
   From (2) and the case condition, we know
   (5) $[\![C_2]\!]_{\mathrm{T}}(s)$
   From (5) and the induction hypothesis, we know (e.2).

`while (E) C` From (1) and the definition of $[\![\_]\!]_{\mathrm{T}}$, we know

$$\begin{array}{ll} & \forall t \in \mathit{State}^\infty, k \in \mathbb{N} : \\ & \quad \neg \mathit{infiniteExecution}(t,s,[\![E]\!],[\![C]\!]) \wedge \\ \text{(2)} & \quad (\mathit{finiteExecution}(k,t,s,[\![E]\!],[\![C]\!]) \Rightarrow \\ & \qquad [\![E]\!](t(k)) = \text{TRUE} \Rightarrow [\![C]\!]_{\mathrm{T}}(t(k))) \end{array}$$

From the definition of $[\![\_]\!]$, it suffices to show

$$\begin{array}{ll} & \exists k \in \mathbb{N}, t \in \mathit{State}^\infty, s' \in \mathit{State} : \\ \text{(c)} & \quad \mathit{finiteExecution}(k,t,s,[\![E]\!],[\![C]\!]) \wedge \\ & \quad [\![E]\!](t(k)) \neq \text{TRUE} \wedge t(k) = s' \end{array}$$

which can be simplified to

$$\begin{array}{ll} & \exists k \in \mathbb{N}, t \in \mathit{State}^\infty : \\ \text{(d)} & \quad \mathit{finiteExecution}(k,t,s,[\![E]\!],[\![C]\!]) \wedge [\![E]\!](t(k)) \neq \text{TRUE} \end{array}$$

We define $t \in \mathit{State}^\infty$ inductively as follows:

$$\begin{array}{lll} \text{(3)} & t(0) := s & \\ & & \text{IF } \exists s \in \mathit{State} : [\![C]\!](t(i),s) \\ \text{(4)} & t(i+1) := & \quad \text{THEN SUCH } s \text{ IN } \mathit{State} : [\![C]\!](t(i),s) \\ & & \quad \text{ELSE } t(i) \end{array}$$

From (2), we know

(5) $\neg \mathit{infiniteExecution}(t,s,[\![E]\!],[\![C]\!])$

i.e. by the definition of *infiniteExecution*

(6) $t(0) \neq s \vee \exists i \in \mathbb{N} : [\![E]\!](t(i)) \neq \text{TRUE} \vee \neg[\![C]\!](t(i), t(i+1))$

From (3) and (6), we know

(7) $\exists i \in \mathbb{N} : [\![E]\!](t(i)) \neq \text{TRUE} \vee \neg[\![C]\!](t(i), t(i+1))$

We define

(8) $k := \text{MIN } i \in \mathbb{N} : [\![E]\!](t(i)) \neq \text{TRUE} \vee \neg[\![C]\!](t(i), t(i+1))$

From (7) and (8), we know

(9) $[\![E]\!](t(k)) \neq \text{TRUE} \vee \neg[\![C]\!](t(k), t(k+1))$
(10) $\forall i \in \mathbb{N}_k : [\![E]\!](t(i)) = \text{TRUE} \wedge [\![C]\!](t(i), t(i+1))$

From (3), (10), and the definition of *finiteExecution*, we thus know

(11) $\mathit{finiteExecution}(k, t, s, [\![E]\!], [\![C]\!])$

It now suffices to show

(e) $[\![E]\!](t(k)) \neq \text{TRUE}$

because from this and (11), we know (d).

To show (e), we assume

(12) $[\![E]\!](t(k)) = \text{TRUE}$

and show a contradiction.

From (12) and (9), we know

(13) $\neg[\![C]\!](t(k), t(k+1))$

From (4) and (13), we know

(14) $\neg\exists s \in \mathit{State} : [\![C]\!](t(k), s)$

But from (2), (11), and (12), we know

(15) $[\![C]\!]_{\text{T}}(t(k))$

which contradicts (14) by the induction hypothesis. □

3.4 Verifying the Termination of Programs

Figures 3.4, 3.5, and 3.6 present a calculus for deriving judgements of the form

$C \downarrow F$

Termination Calculus: Definitions

$[\![\_]\!] : \text{Formula} \to \mathit{StateCondition}$
$[\![F]\!](s) = \forall e \in \mathit{Environment} : [\![F]\!](e)(s,s)$

Termination Calculus: Judgements

$C \downarrow F \Leftrightarrow$
$\quad \forall s \in \mathit{State} : [\![F]\!](s) \Rightarrow [\![C]\!]_{\mathrm{T}}(s)$

Figure 3.4: The Termination Calculus of the Command Language (Part 1 of 3)

which can be read as "command C terminates if formula F is true". F does not have any free occurrence of a mathematical variable (i.e. does not depend on a mathematical environment) and does also not depend on the poststate[1]. The semantics of such a formula can thus be defined by a valuation function

$[\![\_]\!] : \text{Formula} \to \mathit{StateCondition}$
$[\![F]\!](s) \Leftrightarrow \forall e \in \mathit{Environment} : [\![F]\!](e)(s,s)$

The rules of the calculus (explained below) are sound with respect to the termination semantics of the previous section in the sense stated by the following theorem.

Theorem (Soundness of the Termination Calculus) Assume the condition denoted by *DifferentVariables*. If a judgement $C \downarrow F$ can be derived from the rules of the termination calculus of the command language, then it is true that

F has no free variables $\wedge$
F does not depend on the poststate $\Rightarrow$
$\quad \forall s \in \mathit{State} : [\![F]\!](s) \Rightarrow [\![C]\!]_{\mathrm{T}}(s)$

Corollary (Existence of Poststate) If $C \downarrow F$ can be derived, then for every prestate s with $[\![F]\!](s)$, there exists some poststate s' with $[\![C]\!](s,s')$.

[1] We deliberately do not require that the formula must not contain primed program variables: the calculus demands in two cases (command sequences and loops) as parts of such conditions formulas derived from specifications which may make arbitrary use of such variables. Rather than a syntactic restriction, we thus demand a semantic restriction as stated above.

Termination Calculus: Rules

$$I = E \downarrow F$$

$$\frac{\$I \text{ does not occur in } F \qquad C \downarrow \texttt{EXISTS } \$I : F[\$I/I]}{\texttt{var } I\texttt{; } C \downarrow F}$$

$$\frac{E \simeq T \qquad \$I \text{ does not occur in } T \text{ and } F \qquad C \downarrow \texttt{EXISTS } \$I : I{=}T[\$I/I] \texttt{ AND } F[\$I/I]}{\texttt{var } I{=}E\texttt{; } C \downarrow F}$$

$$\frac{C_1 \downarrow F \qquad C_2 \downarrow \texttt{TRUE}}{C_1\texttt{;}C_2 \downarrow F}$$

$$\frac{\begin{array}{l} C_1 \downarrow F \qquad C_1 : [S]_{I_1,\ldots,I_n} \qquad \$I_1,\ldots,\$I_n \text{ do not occur in } F \text{ and } S \\ C_2 \downarrow \texttt{EXISTS } \$I_1,\ldots,\$I_n : \\ \qquad F[\$I_1/I_1,\ldots,\$I_n/I_n] \texttt{ AND} \\ \qquad S[\$I_1/I_1,\ldots,\$I_n/I_n, I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}] \end{array}}{C_1\texttt{;}C_2 \downarrow F}$$

$$\frac{E \simeq T \qquad C \downarrow F \texttt{ AND } T}{\texttt{if (}E\texttt{) } C \downarrow F}$$

$$\frac{E \simeq T \qquad C_1 \downarrow F \texttt{ AND } T \qquad C_2 \downarrow F \texttt{ AND !}T}{\texttt{if (}E\texttt{) } C_1 \texttt{ else } C_2 \downarrow F}$$

Figure 3.5: The Termination Calculus of the Command Language (Part 2 of 3)

Termination Calculus: Rules

$$\frac{\forall s \in \mathit{State} : [\![F \mathrel{=>} \texttt{!}\, E]\!](s)}{\texttt{while (}E\texttt{)}\ C \downarrow F}$$

$$\frac{\begin{array}{l} E \simeq H \\ C : [S]_{I_1,\ldots,I_n} \\ \mathit{Invariant}(G,H,S)_{I_1,\ldots,I_n} \\ T \text{ has no free variables and no primed program variables} \\ \forall s \in \mathit{State} : [\![F \mathrel{=>} G[I_1/I_1\texttt{'}, \ldots, I_n/I_n\texttt{'}\,]]\!](s) \\ C \downarrow \texttt{EXISTS } \$I_1, \ldots, \$I_n : F[\$I_1/I_1, \ldots, \$I_n/I_n] \texttt{ AND} \\ \qquad G[\$I_1/I_1, \ldots, \$I_n/I_n, I_1/I_1\texttt{'}, \ldots, I_n/I_n\texttt{'}\,] \texttt{ AND } H \\ \forall e \in \mathit{Environment}, s, s' \in \mathit{Store}, v_1, \ldots, v_n \in \mathit{Value} : \\ \quad \text{LET } e_0 = e[I_1 \mapsto v_1, \ldots, I_n \mapsto v_n] \text{ IN} \\ \qquad [\![F[\$I_1/I_1, \ldots, \$I_n/I_n] \texttt{ AND} \\ \qquad\quad G[\$I_1/I_1, \ldots, \$I_n/I_n, I_1/I_1\texttt{'}, \ldots, I_n/I_n\texttt{'}\,] \texttt{ AND } H \texttt{ AND} \\ \qquad\quad [S]_{I_1,\ldots,I_n}]\!](e_0)(s,s') \Rightarrow \\ \qquad \text{LET} \\ \qquad\quad m = [\![T]\!](e_0)(s,s'), \\ \qquad\quad m' = [\![T[I_1\texttt{'}/I_1, \ldots, I_n\texttt{'}/I_n]]\!](e_0)(s,s') \\ \qquad \text{IN } m \in \mathbb{N} \wedge m > m' \end{array}}{\texttt{while (}E\texttt{)}\ C \downarrow F}$$

Figure 3.6: The Termination Calculus of the Command Language (Part 3 of 3)

Proof (Soundness Theorem and Corollary) Assume

(1a) *DifferentVariables*

Take C and F such that $C \downarrow F$ can be derived and assume

(1b) F has no free variables

(1c) $\forall s, s', s'' \in \mathit{State}, e \in \mathit{Environment}:$
$[\![F]\!](e)(s, s') \Leftrightarrow [\![F]\!](e)(s, s'')$

Take arbitrary $s \in \mathit{State}$. We prove

$$[\![F]\!](s) \Rightarrow [\![C]\!]_{\mathrm{T}}(s)$$

by induction on the derivation of $C \downarrow F$. The following subsections cover all cases for the last step of such a derivation.

In these proofs, we will assume that the induction hypothesis immediately implies

$$\forall s \in \mathit{State} : [\![F']\!](s) \Rightarrow [\![C']\!]_{\mathrm{T}}(s)$$

which can be justified as follows (a formal proof is omitted):

- From (1b) and the rules, we can easily deduce that in every derivation $C' \downarrow F'$ matching the premise of a rule with conclusion $C \downarrow F$, the formula F' has no free variables.
- Likewise, from (1c) and the rules, we can deduce that in every such derivation, the formula F' does not depend on the poststate: we never explicitly introduce primed variables, we only use expressions T with $E \simeq T$ for some E, and we only use formulas that are either termination conditions or (in the rules for command sequences and loops) specifications; in the later case, all primed variables are removed from the condition whose values in the poststate is different from their values in their prestate.

The corollary follows from the soundness theorem and Theorem "Termination Condition". □

3.4.1 Assignment

$$I = E \downarrow F$$

This rule states that an assignment always terminates. Thus we can e.g. derive

$$\texttt{x = x+x*y} \downarrow \texttt{TRUE}$$

Soundness Proof We have to show

(a) $[\![F]\!](s) \Rightarrow [\![I = E]\!]_{\mathrm{T}}(s)$

By the definition of $[\![\_]\!]_{\mathrm{T}}$, it suffices to show

(b) TRUE

such that we are done. □

3.4.2 Variable Declaration

$$\frac{\$I \text{ does not occur in } F \qquad C \downarrow \texttt{EXISTS \$}I : F[\$I/I]}{\texttt{var } I\texttt{; } C \downarrow F}$$

This rule says that, for proving that a variable declaration terminates under some condition, it suffices to prove that the declaration body terminates under this condition where a mathematical variable replaces the program variable declared. For instance, to prove

```
var x; while (i<n) i=i+j ↓ x>0 AND j=x
```

it suffices to prove

```
while (i<n) i=i+j ↓ EXISTS $x: $x>0 AND j=$x
```

which can be simplified to

```
while (i<n) i=i+j ↓ j > 0
```

Soundness Proof We have to show

(a) $[\![F]\!](s) \Rightarrow [\![\texttt{var } I\texttt{; } C]\!]_{\mathrm{T}}(s)$

We assume

(2) $[\![F]\!](s)$

By the definition of $[\![\_]\!]_{\mathrm{T}}$, it suffices to show

(b) $\forall v \in \mathit{Value} : [\![C]\!]_{\mathrm{T}}(\mathit{write}(s, I, v))$

Take arbitrary $v \in \mathit{Value}$. We show

(c) $[\![C]\!]_{\mathrm{T}}(\mathit{write}(s,I,v))$

From the first premise, we know

(2a) $\$I$ does not occur in F

From the second premise and the induction hypothesis we know

(3) $\forall s \in \mathit{State} : [\![\texttt{EXISTS } \$I : F[\$I/I]]\!](s) \Rightarrow [\![C]\!]_{\mathrm{T}}(s)$

We define

(4) $s_0 := \mathit{write}(s,I,v)$

From (3) and (4), it thus suffices to show

(d) $[\![\texttt{EXISTS } \$I : F[\$I/I]]\!](s_0)$

i.e., by the definition of $[\![\_]\!]$, for arbitrary $e \in \mathit{Environment}$,

(e) $\exists v \in \mathit{Value} : [\![F[\$I/I]]\!](e[I \mapsto v])(s_0,s_0)$

From (2) and the definition of $[\![\_]\!]$, we know

(5) $[\![F]\!](e)(s,s)$

From (1a), (4), and (WS), we know

(6) $s_0 = s$ EXCEPT I

From (2a), (5), (6), and (PMVF1), we know

(7) $[\![F[\$I/I]]\!](e[I \mapsto \mathit{read}(s,I)])(s_0,s_0)$

and thus (e). □

3.4.3 Variable Definition

$$\frac{\begin{array}{l} E \simeq T \\ \$I \text{ does not occur in } T \text{ and } F \\ C \downarrow \texttt{EXISTS } \$I : I{=}T[\$I/I] \texttt{ AND } F[\$I/I] \end{array}}{\texttt{var } I{=}E\texttt{; } C \downarrow F}$$

This rule says that, for proving that a variable definition terminates under some condition, it suffices to prove that the declaration body terminates under this condition where a mathematical term replaces the program expression that yields the variable value. For instance, to prove

```
var x=j*x; while (i<n) i=i+x ↓ x>0 AND j=x
```

it suffices to prove

```
j*x ≃ TIMES(j,x)
while (i<n) i=i+x ↓
  EXISTS $x: x=TIMES(j,$x) AND $x>0 AND j=$x
```

where the latter can be simplified to

```
while (i<n) i=i+x ↓ x=TIMES(j,j) AND j>0
```

Soundness Proof We have to show

(a) $[\![F]\!](s) \Rightarrow [\![\texttt{var } I{=}E\texttt{; } C]\!]_{\mathrm{T}}(s)$

We assume

(2) $[\![F]\!](s)$

By the definition of $[\![\_]\!]_{\mathrm{T}}$, it suffices to show

(b) $[\![C]\!]_{\mathrm{T}}(\mathit{write}(s, I, [\![E]\!](s)))$

From the first premise, we know

(2a) $\$I$ does not occur in T and F

From the other premises, the definition of $\simeq$, and the induction hypothesis, we know

(3) T has no free variables

(4) $\forall s, s' \in \mathit{Store}, e \in \mathit{Environment} : [\![E]\!](s) = [\![T]\!](e)(s, s')$

(5) $\forall s \in \mathit{State} : [\![\texttt{EXISTS \$}I : I\texttt{=}T[\texttt{\$}I/I] \texttt{ AND } F[\texttt{\$}I/I]]\!](s) \Rightarrow [\![C]\!]_{\mathrm{T}}(s)$

We define

(6) $s_0 := \mathit{write}(s, I, [\![E]\!](s))$

From (5) and (6), it suffices to show

(c) $[\![\texttt{EXISTS \$}I : I\texttt{=}T[\texttt{\$}I/I] \texttt{ AND } F[\texttt{\$}I/I]]\!](s_0)$

i.e., by the definition of $[\![\_]\!]$, for arbitrary $e \in \mathit{Environment}$,

(d) $\exists v \in \mathit{Value} :$
$\mathit{read}(s_0, I) = [\![T[\texttt{\$}I/I]]\!](e[I \mapsto v])(s_0, s_0) \wedge$
$[\![F[\texttt{\$}I/I]]\!](e[I \mapsto v])(s_0, s_0)$

We show this for $v := \mathit{read}(s, I)$, i.e.

(d.1) $\mathit{read}(s_0, I) = [\![T[\texttt{\$}I/I]]\!](e[I \mapsto \mathit{read}(s, I)])(s_0, s_0)$

(d.2) $[\![F[\texttt{\$}I/I]]\!](e[I \mapsto \mathit{read}(s, I)])(s_0, s_0)$

From (1a), (6), (WS), we know

(7) $s_0 = s \text{ EXCEPT } I$

From (2) and the definition of $[\![\_]\!]$, we know

(8) $[\![F]\!](e)(s, s)$

From (2a), (7), (8), and (PMVF1), we know (d.2).

From (4) and (6), we know

(9) $s_0 = \mathit{write}(s, I, [\![T]\!](e)(s, s))$

From (9) and (RW1), we know

(10) $\mathit{read}(s_0, I) = [\![T]\!](e)(s, s)$

From (10), to show (d.1), it suffices to show

(d.1.a) $[\![T]\!](e)(s, s) = [\![T[\texttt{\$}I/I]]\!](e[I \mapsto \mathit{read}(s, I)])(s_0, s_0)$

We know this from (2a), (7), and (PMVT1). □

3.4.4 Command Sequence (Basic Rule)

$$\frac{C_1 \downarrow F \qquad C_2 \downarrow \texttt{TRUE}}{C_1\,;C_2 \downarrow F}$$

This rule says that, for proving that a command sequence terminates under a certain condition, it suffices to prove that the first command of the sequence terminates under this condition and that the second command terminates in any case. For instance, for proving

```
while (x>0) x=x-i; y=x+1 ↓i > 0
```

it suffices to prove

```
while (x>0) x=x-i ↓i > 0
y=x+1 ↓ TRUE
```

This rule therefore puts on the one hand a strong obligation on the termination proof of the second command, but has on the other hand the advantage that it does not depend on the specification of the transition relation of the first command.

Soundness Proof We have to show

(a) $[\![F]\!](s) \Rightarrow [\![C_1\,;\,C_2]\!]_{\mathrm{T}}(s)$

We assume

(2) $[\![F]\!](s)$

By the definition of $[\![\,\_\,]\!]_{\mathrm{T}}$, it suffices to show

(b.1) $[\![C_1]\!]_{\mathrm{T}}(s)$
(b.2) $\forall s' \in \mathit{State} : [\![C_1]\!](s,s') \Rightarrow [\![C_2]\!]_{\mathrm{T}}(s')$

From the premises, the induction hypothesis, and the definition of $[\![\,\_\,]\!]$, we know

(3) $\forall s \in \mathit{State} : [\![F]\!](s) \Rightarrow [\![C_1]\!]_{\mathrm{T}}(s)$
(4) $\forall s \in \mathit{State} : [\![C_2]\!]_{\mathrm{T}}(s)$

From (2) and (3), we know (b.1). From (4), we know (b.2). □

3.4.5 Command Sequence (Advanced Rule)

$$\frac{\begin{array}{l} C_1 \downarrow F \\ C_1 : [S]_{I_1,\ldots,I_n} \\ \$I_1,\ldots,\$I_n \text{ do not occur in } F \text{ and } S \\ C_2 \downarrow \texttt{EXISTS } \$I_1,\ldots,\$I_n : \\ \qquad F[\$I_1/I_1,\ldots,\$I_n/I_n] \texttt{ AND} \\ \qquad S[\$I_1/I_1,\ldots,\$I_n/I_n, I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}] \end{array}}{C_1\texttt{;}\, C_2 \downarrow F}$$

This rule says that, for proving that a command sequence terminates under a certain condition, it suffices to prove that the first command of the sequence terminates under this condition and that the second command terminates in all states that may result from the execution of the first command. For instance, for proving

```
while (i<0) i=i+n; while (j>0) j=j-i ↓
  n>0 AND !(n|i)
```

it suffices to prove

```
while (i<0) i=i+n ↓
  n>0 AND !(n|i)
while (i<0) i=i+n:
  [i'>=0 AND n|(i'-i)]_i
while (j>0) j=j-i ↓
  EXISTS $i:
    n>0 AND !(n|$i) AND i>=0 AND n|(i-$i)
```

Soundness Proof We have to show

(a) $[\![F]\!](s) \Rightarrow [\![C_1\texttt{;}\, C_2]\!]_{\mathrm{T}}(s)$

We assume

(2) $[\![F]\!](s)$

By the definition of $[\![\_]\!]_{\mathrm{T}}$, it suffices to show

(b.1) $[\![C_1]\!]_{\mathrm{T}}(s)$

(b.2) $\forall s' \in \mathit{State} : [\![C_1]\!](s,s') \Rightarrow [\![C_2]\!]_{\mathrm{T}}(s')$

From the premises, the induction hypothesis, and the definition of $[\![\_]\!]$, we know

(3) $\forall s \in \mathit{State} : [\![F]\!](s) \Rightarrow [\![C_1]\!]_{\mathrm{T}}(s)$

(4) $\forall s, s' \in \mathit{State}, e \in \mathit{Environment} : [\![C_1]\!](s, s') \Rightarrow [\![[S]_{I_1,\ldots,I_n}]\!](e)(s, s')$

(4a) $\$I_1, \ldots, \$I_n$ do not occur in F and S

(5)
$$\begin{array}{l} \forall s \in \mathit{State} : \\ \quad [\![\texttt{EXISTS } \$I_1, \ldots, \$I_n : \\ \quad\quad F[\$I_1/I_1, \ldots, \$I_n/I_n] \texttt{ AND} \\ \quad\quad S[\$I_1/I_1, \ldots, \$I_n/I_n, I_1/I_1\texttt{'}, \ldots, I_n/I_n\texttt{'}\,]]\!](s) \Rightarrow [\![C_2]\!]_{\mathrm{T}}(s) \end{array}$$

From (2) and (3), we know (b.1).

To show (b.2), we take arbitrary $s' \in \mathit{State}$ and assume

(6) $[\![C_1]\!](s, s')$

We have to show

(b.2.a) $[\![C_2]\!]_{\mathrm{T}}(s')$

To prove this, by (5) and the definition of $[\![\_]\!]$, it suffices to prove for arbitrary but fixed $e \in \mathit{Environment}$

(b.2.b)
$$\begin{array}{l} \exists v_1, \ldots, v_n \in \mathit{Value} : \\ \quad [\![F[\$I_1/I_1, \ldots, \$I_n/I_n]]\!] \\ \quad\quad (e[I_1 \mapsto v_1, \ldots, I_n \mapsto v_n])(s', s') \wedge \\ \quad [\![S[\$I_1/I_1, \ldots, \$I_n/I_n, I_1/I_1\texttt{'}, \ldots, I_n/I_n\texttt{'}\,]]\!] \\ \quad\quad (e[I_1 \mapsto v_1, \ldots, I_n \mapsto v_n])(s', s') \end{array}$$

We prove this for $v_1 := \mathit{read}(s, I_1), \ldots, v_n := \mathit{read}(s, I_n)$ i.e.

(b.2.b.1)
$$\begin{array}{l} [\![F[\$I_1/I_1, \ldots, \$I_n/I_n]]\!] \\ \quad (e[I_1 \mapsto \mathit{read}(s, I_1), \ldots, I_n \mapsto \mathit{read}(s, I_n)])(s', s') \end{array}$$

(b.2.b.2)
$$\begin{array}{l} [\![S[\$I_1/I_1, \ldots, \$I_n/I_n, I_1/I_1\texttt{'}, \ldots, I_n/I_n\texttt{'}\,]]\!] \\ \quad (e[I_1 \mapsto \mathit{read}(s, I_1), \ldots, I_n \mapsto \mathit{read}(s, I_n)])(s', s') \end{array}$$

From (4), (6), and the definition of $[\![\_]\!]$ and $[\, \_ \,]_{\_}$, we know

(7) $[\![S]\!](e)(s, s')$

(8) $s = s'$ EXCEPT $I_1, \ldots, I_n$

From (2) and the definition of $[\![\_]\!]$, we know

(9) $[\![F]\!](e)(s, s)$

From (1c) and (9), we know

(10) $[\![F]\!](e)(s,s')$

From (4a), (8), (10), and (PMVF1), we know (b.2.b.1).

From (4a), (7), (8), and (PMVF1), we know

(11) $[\![S[\$I_1/I_1,\ldots,\$I_n/I_n]]\!]$
$(e[I_1 \mapsto read(s,I_1),\ldots,I_n \mapsto read(sI_n)])(s',s')$

From (IDE), we know

(12) $s' = writes(s',I_1,read(s',I_1),\ldots,I_n,read(s',I_n))$

From (11), (12), and (PPVF2), we know (b.2.b.2). □

3.4.6 One-Sided Conditional

$$\frac{E \simeq T \quad C \downarrow F \texttt{ AND } T}{\texttt{if (}E\texttt{) } C \downarrow F}$$

This rule says that, for proving that a one-sided conditional statement terminates under condition F, it suffices to prove that the command in the if-branch of the statement terminates when both F and the branch condition are true.

For example, for proving

```
if (k>0) while (i>0) i=i+j*k ↓ j>0
```

it suffices to prove

```
k>0 ≃ k>0
while (i>0) i=i+j*k ↓ j>0 AND k>0
```

Soundness Proof We have to show

(a) $[\![F]\!](s) \Rightarrow [\![\texttt{if (}E\texttt{) } C]\!]_{\mathrm{T}}(s)$

We assume

(2) $[\![F]\!](s)$

By the definition of $[\![\_]\!]_T$, it suffices to show

(b) $[\![E]\!](s) = \text{TRUE} \Rightarrow [\![C]\!]_T(s)$

We assume

(3) $[\![E]\!](s) = \text{TRUE}$

and show

(c) $[\![C]\!]_T(s)$

From the premises and the induction hypothesis, we know

(4) $E \simeq T$

(5) $\forall s \in \mathit{State} : [\![F \texttt{ AND } T]\!](s) \Rightarrow [\![C]\!]_T(s)$

By (5) and the definition of $[\![\_]\!]$, to show (c) it suffices to show

(c.1) $[\![F]\!](s)$

(c.2) $[\![T]\!](s)$

From (2), we know (c.1). From (3), (4), and the definition of $\simeq$, we know (c.2). □

3.4.7 Two-Sided Conditional

$$\frac{\begin{array}{l} E \simeq T \\ C_1 \downarrow F \texttt{ AND } T \\ C_2 \downarrow F \texttt{ AND !} T \end{array}}{\texttt{if (}E\texttt{) } C_1 \texttt{ else } C_2 \downarrow F}$$

This rule says that, for proving that a two-sided conditional statement terminates under condition F, it suffices to prove that the command in each branch of the statement terminates when both F and the branch condition (respectively its negation) are true.

For example, for proving

```
if (k>0)
  while (i>0) i=i-k
else
  while (i>0) i=i+k ↓ k!=0
```

it suffices to prove

```
k!=0 ≃ k!=0
while (i>0) i=i-k ↓ k!=0 AND k>0
while (i>0) i=i+k ↓ k!=0 AND !(k>0)
```

Soundness Proof We have to show

(a) $[\![F]\!](s) \Rightarrow [\![\texttt{if (}E\texttt{)}\ C_1\ \texttt{else}\ C_2]\!]_{\mathrm{T}}(s)$

We assume

(2) $[\![F]\!](s)$

By the definition of $[\![\_]\!]_{\mathrm{T}}$, it suffices to show

(b.1) $[\![E]\!](s) = \texttt{TRUE} \Rightarrow [\![C_1]\!]_{\mathrm{T}}(s)$

(b.2) $[\![E]\!](s) \neq \texttt{TRUE} \Rightarrow [\![C]\!]_{\mathrm{T}}(s)$

From the premises and the induction hypothesis, we know

(4) $E \simeq T$

(5) $\forall s \in \mathit{State} : [\![F\ \texttt{AND}\ T]\!](s) \Rightarrow [\![C_1]\!]_{\mathrm{T}}(s)$

(6) $\forall s \in \mathit{State} : [\![F\ \texttt{AND}\ !T]\!](s) \Rightarrow [\![C_2]\!]_{\mathrm{T}}(s)$

To show (b.1), we assume

(7) $[\![E]\!](s) = \texttt{TRUE}$

and show

(b.1.a) $[\![C_1]\!]_{\mathrm{T}}(s)$

By (5) and the definition of $[\![\_]\!]$, it suffices to show

(b.1.a.1) $[\![F]\!](s)$

(b.1.a.2) $[\![T]\!](s)$

From (2), we know (b.1.a.1). From (4), (7), and the definition of $\simeq$, we know (b.1.a.2).

To show (b.2), we assume

(8) $[\![E]\!](s) \neq \texttt{TRUE}$

and show

(b.2.a) $[\![C_2]\!]_{\mathrm{T}}(s)$

By (6) and the definition of $[\![\_]\!]$, it suffices to show

(b.2.a.1) $[\![F]\!](s)$

(b.2.a.2) $\neg [\![T]\!](s)$

From (2), we know (b.2.a.1). From (4), (8), and the definition of $\simeq$, we know (b.2.a.2). $\square$

3.4.8 While Loop (Without Invariant)

$$\frac{\forall s \in \mathit{State} : [\![F \texttt{ => } \texttt{!}\, E]\!](s)}{\texttt{while (}E\texttt{) } C \downarrow F}$$

If no invariant is given, the only way to make sure that a loop terminates is to make sure that it is never entered.

Soundness Proof We have to show

(a) $[\![F]\!](s) \Rightarrow [\![\texttt{while (}E\texttt{) } C]\!]_{\mathrm{T}}(s)$

We assume

(2) $[\![F]\!](s)$

By the definition of $[\![\_]\!]_{\mathrm{T}}$, it suffices to show for arbitrary $t \in \mathit{State}^{\infty}, k \in \mathbb{N}$

(b.1) $\neg \mathit{infiniteExecution}(t, s, [\![E]\!], [\![C]\!])$

(b.2) $\mathit{finiteExecution}(k, t, s, [\![E]\!], [\![C]\!]) \Rightarrow$
$[\![E]\!](t(k)) = \mathrm{TRUE} \Rightarrow [\![C]\!]_{\mathrm{T}}(t(k))$

From the premise and the definition of $[\![\_]\!]$, we know

(3) $\forall s \in \mathit{State} : [\![F]\!](s) \Rightarrow \neg [\![E]\!](s)$

From (2) and (3), we know

(4) $\neg [\![E]\!](s)$

From (4) and the definition of *infiniteExecution*, we know (b.1).

To show (b.2), we assume

(5) $\mathit{finiteExecution}(k, t, s, [\![E]\!], [\![C]\!])$

(6) $[\![E]\!](t(k)) = \mathrm{TRUE}$

(7) $\neg [\![C]\!]_{\mathrm{T}}(t(k))$

and show a contradiction.

From (4), (5), and the definition of *finiteExecution*, we have a contradiction. □

3.4.9 While Loop (With Invariant)

$$\frac{\begin{array}{l} E \simeq H \\ C : [S]_{I_1,\ldots,I_n} \\ \mathit{Invariant}(G,H,S)_{I_1,\ldots,I_n} \\ T \text{ has no free variables and no primed program variables} \\ \forall s \in \mathit{State} : [\![F \Rightarrow G[I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]]\!](s) \\ C \downarrow \texttt{EXISTS } \$I_1,\ldots,\$I_n : F[\$I_1/I_1,\ldots,\$I_n/I_n] \texttt{ AND} \\ \qquad G[\$I_1/I_1,\ldots,\$I_n/I_n,I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}] \texttt{ AND } H \\ \forall e \in \mathit{Environment}, s, s' \in \mathit{Store}, v_1,\ldots,v_n \in \mathit{Value} : \\ \quad \text{LET } e_0 = e[I_1 \mapsto v_1,\ldots,I_n \mapsto v_n] \text{ IN} \\ \qquad [\![F[\$I_1/I_1,\ldots,\$I_n/I_n] \texttt{ AND} \\ \qquad\quad G[\$I_1/I_1,\ldots,\$I_n/I_n,I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}] \texttt{ AND } H \texttt{ AND} \\ \qquad\quad [S]_{I_1,\ldots,I_n}]\!](e_0)(s,s') \Rightarrow \\ \qquad \text{LET} \\ \qquad\quad m = [\![T]\!](e_0)(s,s'), \\ \qquad\quad m' = [\![T[I_1\texttt{'}/I_1,\ldots,I_n\texttt{'}/I_n]]\!](e_0)(s,s') \\ \qquad \text{IN } m \in \mathbb{N} \wedge m > m' \end{array}}{\texttt{while (}E\texttt{) } C \downarrow F}$$

This rule essentially says that, for proving that a while loop terminates under condition F, it suffices to derive a loop invariant G and a "termination term" T that only contains plain program variables such that

1. F implies that G holds in the initial state (i.e. before the loop is entered),
2. the loop body C terminates in every prestate related to the initial state by G for which the loop expression E yields true, and
3. T denotes a natural number which is decreased by every loop iteration (from which it follows that the number of iterations is bounded).

For instance, for proving

```
while (i>=0 && i>=j) {i=i-j; j=j+k} ↓
  int(i) AND int(j) AND int(k) AND k>0
```

with the knowledge

```
i<j ≃ i<j
i=i-j; j=j+k : [i'=i-j AND j'=j-k]_{i,j}
Invariant(int(i') AND int(j'),
  i<j, i'=i-j AND j'=j-k)
```

it suffices to derive the termination term

```
IF j>0 THEN i ELSE i+j*j+1
```

and then to show

$\forall s \in \mathit{State}$:

(1) $[\![$`int(i) AND int(j) AND int(k) AND k>0 =>`
`int(i) AND int(j)` $]\!](s)$

`i=i-j; j=j+k` $\downarrow$

(2)
```
EXISTS $i, $j:
  int($i) AND int($j) AND int(k) AND k>0 AND
  int(i) AND int(j) AND i>=0 AND i>=j
```

$\forall e \in \mathit{Environment}, s, s' \in \mathit{Store}, v_i, v_j \in \mathit{Value}$:

LET $e_0 = e[\mathtt{i} \mapsto v_i, \mathtt{j} \mapsto v_j]$ IN

(3) $[\![$`int($i) AND int($j) AND int(k) AND k>0 AND`
`int(i) AND int(j) AND i>=0 AND i>=j AND`
`i'=i-j AND j'=j+k AND writesonly i,j` $]\!](e_0)(s,s') \Rightarrow$

LET

$m = [\![$`IF j>0 THEN i ELSE i+j*j+1` $]\!](e_0)(s,s')$

$m' = [\![$`IF j'>0 THEN i' ELSE i'+j'*j'+1` $]\!](e_0)(s,s')$

IN $m \in \mathbb{N} \wedge m > m'$

Obligation (3) can be replaced by

$\forall e \in \mathit{Environment}, s, s' \in \mathit{State}$:

(3') $[\![$
```
FORALL $i,$j:
  int($i) AND int($j) AND int(k) AND k>0 AND
  int(i) AND int(j) AND i>=0 AND i>=j AND
  i'=i-j AND j'=j+k AND writesonly i,j =>
  LET
    m0 = IF j>0 THEN i ELSE i+j*j+1,
    m1 = IF j'>0 THEN i' ELSE i'+j'*j'+1
  IN nat(m0) AND m0>m1
```
$]\!](e)(s,s')$

assuming that the predicates `nat` and > have the corresponding interpretation.

Soundness Proof We have to show

(a) $[\![F]\!](s) \Rightarrow [\![\texttt{while (}E\texttt{) } C]\!]_{\mathrm{T}}(s)$

We assume

(2) $[\![F]\!](s)$

By the definition of $[\![\_]\!]_{\mathrm{T}}$, it suffices to show for arbitrary $t \in \mathit{State}^{\infty}, k \in \mathbb{N}$

(a.1) $\neg \mathit{infiniteExecution}(t, s, [\![E]\!], [\![C]\!])$

(a.2) $\mathit{finiteExecution}(k, t, s, [\![E]\!], [\![C]\!]) \Rightarrow$
$[\![E]\!](t(k)) = \mathrm{TRUE} \Rightarrow [\![C]\!]_{\mathrm{T}}(t(k))$

From the premises, the soundness of the verification calculus, and the induction hypothesis, we know

(3) $E \simeq H$

(4) $\forall s, s' \in \mathit{State}, e \in \mathit{Environment} : [\![C]\!](s, s') \Rightarrow [\![[S]_{I_1,\ldots,I_n}]\!](e)(s, s')$

(5) $\mathit{Invariant}(G, H, S)_{I_1,\ldots,I_n}$

(6) T has no free variables and no primed program variables

(7) $\forall s \in \mathit{State} : [\![F \texttt{ => } G[I_1/I_1\texttt{'}, \ldots, I_n/I_n\texttt{'}]]\!](s)$

(8)
$\forall s \in \mathit{State} :$
$\quad [\![\texttt{EXISTS } \$I_1, \ldots, \$I_n : F[\$I_1/I_1, \ldots, \$I_n/I_n] \texttt{ AND}$
$\quad\quad G[\$I_1/I_1, \ldots, \$I_n/I_n, I_1/I_1\texttt{'}, \ldots, I_n/I_n\texttt{'}] \texttt{ AND } H]\!](s) \Rightarrow [\![C]\!]_{\mathrm{T}}(s)$

(9)
$\forall e \in \mathit{Environment}, s, s' \in \mathit{Store}, v_1, \ldots, v_n \in \mathit{Value} :$
$\quad \mathrm{LET}\ e_0 = e[I_1 \mapsto v_1, \ldots, I_n \mapsto v_n]\ \mathrm{IN}$
$\quad\quad [\![F[\$I_1/I_1, \ldots, \$I_n/I_n] \texttt{ AND}$
$\quad\quad\quad G[\$I_1/I_1, \ldots, \$I_n/I_n, I_1/I_1\texttt{'}, \ldots, I_n/I_n\texttt{'}] \texttt{ AND } H \texttt{ AND}$
$\quad\quad\quad [S]_{I_1,\ldots,I_n}]\!](e_0)(s, s') \Rightarrow$
$\quad\quad \mathrm{LET}$
$\quad\quad\quad m = [\![T]\!](e_0)(s, s'),$
$\quad\quad\quad m' = [\![T[I_1\texttt{'}/I_1, \ldots, I_n\texttt{'}/I_n]]\!](e_0)(s, s')$
$\quad\quad \mathrm{IN}\ m \in \mathbb{N} \wedge m > m'$

From (5) and the definition of *Invariant*, we know

(9a) $\$I_1, \ldots, \$I_n$ do not occur in $G, H,$ and F

To show (a.2), we assume

(10) $\mathit{finiteExecution}(k,t,s,[\![E]\!],[\![C]\!])$

(11) $[\![E]\!](t(k)) = \text{TRUE}$

and show

(a.2.a) $[\![C]\!]_{\text{T}}(t(k))$

From (8), it suffices to show

(a.2.b) $[\![\texttt{EXISTS } \$I_1,\ldots,\$I_n : F[\$I_1/I_1,\ldots,\$I_n/I_n] \texttt{ AND}$
$G[\$I_1/I_1,\ldots,\$I_n/I_n,I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}] \texttt{ AND } H]\!](t(k))$

From the definition of $[\![\,\_\,]\!]$, it suffices to show for arbitrary $e \in \mathit{Environment}$,

(a.2.c)
$\exists v_1,\ldots,v_n \in \mathit{Value}:$
LET $e_0 = e[I_1 \mapsto v_1,\ldots,I_n \mapsto v_n]$ IN
$[\![F[\$I_1/I_1,\ldots,\$I_n/I_n]]\!](e_0)(t(k),t(k)) \wedge$
$[\![G[\$I_1/I_1,\ldots,\$I_n/I_n,I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]]\!](e_0)(t(k),t(k)) \wedge$
$[\![H]\!](e_0)(t(k),t(k))$

We show this for $v_1 = \mathit{read}(s,I_1),\ldots,v_n = \mathit{read}(s,I_n)$, i.e. we define

(12) $e_0 := e[I_1 \mapsto \mathit{read}(s,I_1),\ldots,I_n \mapsto \mathit{read}(s,I_n)]$

and show

(a.2.c.1) $[\![F[\$I_1/I_1,\ldots,\$I_n/I_n]]\!](e_0)(t(k),t(k))$

(a.2.c.2) $[\![G[\$I_1/I_1,\ldots,\$I_n/I_n,I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]]\!](e_0)(t(k),t(k))$

(a.2.c.3) $[\![H]\!](e_0)(t(k),t(k))$

From (10) and the definition of *finiteExecution*, we know

(13) $t(0) = s$

(14) $\forall i \in \mathbb{N}_k : [\![E]\!](t(i)) = \text{TRUE} \wedge [\![C]\!](t(i),t(i+1))$

From (4), (14), and the definition of $[\,\_\,]_{\_}$, we know

(15) $\forall i \in \mathbb{N}_k : [\![S]\!](e)(t(i),t(i+1)) \wedge t(i) = t(i+1)$ EXCEPT $I_1,\ldots,I_n$

From (13), (15), and (TRE), we know

(16) $s = t(k)$ EXCEPT $I_1,\ldots,I_n$

From (1c), (2), and the definition of $[\![\_]\!]$, we know

(17) $[\![F]\!](e)(s,s')$

From (9a), (12), (16), (17), and (PMVF1), we know (a.2.c.1).

From (7), (17), and the definition of $[\![\_]\!]$, we know

(18) $[\![G[I_1/I_1{}', \ldots, I_n/I_n{}']]\!](e)(s,s)$

From (9a), (12), (16), (18), and (PMVF1), we know (a.2.c.2).

From (3), (11), and the definition of $\simeq$, we know (a.2.c.3).

To show (a.1), we assume

(19) $\mathit{infiniteExecution}(t,s,[\![E]\!],[\![C]\!])$

and show a contradiction. We take an arbitrary $e \in \mathit{Environment}$ and define

(20) $m : \mathit{Value}^{\infty}, m(i) = [\![T]\!](e)(t(i),t(i+1))$

Since $\langle \mathbb{N}, > \rangle$ is a well-founded ordering, it suffices to show

(a.1.b) $\forall i \in \mathbb{N} : m(i) \in \mathbb{N} \wedge m(i) > m(i+1)$

Take arbitrary $i \in \mathbb{N}$. We show

(a.1.b.1) $m(i) \in \mathbb{N}$

(a.1.b.2) $m(i) > m(i+1)$

We define

(21) $e_0 := e[I_1 \mapsto \mathit{read}(s,I_1), \ldots, I_n \mapsto \mathit{read}(s,I_n)]$

From (9), (21), and the definition of $[\![\_]\!]$, we know

(22)
$$
\begin{array}{l}
([\![F[\$I_1/I_1, \ldots, \$I_n/I_n]]\!](e_0)(t(i),t(i+1)) \wedge \\
\quad [\![G[\$I_1/I_1, \ldots, \$I_n/I_n, I_1/I_1{}', \ldots, I_n/I_n{}']]\!](e_0)(t(i),t(i+1)) \wedge \\
\quad [\![H]\!](e_0)(t(i),t(i+1)) \wedge \\
\quad [\![S]\!](e_0)(t(i),t(i+1) \wedge \\
\quad t(i) = t(i+1) \text{ EXCEPT } I_1, \ldots, I_n) \Rightarrow \\
\text{LET} \\
\quad m = [\![T]\!](e_0)(t(i),t(i+1)), \\
\quad m' = [\![T[I_1{}'/I_1, \ldots, I_n{}'/I_n]]\!](e_0)(t(i),t(i+1)) \\
\text{IN } m \in \mathbb{N} \wedge m > m'
\end{array}
$$

From (2) and the definition of $[\![\_]\!]$, we know

(23) $[\![F]\!](e)(s,s)$

From (19), and the definition of *infiniteExecution*, we know

(24) $t(0) = s$
(25) $\forall i \in \mathbb{N} : [\![E]\!](t(i)) = \text{TRUE}$
(26) $\forall i \in \mathbb{N} : [\![C]\!](t(i),t(i+1))$

From (4), (26), and the definition of $[\_]_\_$, we know

(27) $\forall i \in \mathbb{N} : [\![S]\!](e_0)(t(i),t(i+1))$
(28) $\forall i \in \mathbb{N} : t(i) = t(i+1)$ EXCEPT $I_1,\ldots,I_n$

From (24), (28), and (TRE), we know

(29) $s = t(i)$ EXCEPT $I_1,\ldots,I_n$
(30) $s = t(i+1)$ EXCEPT $I_1,\ldots,I_n$

From (1c) and (23), we know

(31) $[\![F]\!](e)(s,t(i+1))$

From (9a), (21), (29), (31), and (PMVF1), we know

(32) $[\![F[\$I_1/I_1,\ldots,\$I_n/I_n]]\!](e_0)(t(i),t(i+1))$

From (3), (4), (5), and (19), we can show (as presented in the soundness proof the Invariant Rule in Section 3.2.2)

(33) $[\![G]\!](e)(s,t(i))$

From (5), (9a), (21), (29), (33), the def. of *Invariant*, and (PMVF1), we can show

(34) $[\![G[\$I_1/I_1,\ldots,\$I_n/I_n]]\!](e_0)(t(i),t(i))$

From (34), (IDE), and (PPVF1), we can show

(35) $[\![G[\$I_1/I_1,\ldots,\$I_n/I_n,I_1/I_1{}',\ldots,I_n/I_n{}']]\!](e_0)(t(i),t(i))$

From (28), (35), (PPVF0), and (PVF4), we know

(36) $[\![G[\$I_1/I_1,\ldots,\$I_n/I_n,I_1/I_1{}',\ldots,I_n/I_n{}']\!](e_0)(t(i),t(i+1))$

From (3), (25), and the definition of $\simeq$, we know

(37) $[\![H]\!](e_0)(t(i),t(i+1))$

We define

(38) $m_0 = [\![T]\!](e_0)(t(i),t(i+1))$
(39) $m_1 = [\![T[I_1{}'/I_1,\ldots,I_n{}'/I_n]]\!](e_0)(t(i),t(i+1))$

From (22), (27), (28), (32), (36), (37), (38), and (39), we know

(40) $m_0 \in \mathbb{N}$
(41) $m_0 > m_1$

From (6), (38), and (MVT), we know

(42) $m_0 = [\![T]\!](e)(t(i),t(i+1))$

From (20), (40), and (42), we know (a.1.b.1).

We define

(43) $s_0 := writes(t(i),I_1,read(t(i+1),I_1),\ldots,I_n,read(t(i+1),I_n))$

From (39), (43), and (PPVT1), we know

(44) $m_1 = [\![T]\!](e_0)(s_0,t(i+1))$

From (1a), (43), and (WSE), we know

(45) $s_0 = t(i)$ EXCEPT $I_1,\ldots,I_n$

From (28), (45), and (TRE), we know

(46) $s_0 = t(i+1)$ EXCEPT $I_1,\ldots,I_n$

From (43) and (RWE), we know

(47) $read(s_0,I_1) = read(t(i+1),I_1) \wedge \ldots \wedge$
$read(s_0,I_n) = read(t(i+1),I_n)$

From (46), (47), (RVE), and (NEQ), we know

(48) s_0 EQUALS $t(i+1)$

From (44), (48), and (EST), we know

(49) $m_1 = [\![T]\!](e_0)(t(i+1),t(i+1))$

From (6), (49), (MVT), and (PVT2), we know

(50) $m_1 = [\![T]\!](e)(t(i+1),t(i+2))$

From (20), (41), (42), and (50), we know (a.1.b.2). □

Chapter 4

Reasoning on Programs and States

In this chapter, we describe three calculi which are in practice useful for reasoning on programs and the states on which they operate: first, a *precondition calculus* derives a condition which is necessary to hold before the execution of a command to make another condition true afterwards; next, a *postcondition calculus* derives from a condition known to hold before the execution of a command another condition known to hold afterwards; finally, an *assertion* calculus annotates every subcommand with a condition known to hold when the subcommand is executed.

In the presence of loops, the computed precondition may not be the weakest ones (i.e., even if it is violated, the postcondition may be satisfied); likewise, the computed postcondition/assertion may not be the strongest one. Nevertheless, if loop invariants are available, the calculi take these into account and compute the best conditions that can be deduced directly (without sophisticated program analysis) from the information available. In this sense, they reflect the typical reasoning of human programmers when developing code.

4.1 Computing Command Preconditions

Given a command C and a condition Q, we would like to find out under which condition the execution of C yields a post-state that satisfies Q. More precisely, we would like to derive a condition P such that, if P holds on the prestate of C and if the execution of C yields a post-state, this post-state satisfies Q. For instance for command

```
x = x+1
```

and condition `x>0` on the poststate we can derive the condition `x+1>0` (i.e. `x>=0`, if `x` holds an integer number) on the prestate.

We call P a *precondition* of C for postcondition Q and describe in this section a calculus for deriving such preconditions. A precondition P derived by this calculus is not necessarily the weakest one i.e. it is possible that it is not satisfied by a prestate from which the execution of C nevertheless yields a poststate that satisfies Q. Actually, all rules but one compute weakest preconditions; the only exception is the rule for the while command which depends on an invariant provided by some external source from which a precondition, but not the weakest one, can be computed (one might actually also in this case compute a weakest precondition, but such a precondition would involve a fixed point construction which is of little use in practical reasoning).

Figures 4.1, 4.2, and 4.3 give the rules of this calculus which derive judgements of the form $\text{PRE}(C,Q) = P$ with the informal interpretation "P a precondition of C for postcondition Q".

The first rule is *generic*, i.e. applicable to any kind of command: it shows that it suffices, by the rule of the verification calculus, to derive the specification of a command to compute a suitable precondition. In a sense, thus this rule is all we need. However, in practice it might be sometimes easier to use rules which do not refer to the verification calculus but are specialized to the individual kinds of commands; in the following, we also give such rules.

The following theorem states the formal soundness claim for this calculus.

Theorem (Soundness of the Precondition Calculus) Assume the condition denoted by *DifferentVariables*. If $\text{PRE}(C,Q) = P$ can be derived from the rules of the precondition calculus of the command language, then it is true that

$$
\begin{array}{l}
Q \text{ has no primed program variables} \Rightarrow \\
\quad P \text{ has no primed program variables} \ \wedge \\
\quad \forall e \in \mathit{Environment}, s, s' \in \mathit{State} : \\
\qquad [\![P]\!](e)(s,s) \wedge [\![C]\!](s,s') \Rightarrow [\![Q]\!](e)(s',s')
\end{array}
$$

Proof Assume

(1a) *DifferentVariables*

Take C, Q, and P such that $\text{PRE}(C,Q) = P$ can be derived and assume

(1b) Q has no primed program variables

Precondition Calculus: Judgements

$\text{PRE}(C,Q) = P \Leftrightarrow$
Q has no primed program variables $\Rightarrow$
P has no primed program variables $\wedge$
$\forall e \in \mathit{Environment}, s, s' \in \mathit{State}$:
$[\![P]\!](e)(s,s) \wedge [\![C]\!](s,s') \Rightarrow [\![Q]\!](e)(s',s')$

Precondition Calculus: Generic Rule

$C : [F]_{I_1,\ldots,I_n}$
$J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$
$\$J_1,\ldots,\$J_n$ do not occur in F and Q

$\text{PRE}(C,Q) =$
FORALL $\$J_1,\ldots,\$J_n$:
$F[\$J_1/I_1\text{'},\ldots,\$J_n/I_n\text{'}]$ => $Q[\$J_1/I_1,\ldots,\$J_n/I_n]$

Figure 4.1: The Precondition Calculus of the Command Language (Part 1/3)

From (1b) and the rules, it is easy to show by induction on the derivation of $\text{PRE}(C,Q) = P$ that P has no primed program variables.

Now take arbitrary $e \in \mathit{Environment}$ and $s, s' \in \mathit{State}$. We prove

$$[\![P]\!](e)(s,s) \wedge [\![C]\!](s,s') \Rightarrow [\![Q]\!](e)(s',s')$$

by induction on the derivation of $\text{PRE}(C,Q) = P$. The following subsections cover all cases for the last step of such a derivation.

From (1b) and the rules, we can immediately deduce that in every derivation $\text{PRE}(C',Q') = P'$ matching the premise of a rule with conclusion $\text{PRE}(C,Q) = P$, the formula Q' has no primed variables; we thus assume in the proofs that the induction hypothesis immediately implies the core claim $\forall e \in \mathit{Environment}, s, s' \in \mathit{State} : [\![P']\!](e)(s,s) \wedge [\![C']\!](s,s') \Rightarrow [\![Q']\!](e)(s',s')$. $\square$

4.1.1 Generic Rule

$C : [F]_{I_1,\ldots,I_n}$
$J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$
$\$J_1,\ldots,\$J_n$ do not occur in F and Q

$\text{PRE}(C,Q) =$
FORALL $\$J_1,\ldots,\$J_n$:
$F[\$J_1/I_1\text{'},\ldots,\$J_n/I_n\text{'}]$ => $Q[\$J_1/I_1,\ldots,\$J_n/I_n]$

Precondition Calculus: Rules for Non-Loops

$$\frac{E \simeq T \quad \$J \text{ does not occur in } Q}{\textsc{pre}(I\texttt{=}E, Q) = \texttt{LET } \$J\texttt{=}T \texttt{ IN } Q[\$J/I]}$$

$$\frac{\$J \text{ does not occur in } P \quad \$K \text{ does not occur in } Q \quad J \neq K \quad \textsc{pre}(C, Q[\$K/I]) = P}{\textsc{pre}(\texttt{var } I\texttt{;}\, C, Q) = \texttt{FORALL } \$J : P[\$J/I][I/\$K]}$$

$$\frac{E \simeq T \quad \$J \text{ does not occur in } P \quad \$K \text{ does not occur in } Q \quad J \neq K \quad \textsc{pre}(C, Q[\$K/I]) = P}{\textsc{pre}(\texttt{var } I\texttt{=}E\texttt{;}\, C, Q) = \texttt{LET } \$J\texttt{=}T \texttt{ IN } P[\$J/I][I/\$K]}$$

$$\frac{\textsc{pre}(C_1, P) = O \quad \textsc{pre}(C_2, Q) = P}{\textsc{pre}(C_1\texttt{;}\, C_2, Q) = O}$$

$$\frac{E \simeq F \quad \textsc{pre}(C, Q) = P}{\textsc{pre}(\texttt{if (}E\texttt{)}\ C, Q) = \texttt{IF } F \texttt{ THEN } P \texttt{ ELSE } Q}$$

$$\frac{E \simeq F \quad \textsc{pre}(C_1, Q) = P_1 \quad \textsc{pre}(C_2, Q) = P_2}{\textsc{pre}(\texttt{if (}E\texttt{)}\ C_1 \texttt{ else } C_2, Q) = \texttt{IF } F \texttt{ THEN } P_1 \texttt{ ELSE } P_2}$$

Figure 4.2: The Precondition Calculus of the Command Language (Part 2/3)

Precondition Calculus: Rules for Loops

$$\frac{\begin{array}{l} E \simeq H \\ C : [F]_{I_1,\ldots,I_n} \\ J_1,\ldots,J_n \text{ is a renaming of } I_1,\ldots,I_n \\ \$J_1,\ldots,\$J_n \text{ do not occur in } Q \text{ and } H \end{array}}{\begin{array}{l} \texttt{PRE}(\texttt{while (}E\texttt{) } C, Q) = \\ \quad \texttt{FORALL } \$J_1,\ldots,\$J_n\texttt{:} \\ \quad\quad !H[\$J_1/I_1,\ldots,\$J_n/I_n] \texttt{ => } Q[\$J_1/I_1,\ldots,\$J_n/I_n] \end{array}}$$

$$\frac{\begin{array}{l} E \simeq H \\ C : [F]_{I_1,\ldots,I_n} \\ \mathit{Invariant}(G,H,F)_{I_1,\ldots,I_n} \\ J_1,\ldots,J_n \text{ is a renaming of } I_1,\ldots,I_n \\ \$J_1,\ldots,\$J_n \text{ do not occur in } Q, H, \text{ and } G \end{array}}{\begin{array}{l} \texttt{PRE}(\texttt{while (}E\texttt{) } C, Q) = \\ \quad \texttt{FORALL } \$J_1,\ldots,\$J_n\texttt{:} \\ \quad\quad !H[\$J_1/I_1,\ldots,\$J_n/I_n] \texttt{ AND} \\ \quad\quad (G[I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}] \texttt{ => } G[\$J_1/I_1\texttt{'},\ldots,\$J_n/I_n\texttt{'}]) \texttt{ => } \\ \quad\quad Q[\$J_1/I_1,\ldots,\$J_n/I_n] \end{array}}$$

Figure 4.3: The Precondition Calculus of the Command Language (Part 3/3)

Soundness Proof We have to show

$$\text{(a)}\quad \begin{array}{l} [\![\texttt{FORALL } \$J_1,\ldots,\$J_n\texttt{:} \\ \quad F[\$J_1/I_1\texttt{'},\ldots,\$J_n/I_n\texttt{'}]\texttt{ => }Q[\$J_1/I_1,\ldots,\$J_n/I_n]\,]\!](e)(s,s) \wedge \\ [\![C]\!](s,s') \\ \Rightarrow \\ [\![Q]\!](e)(s',s') \end{array}$$

We assume

$$\text{(2)}\quad \begin{array}{l} [\![\texttt{FORALL } \$J_1,\ldots,\$J_n\texttt{:} \\ \quad F[\$J_1/I_1\texttt{'},\ldots,\$J_n/I_n\texttt{'}]\texttt{ => }Q[\$J_1/I_1,\ldots,\$J_n/I_n]\,]\!](e)(s,s) \end{array}$$

$$\text{(3)}\quad [\![C]\!](s,s')$$

and show

$$\text{(b)}\quad [\![Q]\!](e)(s',s')$$

From the hypotheses, we know

(4) $C : [\,F\,]_{I_1,\ldots,I_n}$

(5) $J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$

(6) $\$J_1,\ldots,\$J_n$ do not occur in F and Q

From (1a), (3), (4), and the soundness of the verification calculus, we know

(7) $[\![\,[\,F\,]_{I_1,\ldots,I_n}\,]\!](e)(s,s')$

From (7) and the definition of $[\![\,\_\,]\!]$, we know

(8) $[\![F]\!](e)(s,s')$

(9) $s = s'$ EXCEPT $I_1,\ldots,I_n$

We define

(10) $e_0 := e[J_1 \mapsto \mathit{read}(s',I_1),\ldots,J_n \mapsto \mathit{read}(s',I_n)\,]$

From (2), (10), and the definition of $[\![\,\_\,]\!]$, we know

$$\text{(11)}\quad \begin{array}{l} [\![F[\$J_1/I_1\texttt{'},\ldots,\$J_n/I_n\texttt{'}]\,]\!](e_0)(s,s) \Rightarrow \\ [\![Q[\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e_0)(s,s) \end{array}$$

From (5), (6), (8), (9), (10), and (PMVF2), we know

(12) $[\![F[\$J_1/I_1{}', \ldots, \$J_n/I_n{}']]\!](e_0)(s,s)$

From (11) and (12), we know

(13) $[\![Q[\$J_1/I_1, \ldots, \$J_n/I_n]]\!](e_0)(s,s)$

From (5), (6), (9), (10), (13) and (PMVF1), we know

(14) $[\![Q]\!](e)(s',s)$

From (1b), (14), and (PVF2), we know (b). □

4.1.2 Assignment

$$\frac{E \simeq T \quad \$J \text{ does not occur in } Q}{\text{PRE}(I{=}E, Q) = \texttt{LET \$}J\texttt{=}T\texttt{ IN }Q[\$J/I]}$$

This rule states that an assignment to variable I yields a poststate in which Q holds for I, if Q holds in the prestates for the value assigned to I (denoted by the mathematical variable $\$J$). For instance, for command

```
x = x*y+z
```

and postcondition `x>0 AND y>0`, a precondition is

```
LET $x=x*y+z IN $x>0 AND y>0
```

which can be further simplified to

```
x*y+z>0 AND y>0
```

Soundness Proof We have to show

(a) $[\![\texttt{LET \$}J\texttt{=}T\texttt{ IN }Q[\$J/I]]\!](e)(s,s) \wedge [\![I{=}E]\!](s,s') \Rightarrow [\![Q]\!](e)(s',s')$

We assume

(2) $[\![\texttt{LET \$}J\texttt{=}T\texttt{ IN }Q[\$J/I]]\!](e)(s,s)$

(3) $[\![I{=}E]\!](s,s')$

and show

(b) $[\![Q]\!](e)(s',s')$

From the premise, we know

(4) $E \simeq T$

(5) $\$J$ does not occur in Q

From (2) and the definition of $[\![\_]\!]$, we know

(6) $[\![Q[\$J/I]]\!](e[J \mapsto [\![T]\!](e)(s,s)])(s,s)$

From (3) and the definition of $[\![\_]\!]$, we know

(7) $s' = write(s,I,[\![E]\!](s))$

From (RW1) and (7), we know

(8) $read(s',I) = [\![E]\!](s)$

From (4), (8), and the definition of $\simeq$, we know

(9) $read(s',I) = [\![T]\!](e)(s,s)$

From (1a), (7), and (WS), we know

(9) $s' = s$ EXCEPT I

From (5), (6), (9), (10), and (PMVF1), we know

(11) $[\![Q]\!](e)(s',s)$

From (1b), (11), and (PVF2), we know (b). □

4.1.3 Variable Declaration

$$\frac{\begin{array}{l}\$J \text{ does not occur in } P \\ \$K \text{ does not occur in } Q \\ J \neq K \\ \text{PRE}(C, Q[\$K/I]) = P\end{array}}{\text{PRE}(\texttt{var } I\texttt{;}\, C, Q) = \texttt{FORALL } \$J : P[\$J/I][I/\$K]}$$

This rule states that to compute the precondition for a variable declaration block `var` I`;` C with postcondition Q, we must first compute for C the precondition P of $Q[\$K/I]$ where the fresh mathematical variable `$`K denotes the value of I after the execution of the block (which is different from the value of I after the execution of C, because after the execution of the block I is restored to the value it had before the execution of the block). The precondition P is thus sufficient to ensure postcondition Q after the execution of C, for an unknown value `$`K of I.

We then introduce another fresh mathematical variable `$`J which denotes the value of I before the execution of C. Thus $P[\$J/I]$ is a precondition on the value `$`J of I before the execution of C that is sufficient to ensure Q on an unknown value `$`K of I after the execution of C. Correspondingly, $P[\$J/I][I/\$K]$ is a precondition on the value `$`J of I before the execution of C to ensure Q on the value of I after the execution of the block. Since the value of I before the execution of C is arbitrary, this condition must hold for all possible values of `$`J, thus the overall precondition is `FORALL $`$J : P[\$J/I][I/\$K]$.

As an example, take the block

```
var y; x=x+y*y
```

and postcondition `x>0 AND y>0`. We introduce variable `$yk` for the post-block value of `y` and compute the precondition of the assignment statement for `x>0 AND $yk>0` which is

```
x+y*y>0 AND $yk>0
```

We then introduce the variable `$yj` for the pre-assignment value of `y` and substitute in this condition `$yj` for `y` yielding

```
x+$yj*$yj>0 AND $yk>0
```

In this condition, we substitute back `y` for `$yk` yielding

```
x+$yj*$yj>0 AND y>0
```

The overall precondition is thus

```
FORALL $yj: x+$yj*$yj>0 AND y>0
```

which can be further simplified to `x>0 AND y>0`.

Soundness Proof We have to show

(a) $[\![\texttt{FORALL \$}J : P[\$J/I][I/\$K]]\!](e)(s,s) \wedge [\![\texttt{var } I\texttt{;} C]\!](s,s') \Rightarrow$
$[\![Q]\!](e)(s',s')$

We assume

(2) $[\![\texttt{FORALL \$}J : P[\$J/I][I/\$K]]\!](e)(s,s)$

(3) $[\![\texttt{var } I\texttt{;} C]\!](s,s')$

and show

(b) $[\![Q]\!](e)(s',s')$

From (2) and the definition of $[\![\_]\!]$, we know

(4) $\forall v \in \mathit{Value} : [\![P[\$J/I][I/\$K]]\!](e[J \mapsto v])(s,s)$

From (3) and the definition of $[\![\_]\!]$, we know for some $s_0, s_1 \in \mathit{State}$

(5) $s_0 = s$ EXCEPT I

(6) $[\![C]\!](s_0,s_1)$

(7) $s' = \mathit{write}(s_1, I, \mathit{read}(s,I))$

From (7) and (RW1), we know

(8) $\mathit{read}(s',I) = \mathit{read}(s,I)$

From (1a), (7), and (WS), we know

(9) $s' = s_1$ EXCEPT I

From the premises, we know

(10) $\$J$ does not occur in P

(11) $\$K$ does not occur in Q

(12) $J \neq K$

(13) $\mathrm{PRE}(C, Q[\$K/I]) = P$

From (13) and the induction hypothesis, we know

(14) $\forall e \in \mathit{Environment}$:
$[\![P]\!](e)(s_0,s_0) \wedge [\![C]\!](s_0,s_1) \Rightarrow [\![Q[\$K/I]]\!](e)(s_1,s_1)$

and thus

$$(15)\quad \begin{array}{l}[\![P]\!](e[K \mapsto read(s',I)\,])(s_0,s_0) \wedge [\![C]\!](s_0,s_1) \Rightarrow \\ \quad [\![Q[\$K/I]\,]\!](e[K \mapsto read(s',I)\,])(s_1,s_1)\end{array}$$

Assume that we can show

$$(c)\quad [\![P]\!](e[K \mapsto read(s',I)\,])(s_0,s_0)$$

From (6), (15), and (c), we know

$$(16)\quad [\![Q[\$K/I]\,]\!](e[K \mapsto read(s',I)\,])(s_1,s_1)$$

From (9), (11), (16), and (PMVF1), we know

$$(17)\quad [\![Q]\!](e)(s',s_1)$$

From (1b), (17), and (PVF2), we know (b).

It remains to show (c). We define

$$(18)\quad P' := P[\$J/I]$$

From (4) and (18), we know

$$(19)\quad [\![P'[I/\$K]\,]\!](e[J \mapsto read(s_0,I)\,])(s,s)$$

From (PMVF0), we know

$$(20)\quad \$K \text{ does not occur in } P'[I/\$K]$$

From (8), (19), (20), (RE), and (PMVF1), we know

$$(21)\quad [\![P'[I/\$K][\$K/I]]\!](e[J \mapsto read(s_0,I)\,][K \mapsto read(s',I)\,])(s,s)$$

From (18) and (MPVF0), we know

$$(22)\quad I \text{ does not occur in } P'$$

From (22) and (MPVF2), we know

$$(23)\quad P'[I/\$K][\$K/I] = P'$$

From (18), (21), and (23), we know

(24) $[\![P[\$J/I]]\!](e[J \mapsto read(s_0,I)][K \mapsto read(s',I)])(s,s)$

From (12) and (24), we know

(25) $[\![P[\$J/I]]\!](e[K \mapsto read(s',I)][J \mapsto read(s_0,I)])(s,s)$

From (5), (10), (25), and (PMVF1), we know

(26) $[\![P]\!](e[K \mapsto read(s',I)])(s_0,s)$

From (13) and the induction hypothesis, we know

(27) P has no primed program variables

From (26), (27), and (PVF2), we know (c). □

4.1.4 Variable Definition

$$\frac{\begin{array}{l}E \simeq T\\ \$J \text{ does not occur in } P\\ \$K \text{ does not occur in } Q\\ J \neq K\\ \text{PRE}(C, Q[\$K/I]) = P\end{array}}{\text{PRE}(\texttt{var } I\texttt{=}E\texttt{;}\,C, Q) = \texttt{LET } \$J\texttt{=}T \texttt{ IN } P[\$J/I][I/\$K]}$$

This rule is very similar to the one for variable declarations (see Subsection 4.1.3). The major exception is that the value of I before the execution of C is uniquely determined by the program expression E, respectively by its mathematical counterpart T, such that in the resulting pre-condition the variable $\$J$ is introduced by `LET $J=T` (rather than by `FORALL $J`).

As an example, take the block

```
var y=x; x=x+y*y
```

and postcondition `x>0 AND y>0`. With the same sequence of steps as the example of Subsection 4.1.3, we derive the condition

```
x+$yj*$yj>0 AND y>0
```

The overall precondition is thus

```
LET $yj=x IN x+$yj*$yj>0 AND y>0
```

which can be further simplified to `x+x*x>0 AND y>0`.

Soundness Proof We have to show

(a) $[\![\texttt{LET \$}J\texttt{=}T\ \texttt{IN}\ P[\$J/I][I/\$K]]\!](e)(s,s) \wedge [\![\texttt{var}\ I\texttt{=}E\texttt{;}C]\!](s,s') \Rightarrow [\![Q]\!](e)(s',s')$

We assume

(2) $[\![\texttt{LET \$}J\texttt{=}T\ \texttt{IN}\ P[\$J/I][I/\$K]]\!](e)(s,s)$

(3) $[\![\texttt{var}\ I\texttt{=}E\texttt{;}C]\!](s,s')$

and show

(b) $[\![Q]\!](e)(s',s')$

From (2) and the definition of $[\![\_]\!]$, we know

(4) $[\![P[\$J/I][I/\$K]]\!](e[J \mapsto [\![T]\!](e)(s,s)])(s,s)$

From (3) and the definition of $[\![\_]\!]$, we know for some $s_0, s_1 \in \mathit{State}$

(5) $s_0 = \mathit{write}(s, I, [\![E]\!]s)$

(6) $[\![C]\!](s_0, s_1)$

(7) $s' = \mathit{write}(s_1, I, \mathit{read}(s, I))$

From (7) and (RW1), we know

(8) $\mathit{read}(s', I) = \mathit{read}(s, I)$

From (1a), (7), and (WS), we know

(9) $s' = s_1$ EXCEPT I

From the premises, we know

(10) $E \simeq T$

(11) $\$J$ does not occur in P

(12) $\$K$ does not occur in Q

(13) $J \neq K$

(14) $\mathrm{PRE}(C, Q[\$K/I]) = P$

From (14) and the induction hypothesis, we know

(15) $$\begin{array}{l}\forall e \in \mathit{Environment}: \\ \quad [\![P]\!](e)(s_0,s_0) \wedge [\![C]\!](s_0,s_1) \Rightarrow [\![Q[\$K/I]]\!](e)(s_1,s_1)\end{array}$$

and thus

(16) $$\begin{array}{l}[\![P]\!](e[K \mapsto \mathit{read}(s',I)])(s_0,s_0) \wedge [\![C]\!](s_0,s_1) \Rightarrow \\ \quad [\![Q[\$K/I]]\!](e[K \mapsto \mathit{read}(s',I)])(s_1,s_1)\end{array}$$

Assume that we can show

(c) $[\![P]\!](e[K \mapsto \mathit{read}(s',I)])(s_0,s_0)$

From (6), (16), and (c), we know

(17) $[\![Q[\$K/I]]\!](e[K \mapsto \mathit{read}(s',I)])(s_1,s_1)$

From (9), (12), (17), and (PMVF1), we know

(18) $[\![Q]\!](e)(s',s_1)$

From (1b), (18), and (PVF2), we know (b).

It remains to show (c). We define

(19) $P' := P[\$J/I]$

From (4) and (19), we know

(20) $[\![P'[I/\$K]]\!](e[J \mapsto [\![T]\!](e)(s,s)])(s,s)$

From (5), (10), (20), (RW1), and the definition of $\simeq$, we know

(21) $[\![P'[I/\$K]]\!](e[J \mapsto \mathit{read}(s_0,I)])(s,s)$

From (PMVF0), we know

(22) $\$K$ does not occur in $P'[I/\$K]$

From (8), (21), (22), (RE), and (PMVF1), we know

(23) $[\![P'[I/\$K][\$K/I]]\!](e[J \mapsto \mathit{read}(s_0,I)][K \mapsto \mathit{read}(s',I)])(s,s)$

From (19) and (MPVF0), we know

(24) I does not occur in P'

From (24) and (MPVF2), we know

(25) $P'[I/\$K][\$K/I] = P'$

From (19), (23), and (25), we know

(26) $[\![P[\$J/I]]\!](e[J \mapsto read(s_0, I)][K \mapsto read(s', I)])(s, s)$

From (13) and (26), we know

(27) $[\![P[\$J/I]]\!](e[K \mapsto read(s', I)][J \mapsto read(s_0, I)])(s, s)$

From (5), (11), (27), and (PMVF1), we know

(28) $[\![P]\!](e[K \mapsto read(s', I)])(s_0, s)$

From (14) and the induction hypothesis, we know

(29) P has no primed program variables

From (28), (29), and (PVF2), we know (b). □

4.1.5 Command Sequence

$$\frac{\text{PRE}(C_1, P) = O \quad \text{PRE}(C_2, Q) = P}{\text{PRE}(C_1 ; C_2, Q) = O}$$

This rule describes how we can "back-propagate" the computation of a precondition O from a postcondition Q through a command sequence $C_1 ; C_2$. We first compute the precondition P of C_2 for Q and then compute the precondition O of C_1 for P.

As an example, take the program

```
x=x+y; y=y+x
```

with postcondition

```
x=a AND y=b
```

The precondition of the second assignment for this postcondition can be derived as

```
x=a AND y+x=b
```

The precondition of the first assignment for this postcondition is

```
x+y=a AND y+(x+y)=b
```

which is the overall precondition of the sequence.

Soundness Proof We have to show

(a) $[\![O]\!](e)(s,s) \wedge [\![C_1 ; C_2]\!](s,s') \Rightarrow [\![Q]\!](e)(s',s')$

We assume

(2) $[\![O]\!](e)(s,s)$

(3) $[\![C_1 ; C_2]\!](s,s')$

and show

(b) $[\![Q]\!](e)(s',s')$

From the premises, we know by the induction hypothesis

(4) $\forall e \in \mathit{Environment}, s, s' \in \mathit{State}:$
$[\![O]\!](e)(s,s) \wedge [\![C_1]\!](s,s') \Rightarrow [\![P]\!](e)(s',s')$

(5) $\forall e \in \mathit{Environment}, s, s' \in \mathit{State}:$
$[\![P]\!](e)(s,s) \wedge [\![C_2]\!](s,s') \Rightarrow [\![Q]\!](e)(s',s')$

From (3) and the definition of $[\![\_]\!]$, we know for some $s_0 \in \mathit{State}$

(6) $[\![C_1]\!](s,s_0)$

(7) $[\![C_2]\!](s_0,s')$

From (2), (4), and (6), we know

(8) $[\![P]\!](e)(s_0,s_0)$

From (5), (7), and (8), we know (b). □

4.1.6 One-Sided Conditional

$$\frac{E \simeq F \qquad \text{PRE}(C,Q) = P}{\text{PRE}(\texttt{if (}E\texttt{) }C, Q) = \texttt{IF } F \texttt{ THEN } P \texttt{ ELSE } Q}$$

This rule states that the precondition of a one-sided conditional statement `if (E) C` for post-condition Q can be derived by computing the precondition P of C for Q and then constructing a conditional pre-condition where, depending on the value of the mathematical counterpart F of the branch condition E, either P or Q holds (because, if the branch condition yields false, the state is not changed).

As an example, take the program

```
if (x<0) x=x*x
```

with postcondition `x>0`. From the branch command, we can derive the precondition `x*x>0` which yields the overall precondition

```
IF x<0 THEN x*x>0 ELSE x>0
```

which can be further simplified to `x/=0`.

Soundness Proof We have to show

(a) $[\![\texttt{IF}\ F\ \texttt{THEN}\ P\ \texttt{ELSE}\ Q]\!](e)(s,s) \wedge [\![\texttt{if (}E\texttt{)}\ C]\!](s,s') \Rightarrow [\![Q]\!](e)(s',s')$

We assume

(2) $[\![\texttt{IF}\ F\ \texttt{THEN}\ P\ \texttt{ELSE}\ Q]\!](e)(s,s)$

(3) $[\![\texttt{if (}E\texttt{)}\ C]\!](s,s')$

and show

(b) $[\![Q]\!](e)(s',s')$

From the premises, we know by the induction hypothesis

(4) $E \simeq F$

(5) $[\![P]\!](e)(s,s) \wedge [\![C]\!](s,s') \Rightarrow [\![Q]\!](e)(s',s')$

We proceed by case distinction.

- Case $[\![E]\!](s) = \texttt{TRUE}$: from (3), the definition of $[\![\_]\!]$, and the case condition, we know

  (6) $[\![C]\!](s,s')$

  From (4), the definition of $\simeq$, and the case condition, we know

  (7) $[\![F]\!](e)(s,s)$

  From (2), (7) and the definition of$[\![_]\!]$, we know

  (8) $[\![P]\!](e)(s,s)$.

  From (5), (6), and (8), we know (b).

- Case $[\![E]\!](s) \neq \text{TRUE}$: from (3), the definition of $[\![\_]\!]$, and the case condition, we know

  (9) $s' = s$

  From (4), the definition of $\simeq$, and the case condition, we know

  (10) $\neg[\![F]\!](e)(s,s)$

  From (2), (10), and the definition of $[\![\_]\!]$, we know

  (11) $[\![Q]\!](e)(s,s)$

  From (9) and (11), we know (b). □

4.1.7 Two-Sided Conditional

$$\frac{\begin{array}{l} E \simeq F \\ \text{PRE}(C_1, Q) = P_1 \\ \text{PRE}(C_2, Q) = P_2 \end{array}}{\text{PRE}(\texttt{if (}E\texttt{) } C_1 \texttt{ else } C_2, Q) = \texttt{IF } F \texttt{ THEN } P_1 \texttt{ ELSE } P_2}$$

This rule states that the precondition of a conditional `if (`E`) `C_1 `else` C_2 for post-condition Q can be derived by computing the precondition P_1 of C_1 for Q respectively P_2 of C_2 for Q and then constructing a conditional pre-condition where, depending on the value of the mathematical counterpart F of the branch condition E, either P_1 or P_2 holds.

As an example, take the program

```
if (x<0) x=x*x else x=x+1
```

with postcondition `x>0`. From the branch commands, we can derive the preconditions `x*x>0` respectively `x+1>0` which yields the overall precondition

```
IF x<0 THEN x*x>0 ELSE x+1>0
```

which can be further simplified to `TRUE`.

Soundness Proof We have to show

(a) $[\![\texttt{IF}\ F\ \texttt{THEN}\ P_1\ \texttt{ELSE}\ P_2]\!](e)(s,s) \wedge [\![\texttt{if (}E\texttt{)}\ C_1\ \texttt{else}\ C_2]\!](s,s') \Rightarrow [\![Q]\!](e)(s',s')$

We assume

(2) $[\![\texttt{IF}\ F\ \texttt{THEN}\ P_1\ \texttt{ELSE}\ P_2]\!](e)(s,s)$

(3) $[\![\texttt{if (}E\texttt{)}\ C_1\ \texttt{else}\ C_2]\!](s,s')$

and show

(b) $[\![Q]\!](e)(s',s')$

From the premises, we know by the induction hypothesis

(4) $E \simeq F$

(5) $[\![P_1]\!](e)(s,s) \wedge [\![C_1]\!](s,s') \Rightarrow [\![Q]\!](e)(s',s')$

(6) $[\![P_2]\!](e)(s,s) \wedge [\![C_2]\!](s,s') \Rightarrow [\![Q]\!](e)(s',s')$

We proceed by case distinction.

- Case $[\![E]\!](s) = \textsc{true}$: from (3), the definition of $[\![\_]\!]$, and the case condition, we know

  (7) $[\![C_1]\!](s,s')$

  From (4), the definition of $\simeq$, and the case condition, we know

  (8) $[\![F]\!](e)(s,s)$

  From (2), (8) and the definition of$[\![_]\!]$, we know

  (9) $[\![P_1]\!](e)(s,s)$.

  From (5), (7), and (9), we know (b).

- Case $[\![E]\!](s) \neq \textsc{true}$: from (3), the definition of $[\![\_]\!]$, and the case condition, we know

  (10) $[\![C_2]\!](s,s')$

  From (4), the definition of $\simeq$, and the case condition, we know

  (11) $\neg [\![F]\!](e)(s,s)$

  From (2), (11) and the definition of $[\![\_]\!]$, we know

  (12) $[\![P_2]\!](e)(s,s)$.

  From (6), (10), and (12), we know (b). $\square$

4.1.8 While Loop (Without Invariant)

$E \simeq H$
$C : [F]_{I_1,\ldots,I_n}$
$J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$
$\$J_1,\ldots,\$J_n$ do not occur in Q and H

PRE(`while (`E`)` C, Q) =
`FORALL` $\$J_1,\ldots,\$J_n$`:`
`!`$H[\$J_1/I_1,\ldots,\$J_n/I_n]$ `=>` $Q[\$J_1/I_1,\ldots,\$J_n/I_n]$

This rule states that to derive a particular postcondition of a loop, it suffices to show as a precondition that the negation of the loop condition implies the postcondition for all possible values of the variables changed by the loop body. In certain situations, it may be thus possible to derive a useful loop precondition without any further information about the loop (such as the invariant needed by the rule in Section 4.1.9).

As an example, take the program

```
while (x>0) {s=s+x; x=x-1}
```

and postcondition `s*s>=0 AND y >= 0`. Then an appropriate precondition is

```
FORALL $s, $x: $s*$s > 0 AND y >= 0
```

we can be simplified to

```
y >= 0
```

Soundness Proof We have to show

(a) $[\![$`FORALL` $\$J_1,\ldots,\$J_n$`:` `!`$H[\$J_1/I_1,\ldots,\$J_n/I_n]$ `=>` $Q[\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e)(s,s) \wedge$
$[\![$`while (`E`)` $C]\!](s,s') \Rightarrow$
$[\![Q]\!](e)(s',s')$

We assume

(2) $[\![$`FORALL` $\$J_1,\ldots,\$J_n$`:` `!`$H[\$J_1/I_1,\ldots,\$J_n/I_n]$ `=>` $Q[\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e)(s,s)$

(3) $[\![$`while (`E`)` $C]\!](s,s')$

and show

(b) $[\![Q]\!](e)(s', s')$

From the premises, we know

(4) $E \simeq H$
(5) $C : [F]_{I_1,\ldots,I_n}$
(6) $J_1, \ldots, J_n$ is a renaming of $I_1, \ldots, I_n$
(7) $\$J_1, \ldots, \$J_n$ do not occur in Q and H

From (3) and the definition of $[\![\_]\!]$, we know for some $k \in \mathbb{N}, t \in \mathit{State}^\infty$:

(8) $\mathit{finiteExecution}(k, t, s, [\![E]\!], [\![C]\!])$
(9) $[\![E]\!](t(k)) \neq \text{TRUE}$
(10) $t(k) = s'$

From (4), (9), (10), and the definition of $\simeq$, we know

(11) $\neg [\![H]\!](e)(s', s')$

From (1a), (5), (8), and the soundness of the verification calculus, we can prove (as shown in the proof of the soundness of the basic rule for loops on page 48)

(12) $s = s'$ EXCEPT $I_1, \ldots, I_n$

We define

(13) $e_0 := e[J_1 \mapsto \mathit{read}(s', I_1), \ldots, J_n \mapsto \mathit{read}(s', I_n)]$

From (2), (13), and the definition of $[\![\_]\!]$, we know

(14) $\neg [\![H[\$J_1/I_1, \ldots, \$J_n/I_n]]\!](e_0)(s, s) \Rightarrow$
$[\![Q[\$J_1/I_1, \ldots, \$J_n/I_n]]\!](e_0)(s, s)$

From (4) and the definition of $\simeq$, we know

(15) H has no free variables
(16) H has no primed program variables

From (6), (7), (11), (12), (13), and (PMVF1), we know

(17) $\neg[\![H[\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e_0)(s,s')$

From (16), (17), and (PVF2), we know

(18) $\neg[\![H[\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e_0)(s,s)$

From (14) and (18), we know

(19) $[\![Q[\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e_0)(s,s)$

From (6), (7), (12), (13), (19) and (PMVF1), we know

(20) $[\![Q]\!](e)(s',s)$

From (1b), (20), and (PVF2), we know (b). □

4.1.9 While Loop (With Invariant)

$E \simeq H$
$C : [F]_{I_1,\ldots,I_n}$
$\mathit{Invariant}(G,H,F)_{I_1,\ldots,I_n}$
$J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$
$\$J_1,\ldots,\$J_n$ do not occur in $Q,H,$ and G

PRE(`while(`E`)` C,Q) =
  `FORALL` $\$J_1,\ldots,\$J_n$ **:**
    `!`$H[\$J_1/I_1,\ldots,\$J_n/I_n]$ `AND`
    $(G[I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]$ `=>` $G[\$J_1/I_1\texttt{'},\ldots,\$J_n/I_n\texttt{'}])$ `=>`
    $Q[\$J_1/I_1,\ldots,\$J_n/I_n]$

This rule shows how an invariant can be used to strengthen the precondition of a loop for a given postcondition (in comparison to the precondition derived from the rule presented in Section 4.1.8). The first three premises of the rule are identical to those of the verification rule for while loops (see Section 3.2); they make sure that G is an invariant of the loop whose body only modifies program variables $I_1,\ldots,I_n$. Any application of the rule thus demands the proof of the correctness of the invariant (which may already have been performed in a separate step).

The derived precondition differs from the one in the rule without invariant in the following way: the fact that the invariant holds in the terminal state may be added as a hypothesis to the implication (which simplifies the proof of the conclusion) provided that the invariant holds in the initial state (which represents an additional obligation on the prestate). We do not add the requirement that the invariant must

hold in the prestate as a conjunct to the precondition, because this would yield a slightly stronger precondition: in the formulation above, it still suffices to derive the postcondition from the negation of the loop condition alone.

As an example, take the program

```
while (x>0) {s=s+x; x=x-1}
```

and postcondition `s>0`. From the rules of the verification calculus we can derive

$$\texttt{s=s+x; x=x-1} : [\texttt{s'=s+x AND x'=x-1}]_{\texttt{s,x}}$$

and thus show that only the program variables `s` and `x` are modified by the loop. We can also show (in a separate proof) that the following is a loop invariant:

```
s'>=0 AND x'>=0 AND (s'>0 OR x'>0)
```

We may thus derive the following precondition for above postcondition

```
FORALL $s, $x:
  !($x>0) AND
  (s>=0 AND x>=0 (s>0 OR x>0) =>
    $s>=0 AND $x>=0 AND ($s>0 OR $x>0))
  => $s>0
```

which can be simplified to

```
s>=0 AND x>=0 AND (s>0 OR x>0)
```

i.e. the requirement that the invariant holds in the prestate of the loop.

Soundness Proof We have to show

(a)
$$\begin{array}{l}[\![\texttt{FORALL } \$J_1,\ldots,\$J_n\texttt{:}\\ \quad !H[\$J_1/I_1,\ldots,\$J_n/I_n] \texttt{ AND}\\ \quad (G[I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}] \texttt{ => } G[\$J_1/I_1\texttt{'},\ldots,\$J_n/I_n\texttt{'}]) \texttt{ =>}\\ \quad Q[\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e)(s,s) \wedge\\ [\![\texttt{while(}E\texttt{) } C]\!](s,s') \Rightarrow\\ \quad [\![Q]\!](e)(s',s')\end{array}$$

We assume

(2) $[\![$ FORALL $\$J_1,\ldots,\$J_n$:
$\quad !H[\$J_1/I_1,\ldots,\$J_n/I_n]$ AND
$\quad (G[I_1/I_1\text{'},\ldots,I_n/I_n\text{'}]$ => $G[\$J_1/I_1\text{'},\ldots,\$J_n/I_n\text{'}])$ =>
$\quad Q[\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e)(s,s)$

(3) $[\![\texttt{while}\,(E)\ C]\!](s,s')$

and show

(b) $[\![Q]\!](e)(s',s')$

From the premises, we know

(4) $E \simeq H$

(5) $C : [F]_{I_1,\ldots,I_n}$

(6) $\mathit{Invariant}(G,H,F)_{I_1,\ldots,I_n}$

(7) $J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$

(8) $\$J_1,\ldots,\$J_n$ do not occur in $Q, H,$ and G

We define

(9) $e_0 := e[J_1 \mapsto \mathit{read}(s',I_1),\ldots,J_n \mapsto \mathit{read}(s',I_n)]$

From (2), (9), and the definition of $[\![\_]\!]$, we know

(10) $\neg[\![H[\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e_0)(s,s) \wedge$
$([\![G[I_1/I_1\text{'},\ldots,I_n/I_n\text{'}]]\!](e_0)(s,s) \Rightarrow$
$\quad [\![G[\$J_1/I_1\text{'},\ldots,\$J_n/I_n\text{'}]]\!](e_0)(s,s)) \Rightarrow$
$[\![Q[\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e_0)(s,s)$

From (4), (5), (6), we can with the verification calculus derive

(11) while $(E)\ C : [\ !H[I_1\text{'}/I_1,\ldots,I_n\text{'}/I_n]$ AND
$\quad (G[I_1/I_1\text{'},\ldots,I_n/I_n\text{'}]$ => $G)\,]_{I_1,\ldots,I_n}$

From (11) and the soundness of the verification calculus, we know

(12) $[\![\texttt{while}\,(E)\ C]\!](s,s') \Rightarrow [\![[\ !H[I_1\text{'}/I_1,\ldots,I_n\text{'}/I_n]$ AND
$\quad (G[I_1/I_1\text{'},\ldots,I_n/I_n\text{'}]$ => $G)\,]_{I_1,\ldots,I_n}]\!](e)(s,s')$

From (3), (12), and the definition of $[\![\_]\!]$, we know

(13) $\neg[\![H[I_1\text{'}/I_1,\ldots,I_n\text{'}/I_n]]\!](e)(s,s')$

(14) $[\![G[I_1/I_1\,', \ldots, I_n/I_n\,']]\!](e)(s,s') \Rightarrow [\![G]\!](e)(s,s')$

(15) $s = s'$ EXCEPT $I_1, \ldots, I_n$

From (15), (RWE), and and (RVE), we know

(16) $s' = writes(s, I_1, read(s', I_1), \ldots, I_n, read(s', I_n))$

From (13), (16), and (PPVF1), we know

(17) $\neg [\![H]\!](e)(s',s')$

From (4) and the definition of $\simeq$, we know

(18) H has no free variables

(19) H has no primed program variables

From (7), (8), (9), (15), (17), and (PMVF1), we know

(20) $\neg [\![H[\$J_1/I_1, \ldots, \$J_n/I_n]]\!](e_0)(s,s')$

From (19), (20), and (PVF2), we know

(21) $\neg [\![H[\$J_1/I_1, \ldots, \$J_n/I_n]]\!](e_0)(s,s)$

We are now going to show

(b.1.a) $[\![G[I_1/I_1\,', \ldots, I_n/I_n\,']]\!](e_0)(s,s) \Rightarrow [\![G[\$J_1/I_1\,', \ldots, \$J_n/I_n\,']]\!](e_0)(s,s)$

We assume

(22) $[\![G[I_1/I_1\,', \ldots, I_n/I_n\,']]\!](e_0)(s,s)$

and show

(b.1.b) $[\![G[\$J_1/I_1\,', \ldots, \$J_n/I_n\,']]\!](e_0)(s,s)$

From (6) and the definition of invariant, we know

(23) G has no free variables

From (22), (23), and (MVF), we know

(24) $[\![G[I_1/I_1\,', \ldots, I_n/I_n\,']]\!](e)(s,s')$

From (14) and (24), we know

(25) $[\![G]\!](e)(s,s')$

From (7), (8), (9), (15), (25), and (PMVF2), we know (b.1.b).

From (10), (21), and (b.1.a), we know

(26) $[\![Q[\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e_0)(s,s)$

From (7), (8), (9), (15), (26), and (PMVF1), we know

(27) $[\![Q]\!](e)(s',s)$

From (1b), (27), and (PVF2), we know (b). □

4.2 Computing Command Postconditions

Given a command C and a condition P on the program state in which C is executed, we would like to determine a condition Q on the program state after the execution. For instance, for command

```
x = x*x
```

and condition `x/=0` on the prestate, we can derive the condition `EXISTS $x: $x/=0 AND x=$x*$x` (which can be simplified to `x>0`) on the poststate.

We call Q a *postcondition* of C for precondition P and describe in this section a calculus for deriving such postconditions. The calculus does not necessarily yield the strongest postcondition, but this is only because of the rule(s) for while loops; the rules for all other commands derive strongest postconditions.

Figures 4.4, 4.5, and 4.6 give the rules of the calculus which derive judgements of the form $\textsc{post}(C,P) = Q$ with the informal interpretation "Q is a postcondition of C for precondition P". Like for the precondition calculus, we first give a generic rule that depends on the verification calculus before stating specific rules for the individual kinds of commands.

The following theorem states the formal soundness claim for this calculus.

Postcondition Calculus: Judgements

$\text{POST}(C,P) = Q \Leftrightarrow$
P has no primed program variables $\Rightarrow$
Q has no primed program variables $\wedge$
$\forall e \in \mathit{Environment}, s, s' \in \mathit{State}$:
$[\![P]\!](e)(s,s) \wedge [\![C]\!](s,s') \Rightarrow [\![Q]\!](e)(s',s')$

Postcondition Calculus: Generic Rule

$C : [F]_{I_1,\ldots,I_n}$
$J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$
$\$J_1,\ldots,\$J_n$ do not occur in F and P

$\text{POST}(C,P) =$
EXISTS $\$J_1,\ldots,\$J_n$: $P[\$J_1/I_1,\ldots,\$J_n/I_n]$ AND
$F[\$J_1/I_1,\ldots,\$J_n/I_n, I_1/I_1{}',\ldots,I_n/I_n{}']$

Figure 4.4: The Postcondition Calculus of the Command Language (Part 1/3)

Theorem (Soundness of the Postcondition Calculus) Assume the condition denoted by *DifferentVariables*. If $\text{POST}(C,P) = Q$ can be derived from the rules of the postcondition calculus of the command language, then it is true that

P has no primed program variables $\Rightarrow$
Q has no primed program variables $\wedge$
$\forall e \in \mathit{Environment}, s, s' \in \mathit{State}$:
$[\![P]\!](e)(s,s) \wedge [\![C]\!](s,s') \Rightarrow [\![Q]\!](e)(s',s')$

Proof Assume

(1a) *DifferentVariables*

Take C, P, and Q such that $\text{POST}(C,P) = Q$ can be derived and assume

(1b) P has no primed program variables

From (1b) and the rules, it is easy to show by induction on the derivation of $\text{POST}(C,P) = Q$ that Q has no primed program variables.

Now take arbitrary $e \in \mathit{Environment}$ and $s, s' \in \mathit{State}$. We prove

$[\![P]\!](e)(s,s) \wedge [\![C]\!](s,s') \Rightarrow [\![Q]\!](e)(s',s')$

Postcondition Calculus: Rules for Non-Loops

$$\frac{E \simeq T \quad \$J \text{ does not occur in } P}{\textsc{post}(I{=}E, P) = \texttt{EXISTS } \$J : P[\$J/I] \texttt{ AND } I{=}T[\$J/I]}$$

$$\frac{\$J \text{ does not occur in } P \quad \$K \text{ does not occur in } Q \quad J \neq K \quad \textsc{post}(C, P[\$J/I]) = Q}{\textsc{post}(\texttt{var } I\texttt{;}\, C, P) = \texttt{EXISTS } \$K : Q[\$K/I][I/\$J]}$$

$$\frac{E \simeq T \quad \$J \text{ does not occur in } P \text{ and in } T \quad \$K \text{ does not occur in } Q \quad J \neq K \quad \textsc{post}(C, P[\$J/I] \texttt{ AND } I = T[\$J/I]) = Q}{\textsc{post}(\texttt{var } I{=}E\texttt{;}\, C, P) = \texttt{EXISTS } \$K : Q[\$K/I][I/\$J]}$$

$$\frac{\textsc{post}(C_1, P) = Q \quad \textsc{post}(C_2, Q) = O}{\textsc{post}(C_1\texttt{;}\, C_2, P) = O}$$

$$\frac{E \simeq F \quad \textsc{post}(C, P \texttt{ AND } F) = Q}{\textsc{post}(\texttt{if (}E\texttt{)}\ C, P) = Q \texttt{ OR } (P \texttt{ AND !}F)}$$

$$\frac{E \simeq F \quad \textsc{post}(C_1, P \texttt{ AND } F) = Q_1 \quad \textsc{post}(C_2, P \texttt{ AND !}F) = Q_2}{\textsc{post}(\texttt{if (}E\texttt{)}\ C_1 \texttt{ else } C_2, P) = Q_1 \texttt{ OR } Q_2}$$

Figure 4.5: The Postcondition Calculus of the Command Language (Part 2/3)

Postcondition Calculus: Rules for Loops

$$\frac{\begin{array}{l} E \simeq H \\ C : [F]_{I_1,\ldots,I_n} \\ J_1,\ldots,J_n \text{ is a renaming of } I_1,\ldots,I_n \\ \$J_1,\ldots,\$J_n \text{ does not occur in } P \end{array}}{\begin{array}{l} \text{POST}(\texttt{while(}E\texttt{)}\ C, P) = \\ \quad !H\ \texttt{AND EXISTS}\ \$J_1,\ldots,\$J_n\texttt{:}\ P[\$J_1/I_1,\ldots,\$J_n/I_n] \end{array}}$$

$$\frac{\begin{array}{l} E \simeq H \\ C : [F]_{I_1,\ldots,I_n} \\ J_1,\ldots,J_n \text{ is a renaming of } I_1,\ldots,I_n \\ \$J_1,\ldots,\$J_n \text{ does not occur in } P \text{ and } G \\ \mathit{Invariant}(G,H,F)_{I_1,\ldots,I_n} \end{array}}{\begin{array}{l} \text{POST}(\texttt{while(}E\texttt{)}\ C, P) = \\ \quad !H\ \texttt{AND} \\ \quad \texttt{EXISTS}\ \$J_1,\ldots,\$J_n\texttt{:} \\ \qquad P[\$J_1/I_1,\ldots,\$J_n/I_n]\ \texttt{AND} \\ \qquad (G[I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}][\$J_1/I_1,\ldots,\$J_n/I_n] \texttt{=>} \\ \qquad\quad G[\$J_1/I_1,\ldots,\$J_n/I_n,I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]) \end{array}}$$

Figure 4.6: The Postcondition Calculus of the Command Language (Part 3/3)

by induction on the derivation of $\textsc{post}(C,P) = Q$. The following subsections cover all cases for the last step of such a derivation.

From (1b) and the rules, we can immediately deduce that in every derivation $\textsc{post}(C',P') = Q'$ matching the premise of a rule with conclusion $\textsc{post}(C,P) = Q$, the formula P' has no primed variables; we thus assume in the proofs that the induction hypothesis immediately implies the core claim $\forall e \in \mathit{Environment}, s, s' \in \mathit{State} : [\![P']\!](e)(s,s) \wedge [\![C']\!](s,s') \Rightarrow [\![Q']\!](e)(s',s')$. $\square$

4.2.1 Generic Rule

$$\frac{\begin{array}{l} C : [\,F\,]_{I_1,\ldots,I_n} \\ J_1,\ldots,J_n \text{ is a renaming of } I_1,\ldots,I_n \\ \$J_1,\ldots,\$J_n \text{ do not occur in } F \text{ and } P \end{array}}{\begin{array}{l} \textsc{post}(C,P) = \\ \quad \texttt{EXISTS } \$J_1,\ldots,\$J_n \texttt{:}\ P[\$J_1/I_1,\ldots,\$J_n/I_n] \texttt{ AND} \\ \qquad F[\$J_1/I_1,\ldots,\$J_n/I_n, I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}\,] \end{array}}$$

Soundness Proof We have to show

$$\begin{array}{ll} & [\![P]\!](e)(s,s) \wedge [\![C]\!](s,s') \Rightarrow \\ \text{(a)} & \quad [\![\texttt{EXISTS } \$J_1,\ldots,\$J_n \texttt{:}\ P[\$J_1/I_1,\ldots,\$J_n/I_n] \texttt{ AND} \\ & \qquad F[\$J_1/I_1,\ldots,\$J_n/I_n, I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}\,]]\!](e)(s',s') \end{array}$$

We assume

(2) $[\![P]\!](e)(s,s)$

(3) $[\![C]\!](s,s')$

and show

$$\begin{array}{ll} \text{(b)} & [\![\texttt{EXISTS } \$J_1,\ldots,\$J_n \texttt{:}\ P[\$J_1/I_1,\ldots,\$J_n/I_n] \texttt{ AND} \\ & \quad F[\$J_1/I_1,\ldots,\$J_n/I_n, I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}\,]]\!](e)(s',s') \end{array}$$

We define

(4) $e_0 := e[J_1 \mapsto \mathit{read}(s,I_1),\ldots,J_n \mapsto \mathit{read}(s,I_n)\,]$

From (4) and the definition of $[\![\_]\!]$, it suffices to show

(c.1) $[\![P[\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e_0)(s',s')$

(c.2) $[\![F[\$J_1/I_1,\ldots,\$J_n/I_n, I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}\,]]\!](e_0)(s',s')$

From the hypotheses, we know

(5) $C : [F]_{I_1,\ldots,I_n}$

(6) $J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$

(7) $\$J_1,\ldots,\$J_n$ do not occur in F and P

From (1a), (3), (5), the soundness of the verification calculus, and the definition of $[\![\_]\!]$, we know

(8) $[\![F]\!](e)(s,s')$

(9) $s = s'$ EXCEPT $I_1,\ldots,I_n$

From (2), (4), (6), (7), (9), and (PMVF1), we know

(10) $[\![P[\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e_0)(s',s)$

From (1b), (10), and (PVF2), we know (c.1).

From (4), (6), (7), (8), (9), and (PMVF1), we know

(11) $[\![F[\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e_0)(s',s')$

From (IDE), we know

(12) $s' = writes(s',I_1,read(s',I_1),\ldots,I_n,read(s',I_n))$

From (11), (12), and (PPVF2), we know (c.2). □

4.2.2 Assignment

$$\frac{E \simeq T \qquad \$J \text{ does not occur in } P}{\text{POST}(I{=}E,P) = \texttt{EXISTS } \$J : P[\$J/I] \texttt{ AND } I{=}T[\$J/I]}$$

This rule states that, if we know in the prestate of the assignment of the expression E to the variable I that the condition P holds for I, then we know in the poststate that there exists a value (the prestate value of I denoted by the mathematical variable $\$J$), for which P holds; furthermore, if we replace in the mathematical counterpart T of E any occurrence of I by that value we yield a term whose value equals the new value of I. For instance for command

```
x = x*y+z
```

and precondition `x>0 AND y=z AND z>0`, we derive the postcondition

```
EXISTS $x: $x>0 AND y=z AND z>0 AND x=$x*y+z
```

which can be transformed to

```
y=z AND z>0 AND EXISTS $x: $x>0 AND AND x=$x*y+z
```

Soundness Proof We have to show

(a) $[\![P]\!](e)(s,s) \wedge [\![I{=}E]\!](s,s') \Rightarrow$
$[\![\texttt{EXISTS } \$J : P[\$J/I] \texttt{ AND } I{=}T[\$J/I]]\!](e)(s',s')$

We assume

(2) $[\![P]\!](e)(s,s)$
(3) $[\![I{=}E]\!](s,s')$

and show

(b) $[\![\texttt{EXISTS } \$J : P[\$J/I] \texttt{ AND } I{=}T[\$J/I]]\!](e)(s',s')$

From the definition of $[\![\_]\!]$, it suffices to show

(c) $\exists v \in \mathit{Value}$:
$[\![P[\$J/I]]\!](e[J \mapsto v])(s',s') \wedge$
$\mathit{read}(s',I) = [\![T[\$J/I]]\!](e[J \mapsto v])(s',s')$

It thus suffices to show

(d.1) $[\![P[\$J/I]]\!](e[J \mapsto \mathit{read}(s,I)])(s',s')$
(d.2) $\mathit{read}(s',I) = [\![T[\$J/I]]\!](e[J \mapsto \mathit{read}(s,I)])(s',s')$

From the premises, we know

(4) $E \simeq T$
(5) $\$J$ does not occur in P

From (3) and the definition of $[\![\_]\!]$, we know

(6) $s' = write(s, I, [\![E]\!](s))$

From (1a), (6), and (WS), we know

(7) $s' = s$ EXCEPT I

From (2), (5), (7), and (PMVF1), we know

(8) $[\![P]\!](e)(s, s')$

From (1b), (8), and (PVF2), we know (d.1).

From (6) and (RW1), we know

(9) $read(s', I) = [\![E]\!](s)$

From (5), (7), and (PMVT1), we know

(10) $[\![T]\!](e)(s, s') = [\![T[\$J/I]]\!](e[J \mapsto read(s, I)])(s', s')$

From (4), (9), (10), and the definition of $\simeq$, we know (d.2). □

4.2.3 Variable Declaration

$$\frac{\begin{array}{l}\$J \text{ does not occur in } P \\ \$K \text{ does not occur in } Q \\ J \neq K \\ \text{POST}(C, P[\$J/I]) = Q\end{array}}{\text{POST}(\texttt{var } I\texttt{; } C, P) = \texttt{EXISTS } \$K : Q[\$K/I][I/\$J]}$$

This rule computes the postcondition for a variable declaration block `var I; C` with precondition P. Since after the declaration of I this variable has an arbitrary value, we first compute for command C the postcondition Q of the condition which states that P holds for the value of I outside the block (denoted by the mathematical variable $\$J$). Since the original value of I is restored after the block, in the actual postcondition any reference to I is replaced by an existentially quantified variable $\$K$ (denoting the value of I after the execution of C) and any reference to $\$J$ is restored to I. For instance, to derive the postcondition of program

```
var x; (x=x*x; y=y+x)
```

with precondition

```
x>0 AND y>0
```

we first compute from the assignment `x=x*x` and precondition

```
$z>0 AND y>0
```

the postcondition

```
EXISTS $x: $z>0 AND y>0 AND x=$x*$x
```

from which we derive for the assignment `y=y+x` the postcondition

```
EXISTS $x, $y:
  $z>0 AND $y>0 AND x=$x*$x AND y=$y+x
```

We then introduce an existentially quantified variable `$u`, replace any reference to `x` by `$u` and substitute `$z` back to `x`, which yields the final postcondition

```
EXISTS $u, $x, $y:
  x>0 AND $y>0 AND $u=$x*$x AND y=$y+$u
```

This can be simplified to

```
x>0 AND EXISTS $x,$y: $y>0 AND y=$y+$x*$x
```

where `$x` denotes the initial value of the locally declared variable `x` and `$y` denotes the original value of `y`.

Soundness Proof We have to show

(a) $[\![P]\!](e)(s,s) \wedge [\![\texttt{var}\ I\texttt{;}\ C]\!](s,s') \Rightarrow$
$[\![\texttt{EXISTS}\ \$K\texttt{:}\ Q[\$K/I][I/\$J]]\!](e)(s',s')$

We assume

(2) $[\![P]\!](e)(s,s)$

(3) $[\![\texttt{var}\ I\texttt{;}\ C]\!](s,s')$

and show

(b) $[\![\texttt{EXISTS}\ \$K\texttt{:}\ Q[\$K/I][I/\$J]]\!](e)(s',s')$

From (3) and the definition of $[\![\_]\!]$, we know for some $s_0, s_1 \in \mathit{State}$

(4) $s_0 = s \text{ EXCEPT } I$

(5) $[\![C]\!](s_0, s_1)$

(6) $s' = \mathit{write}(s_1, I, \mathit{read}(s, I))$

From the premises, we know

(7) $\$J$ does not occur in P

(8) $\$K$ does not occur in Q

(9) $J \neq K$

(10) $\text{POST}(C, P[\$J/I]) = Q$

From (1b), (10), and the induction hypothesis, we know

(11) Q has no primed variables

(12) $\forall e \in \mathit{Environment} :$
$[\![P[\$J/I]]\!](e)(s_0, s_0) \wedge [\![C]\!](s_0, s_1) \Rightarrow [\![Q]\!](e)(s_1, s_1)$

To show (b), it suffices by the definition of $[\![\_]\!]$ to show

(c) $\exists v \in \mathit{Value} : [\![Q[\$K/I][I/\$J]]\!](e[K \mapsto v])(s', s')$

To show (c), it suffices to show

(d) $[\![Q[\$K/I][I/\$J]]\!](e[K \mapsto \mathit{read}(s_1, I)])(s', s')$

From (2), (4), (7), and (PMVF1), we know

(13) $[\![P[\$J/I]]\!](e[J \mapsto \mathit{read}(s, I)])(s_0, s)$

From (1b), (13), and (PVF2), we know

(14) $[\![P[\$J/I]]\!](e[J \mapsto \mathit{read}(s, I)])(s_0, s_0)$

From (5), (12), (14), we know

(15) $[\![Q]\!](e[J \mapsto \mathit{read}(s, I)])(s_1, s_1)$

From (6) and (WS), we know

(16) $s' = s_1 \text{ EXCEPT } I$

From (8), (15), (16), and (PMVF1), we know

(17) $[\![Q[\$K/I]]\!](e[J \mapsto read(s,I)][K \mapsto read(s_1,I)])(s',s_1)$

We define

(18) $Q' := Q[\$K/I][I/\$J]$

From (18), (MPVF0), and (MPVF2), we know

(19) $Q'[\$J/I] = Q[\$K/I]$

From (PMVF0), (RE), and (PMVF1), we know

(20) $[\![Q']\!](e[K \mapsto read(s_1,I)])(s',s') \Leftrightarrow$
$[\![Q'[\$J/I]]\!](e[K \mapsto read(s_1,I)][J \mapsto read(s',I)])(s',s')$

From (19) and (20), we know

(21) $[\![Q[\$K/I][I/\$J]]\!](e[K \mapsto read(s_1,I)])(s',s') \Leftrightarrow$
$[\![Q[\$K/I]]\!](e[K \mapsto read(s_1,I)][J \mapsto read(s',I)])(s',s')$

From (11), (17), and (PVF2), we know

(22) $[\![Q[\$K/I]]\!](e[J \mapsto read(s,I)][K \mapsto read(s_1,I)])(s',s')$

From (6) and (RW1), we know

(23) $read(s',I) = read(s,I)$

From (9), (22), and (23), we know

(24) $[\![Q[\$K/I]]\!](e[K \mapsto read(s_1,I)][J \mapsto read(s',I)])(s',s')$

From (21) and (24), we know (d). □

4.2.4 Variable Definition

$$\frac{\begin{array}{l} E \simeq T \\ \$J \text{ does not occur in } P \text{ and in } T \\ \$K \text{ does not occur in } Q \\ J \neq K \\ \textsc{post}(C, P[\$J/I] \texttt{ AND } I = T[\$J/I]) = Q \end{array}}{\textsc{post}(\texttt{var } I\texttt{=}E\texttt{;}\, C, P) = \texttt{EXISTS } \$K\texttt{:}\, Q[\$K/I][I/\$J]}$$

This rule is very similar to the one for variable declarations (see Subsection 4.2.3). The major exception is that the value of I before the execution of C is uniquely determined by the program expression E, respectively by its mathematical counterpart T, such that we add an additional equality $\$J = T$ to the precondition of C.

As an example take the program

```
var x=x*x; y=y+x
```

where the local variable `x` is defined as the square of the value of a global variable `x` (note the difference to the program in the last subsection: while there the expression `x*x` is evaluated in the context of the local variable declaration, it is here evaluated outside of this context). From the program's precondition

```
x>0 AND y>0
```

we construct for the assignment `y=y+x` the precondition

```
$z>0 AND y>0 AND x=$z*$z
```

from which we can compute the assignment's postcondition

```
EXISTS $y:
  $z>0 AND $y>0 AND x=$z*$z AND y=$y+x
```

The overall precondition is then

```
EXISTS $u, $y:
  x>0 AND $y>0 AND $u=x*x AND y=$y+$u
```

which can be simplified to

```
x>0 AND EXISTS $y: $y>0 AND y=$y+x*x
```

where `$y` denotes the original value of variable `y`. A comparison with the example in Subsection 4.2.3 clearly shows the difference in the semantics of the programs.

Soundness Proof We have to show

(a) $[\![P]\!](e)(s,s) \wedge [\![\texttt{var } I\texttt{=}E\texttt{;}C]\!](s,s') \Rightarrow$
$[\![\texttt{EXISTS } \$K\texttt{:}Q[\$K/I][I/\$J]]\!](e)(s',s')$

We assume

(2) $[\![P]\!](e)(s,s)$

(3) $[\![\texttt{var } I\texttt{=}E\texttt{;}C]\!](s,s')$

and show

(b) $[\![\texttt{EXISTS } \$K\texttt{:}Q[\$K/I][I/\$J]]\!](e)(s',s')$

From (3) and the definition of $[\![\_]\!]$, we know for some $s_0, s_1 \in \mathit{State}$

(4) $s_0 = \mathit{write}(s, I, [\![E]\!](s))$

(5) $[\![C]\!](s_0, s_1)$

(6) $s' = \mathit{write}(s_1, I, \mathit{read}(s, I))$

From the premises, we know

(7) $E \simeq T$

(8) $\$J$ does not occur in P and in T

(9) $\$K$ does not occur in Q

(10) $J \neq K$

(11) $\texttt{POST}(C, P[\$J/I] \texttt{ AND } I = T[\$J/I]) = Q$

From (1b), (11), and the induction hypothesis, we know

(12) Q has no primed variables

(13) $\forall e \in \mathit{Environment}:$
$[\![P[\$J/I] \texttt{ AND } I = T[\$J/I]]\!](e)(s_0,s_0) \wedge [\![C]\!](s_0,s_1)$
$\Rightarrow [\![Q]\!](e)(s_1,s_1)$

To show (b), it suffices by the definition of $[\![\_]\!]$ to show

(c) $\exists v \in \mathit{Value} : [\![Q[\$K/I][I/\$J]]\!](e[K \mapsto v])(s',s')$

To show (c), it suffices to show

(d) $[\![Q[\$K/I][I/\$J]]\!](e[K \mapsto read(s_1, I)])(s', s')$

Let us assume

(e) $[\![P[\$J/I] \texttt{ AND } I = T[\$J/I]]\!](e[J \mapsto read(s, I)])(s_0, s_0)$

From (5), (13), and (e), we know

(14) $[\![Q]\!](e[J \mapsto read(s, I)])(s_1, s_1)$

From (6) and (WS), we know

(15) $s' = s_1 \text{ EXCEPT } I$

From (8), (14), (15), and (PMVF1), we know

(16) $[\![Q[\$K/I]]\!](e[J \mapsto read(s, I)][K \mapsto read(s_1, I)])(s', s_1)$

We define

(17) $Q' := Q[\$K/I][I/\$J]$

From (17), (MPVF0), and (MPVF2), we know

(18) $Q'[\$J/I] = Q[\$K/I]$

From (PMVF0), (RE), and (PMVF1), we know

(19) $[\![Q']\!](e[K \mapsto read(s_1, I)])(s', s') \Leftrightarrow$
$[\![Q'[\$J/I]]\!](e[K \mapsto read(s_1, I)][J \mapsto read(s', I)])(s', s')$

From (18) and (19), we know

(20) $[\![Q[\$K/I][I/\$J]]\!](e[K \mapsto read(s_1, I)])(s', s') \Leftrightarrow$
$[\![Q[\$K/I]]\!](e[K \mapsto read(s_1, I)][J \mapsto read(s', I)])(s', s')$

From (12), (16), and (PVF2), we know

(21) $[\![Q[\$K/I]]\!](e[J \mapsto read(s, I)][K \mapsto read(s_1, I)])(s', s')$

From (6) and (RW1), we know

(22) $read(s', I) = read(s, I)$

From (9), (21), and (22), we know

(23) $[\![Q[\$K/I]]\!](e[K \mapsto read(s_1, I)][J \mapsto read(s', I)])(s', s')$

From (20) and (23), we know (d).

It remains to show (e). By the definition of $[\![\_]\!]$, it suffices to show

(f.1) $[\![P[\$J/I]]\!](e[J \mapsto read(s, I)])(s_0, s_0)$

(f.2) $read(s_0, I) = [\![T[\$J/I]]\!](e[J \mapsto read(s, I)])(s_0, s_0)$

From (4) and (WS), we know

(24) $s_0 = s$ EXCEPT I

From (2), (8), (24), and (PMVF1), we know

(25) $[\![P[\$J/I]]\!](e[J \mapsto read(s, I)])(s_0, s)$

From (1b), (25), and (PVF2), we know (f.1).

From (8), (24) and (PMVT1), we know

(25) $[\![T]\!](e)(s, s_0) = [\![T[\$J/I]]\!](e[J \mapsto read(s, I)])(s_0, s_0)$

From (7) and the definition of $\simeq$, we know

(26) $[\![E]\!](s) = [\![T]\!](e)(s, s_0)$

From (4) and (RW1), we know

(27) $read(s_0, I) = [\![E]\!](s)$

From (25), (26) and (27), we know (f.2). □

4.2.5 Command Sequence

$$\frac{\text{POST}(C_1, P) = Q \quad \text{POST}(C_2, Q) = O}{\text{POST}(C_1 \,;\, C_2, P) = O}$$

This rule shows how we can propagate the computation of postcondition among sequences of commands. Given a precondition P and a sequence $C_1 \,;\, C_2$, we first compute the postcondition O of C_1 w.r.t. P, and then the postcondition Q of C_2 w.r.t. O. For instance, for the program

```
x=x+1; y=y+x
```

with precondition `x=a AND y=b` we derive the first command's postcondition

```
EXISTS $x: $x=a AND y=b AND x=$x+1
```

and then the second command's postcondition

```
EXISTS $x, $y:
  $x=a AND $y=b AND x=$x+1 AND y=$y+x
```

which can be simplified to

```
y=b+a+1
```

Soundness Proof We have to show

(a) $[\![P]\!](e)(s,s) \wedge [\![C_1 ; C_2]\!](s,s') \Rightarrow [\![O]\!](e)(s',s')$

We assume

(2) $[\![P]\!](e)(s,s)$
(3) $[\![C_1 ; C_2]\!](s,s')$

and show

(b) $[\![O]\!](e)(s',s')$

From the premises, we know by the induction hypothesis

(4) $\forall s,s' \in State : [\![P]\!](e)(s,s) \wedge [\![C_1]\!](s,s') \Rightarrow [\![Q]\!](e)(s',s')$
(5) $\forall s,s' \in State : [\![Q]\!](e)(s,s) \wedge [\![C_2]\!](s,s') \Rightarrow [\![O]\!](e)(s',s')$

From (3) and the definition of $[\![\_]\!]$, we know for some $s_0 \in State$

(6) $[\![C_1]\!](s,s_0)$
(7) $[\![C_2]\!](s_0,s')$

From (2), (4), and (6), we know

(8) $[\![Q]\!](e)(s_0,s_0)$

From (5), (7), and (8), we know (b). □

4.2.6 One-Sided Conditional

$$\frac{E \simeq F \qquad \text{POST}(C, P \texttt{ AND } F) = Q}{\text{POST}(\texttt{if (}E\texttt{) } C, P) = Q \texttt{ OR } (P \texttt{ AND !}F)}$$

To compute the postcondition of a one-sided conditional with precondition P, we compute the postcondition of the branch with respect to the conjunction of P and the branch condition; the overall postcondition is the disjunction of this postcondition (corresponding to the possibility that the branch was taken) and the conjunction of P and the negation of the branch condition (corresponding to the possibility that the branch was not taken). For example, for program

```
if (x>0) x=x-a;
```

with precondition `a>0` gives postcondition

```
(EXISTS $x: a>0 AND $x>0 AND x=$x-a) OR
(a>0 AND !(x>0))
```

Soundness Proof We have to show

(a) $[\![P]\!](e)(s,s) \wedge [\![\texttt{if (}E\texttt{) } C]\!](s,s') \Rightarrow [\![Q \texttt{ OR } (P \texttt{ AND !}F)]\!](e)(s',s')$

We assume

(2) $[\![P]\!](e)(s,s)$

(3) $[\![\texttt{if (}E\texttt{) } C]\!](s,s')$

and show

(b) $[\![Q \texttt{ OR } (P \texttt{ AND !}F)]\!](e)(s',s')$

i.e. by the definition of $[\![\_]\!]$

(c) $[\![Q]\!](e)(s',s') \vee ([\![P]\!](e)(s',s') \wedge \neg [\![F]\!](e)(s',s'))$

From the premises and the induction hypothesis, we know

(4) $E \simeq F$

(5) $[\![P \texttt{ AND } F]\!](e)(s,s) \wedge [\![C]\!](s,s') \Rightarrow [\![Q]\!](e)(s',s')$

From (3) and the definition of $[\![\_]\!]$, we know

(6) IF $[\![E]\!](s) =$ TRUE THEN $[\![C]\!](s,s')$ ELSE $s' = s$

We now distinguish two cases:

- **Case $[\![E]\!](s) =$ TRUE:** From the case condition and (6), we know

  (7) $[\![C]\!](s,s')$

  From (5), (7), and the definition of $[\![\_]\!]$, to show (c), it suffices to show

  (d) $[\![P \texttt{ AND } F]\!](e)(s,s)$

  i.e. by the definition of $[\![\_]\!]$,

  (e.1) $[\![P]\!](e)(s,s)$
  (e.2) $[\![F]\!](e)(s,s)$

  From (2) and the definition of $[\![\_]\!]$, we have (e.1). From the case condition, (4), and the definition of $\simeq$, we have (e.2).

- **Case $[\![E]\!](s) \neq$ TRUE:** From the case condition and (6), we know

  (8) $s' = s$

  From (8), to show (c), it suffices to show

  (d.1) $[\![P]\!](e)(s,s)$
  (d.2) $\neg[\![F]\!](e)(s,s)$

  From (2) and the definition of $[\![\_]\!]$, we have (d.1). From the case condition, (4), and the definition of $\simeq$, we have (d.2). □

4.2.7 Two-Sided Conditional

$$\frac{\begin{array}{l} E \simeq F \\ \text{POST}(C_1, P \texttt{ AND } F) = Q_1 \\ \text{POST}(C_2, P \texttt{ AND } \texttt{!}F) = Q_2 \end{array}}{\text{POST}(\texttt{if (}E\texttt{) } C_1 \texttt{ else } C_2, P) = Q_1 \texttt{ OR } Q_2}$$

To compute the postcondition of a two-sided conditional with precondition P, we compute the postcondition of the first branch with respect to the conjunction of P and the branch condition as well as the postcondition of the second branch with respect to the conjunction of P and the negation of the branch condition; the overall postcondition is the disjunction of the two postconditions corresponding to the two possibilities of which branch is taken. For example, for program

```
if (a>0) x=x-a else x=x+a;
```

with precondition `x>0 AND 2|a` gives postcondition

```
(EXISTS $x: $x>0 AND 2|a AND a>0 AND
  x=$x-a) OR
(EXISTS $x: $x>0 AND 2|a AND !(a>0) AND
  x=$x+a)
```

Soundness Proof We have to show

(a) $[\![P]\!](e)(s,s) \wedge [\![\texttt{if (}E\texttt{)}\ C_1\ \texttt{else}\ C_2]\!](s,s') \Rightarrow [\![Q_1\ \texttt{OR}\ Q_2]\!](e)(s',s')$

We assume

(2) $[\![P]\!](e)(s,s)$
(3) $[\![\texttt{if (}E\texttt{)}\ C_1\ \texttt{else}\ C_2]\!](s,s')$

and show

(b) $[\![Q_1\ \texttt{OR}\ Q_2]\!](e)(s',s')$

i.e. by the definition of $[\![\_]\!]$

(c) $[\![Q_1]\!](e)(s',s') \vee [\![Q_2]\!](e)(s',s')$

From the premises and the induction hypothesis, we know

(4) $E \simeq F$
(5) $[\![P\ \texttt{AND}\ F]\!](e)(s,s) \wedge [\![C_1]\!](s,s') \Rightarrow [\![Q_1]\!](e)(s',s')$
(6) $[\![P\ \texttt{AND}\ \texttt{!}F]\!](e)(s,s) \wedge [\![C_2]\!](s,s') \Rightarrow [\![Q_2]\!](e)(s',s')$

From (3) and the definition of $[\![\_]\!]$, we know

(7) $\textsc{if}\ [\![E]\!](s) = \textsc{true}\ \textsc{then}\ [\![C_1]\!](s,s')\ \textsc{else}\ [\![C_2]\!](s,s')$

We now distinguish two cases:

- **Case $[\![E]\!](s) = \textsc{true}$:** From the case condition and (7), we know
  (8) $[\![C_1]\!](s,s')$

From (5), (8), and the definition of $[\![\_]\!]$, to show (c), it suffices to show

(d) $[\![P \texttt{ AND } F]\!](e)(s,s)$

i.e. by the definition of $[\![\_]\!]$,

(e.1) $[\![P]\!](e)(s,s)$
(e.2) $[\![F]\!](e)(s,s)$

From (2) and the definition of $[\![\_]\!]$, we have (e.1). From the case condition, (4), and the definition of $\simeq$, we have (e.2).

- **Case $[\![E]\!](s) \neq$ TRUE:** From the case condition and (7), we know

  (9) $[\![C_2]\!](s,s')$

  From (6), (9), and the definition of $[\![\_]\!]$, to show (c), it suffices to show

  (d) $[\![P \texttt{ AND !} F]\!](e)(s,s)$

  i.e. by the definition of $[\![\_]\!]$,

  (e.1) $[\![P]\!](e)(s,s)$
  (e.2) $\neg[\![F]\!](e)(s,s)$

  From (2) and the definition of $[\![\_]\!]$, we have (e.1). From the case condition, (4), and the definition of $\simeq$, we have (e.2). □

4.2.8 While Loop (Without Invariant)

$E \simeq H$
$C : [F]_{I_1,\ldots,I_n}$
$J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$
$\$J_1,\ldots,\$J_n$ do not occur in P

$\texttt{POST}(\texttt{while (}E\texttt{) } C, P) =$
$\quad !H \texttt{ AND EXISTS } \$J_1,\ldots,\$J_n \texttt{:} \; P[\$J_1/I_1,\ldots,\$J_n/I_n]$

This rule shows that, given a condition P on the prestate of a loop, we know that after the loop the loop condition (the mathematical counterpart of the loop expression) does not hold and that P holds for some values (the prestate values) of the variables that are changed by the loop.

For instance, for program

```
while (x>0) x=x-y;
```

with precondition `y>x AND x>=0` we can derive the postcondition

```
!(x>0) AND EXISTS $x: y>$x AND $x>=0
```

which can be further simplified to

```
x<=0 AND y>0
```

Soundness Proof We have to show

(a) $[\![P]\!](e)(s,s) \wedge [\![\texttt{while (}E\texttt{) } C]\!](s,s') \Rightarrow$
$[\![\, !H \text{ AND EXISTS } \$J_1,\ldots,\$J_n\texttt{: } P[\$J_1/I_1,\ldots,\$J_n/I_n]\,]\!]$
$(e)(s',s')$

We assume

(2) $[\![P]\!](e)(s,s)$

(3) $[\![\texttt{while (}E\texttt{) } C]\!](s,s')$

and show

(b) $[\![\, !H \text{ AND EXISTS } \$J_1,\ldots,\$J_n\texttt{: } P[\$J_1/I_1,\ldots,\$J_n/I_n]\,]\!](e)(s',s')$

From the definition of $[\![\_]\!]$, we have to show

(c.1) $\neg[\![H]\!](e)(s',s')$

(c.2) $\exists v_1,\ldots,v_n \in \mathit{Value}:$
$[\![P[\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e[J_1 \mapsto v_1,\ldots,J_n \mapsto v_n])(s',s')$

From the premises, we know

(4) $E \simeq H$

(5) $C : [F]_{I_1,\ldots,I_n}$

(6) $J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$

(7) $\$J_1,\ldots,\$J_n$ do not occur in P

From (3) and the definition of $[\![\_]\!]$, we know for some $k \in \mathbb{N}, t \in \mathit{State}^\infty$:

(8) $\mathit{finiteExecution}(k,t,s,[\![E]\!],[\![C]\!])$

(9) $[\![E]\!](t(k)) \neq \text{TRUE}$

(10) $t(k) = s'$

From (4), (9), (10), and the definition of $\simeq$, we know (c.1).

From (1a), (5), the definitions of $[\![\_]\!]_\_$ and $[\![\_]\!]$, and the soundness of the verification calculus, we know

(11) $\forall s, s' \in \mathit{State}, e \in \mathit{Environment}:$
$[\![C]\!](s,s') \Rightarrow [\![F]\!](e)(s,s') \wedge s = s' \text{ EXCEPT } I_1, \ldots, I_n$

From (8), (10), (11), the definition of *finiteExecution*, and (TRE), we know

(12) $s = s' \text{ EXCEPT } I_1, \ldots, I_n$

From (2), (6), (7), (12), and (PMVF1), we know

(13) $[\![P[\$J_1/I_1, \ldots, \$J_n/I_n]]\!]$
$(e[J_1 \mapsto \mathit{read}(s, I_1), \ldots, J_n \mapsto \mathit{read}(s, I_n)])(s', s)$

From (1b), (13), and (PVF2), we know (c.2). □

4.2.9 While Loop (With Invariant)

$E \simeq H$
$C : [F]_{I_1,\ldots,I_n}$
$J_1, \ldots, J_n$ is a renaming of $I_1, \ldots, I_n$
$\$J_1, \ldots, \$J_n$ do not occur in P and G
$\mathit{Invariant}(G, H, F)_{I_1,\ldots,I_n}$

POST(`while (`E`)` C, P) =
`!`H `AND`
`EXISTS` $\$J_1, \ldots, \$J_n$`:`
$P[\$J_1/I_1, \ldots, \$J_n/I_n]$ `AND`
$(G[I_1/I_1\texttt{'}, \ldots, I_n/I_n\texttt{'}][\$J_1/I_1, \ldots, \$J_n/I_n]$ =>
$G[\$J_1/I_1, \ldots, \$J_n/I_n, I_1/I_1\texttt{'}, \ldots, I_n/I_n\texttt{'}])$

The rule presented in Subsection 4.2.8 can be strengthened, if we are provided with a loop invariant G. Provided that the precondition P implies G in the initial state, the poststate also satisfies G.

For instance, for program

```
while (x<n) {s=s+x; x=x+1}
```

with precondition `s=0` and invariant `s'=sum(x,x'-1)` (where `sum(`i`,` j`)` denotes the sum of all integers numbers from i to j) we can derive the postcondition

```
!(x<n) AND
EXISTS $s,$x:
  $s=0 AND
  ($s=sum($x,$x-1) => s=sum($x, x-1))
```

which can be simplified to

```
!(x<n) AND EXISTS $x: s=sum($x, x-1)
```

Soundness Proof We have to show

$$
\begin{array}{ll}
 & [\![P]\!](e)(s,s) \wedge [\![\texttt{while(}E\texttt{) } C]\!](s,s') \Rightarrow \\
 & [\![\ \texttt{!}H\ \texttt{AND} \\
 & \quad \texttt{EXISTS } \$J_1,\ldots,\$J_n\texttt{:} \\
\text{(a)} & \quad\quad P[\$J_1/I_1,\ldots,\$J_n/I_n]\ \texttt{AND} \\
 & \quad\quad (G[I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}][\$J_1/I_1,\ldots,\$J_n/I_n] \texttt{=>} \\
 & \quad\quad\quad G[\$J_1/I_1,\ldots,\$J_n/I_n,I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}])\ ]\!](e)(s',s')
\end{array}
$$

We assume

$$
\begin{array}{ll}
\text{(2)} & [\![P]\!](e)(s,s) \\
\text{(3)} & [\![\texttt{while(}E\texttt{) } C]\!](s,s')
\end{array}
$$

and show

$$
\begin{array}{ll}
 & [\![\ \texttt{!}H\ \texttt{AND} \\
 & \quad \texttt{EXISTS } \$J_1,\ldots,\$J_n\texttt{:} \\
\text{(b)} & \quad\quad P[\$J_1/I_1,\ldots,\$J_n/I_n]\ \texttt{AND} \\
 & \quad\quad (G[I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}][\$J_1/I_1,\ldots,\$J_n/I_n] \texttt{=>} \\
 & \quad\quad\quad G[\$J_1/I_1,\ldots,\$J_n/I_n,I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}])\ ]\!](e)(s',s')
\end{array}
$$

From the definition of $[\![\ \_\ ]\!]$, we have to show

$$
\begin{array}{ll}
\text{(c.1)} & \neg[\![H]\!](e)(s',s') \\
 & \exists v_1,\ldots,v_n \in \mathit{Value}\texttt{:} \\
 & \quad [\![P[\$J_1/I_1,\ldots,\$J_n/I_n]\ \texttt{AND} \\
\text{(c.2)} & \quad\quad (G[I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}][\$J_1/I_1,\ldots,\$J_n/I_n] \texttt{=>} \\
 & \quad\quad\quad G[\$J_1/I_1,\ldots,\$J_n/I_n,I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}])\ ]\!] \\
 & \quad\quad (e[J_1 \mapsto v_1,\ldots,J_n \mapsto v_n])(s',s')
\end{array}
$$

From the premises, we know

(4) $E \simeq H$

(5) $C : [F]_{I_1,\ldots,I_n}$

(6) $J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$

(7) $\$J_1,\ldots,\$J_n$ do not occur in P and G

(8) $\mathit{Invariant}(G,H,F)_{I_1,\ldots,I_n}$

From (3) and the definition of $[\![\_]\!]$, we know for some $k \in \mathbb{N}, t \in \mathit{State}^\infty$:

(9) $\mathit{finiteExecution}(k,t,s,[\![E]\!],[\![C]\!])$

(10) $[\![E]\!](t(k)) \neq \text{TRUE}$

(11) $t(k) = s'$

From (4), (10), (11), and the definition of $\simeq$, we know (c.1).

To show (c.2), it suffices, from the definition of $[\![\_]\!]$, to show

(c.2.a.1)
$$[\![P[\$J_1/I_1,\ldots,\$J_n/I_n]]\!] \\ (e[J_1 \mapsto \mathit{read}(s,I_1),\ldots,J_n \mapsto \mathit{read}(s,I_n)])(s',s')$$

(c.2.a.2)
$$[\![G[I_1/I_1{}',\ldots,I_n/I_n{}'][\$J_1/I_1,\ldots,\$J_n/I_n]]\!] \\ (e[J_1 \mapsto \mathit{read}(s,I_1),\ldots,J_n \mapsto \mathit{read}(s,I_n)])(s',s') \Rightarrow \\ [\![G[\$J_1/I_1,\ldots,\$J_n/I_n,I_1/I_1{}',\ldots,I_n/I_n{}']]\!] \\ (e[J_1 \mapsto \mathit{read}(s,I_1),\ldots,J_n \mapsto \mathit{read}(s,I_n)])(s',s')$$

From (1a), (5), the definitions of $[\_]_\_$ and $[\![\_]\!]$, and the soundness of the verification calculus, we know

(12)
$$\forall s,s' \in \mathit{State}, e \in \mathit{Environment}: \\ [\![C]\!](s,s') \Rightarrow [\![F]\!](e)(s,s') \wedge s = s' \text{ EXCEPT } I_1,\ldots,I_n$$

From (9), (11), (12), the definition of *finiteExecution*, and (TRE), we know

(13) $s = s' \text{ EXCEPT } I_1,\ldots,I_n$

From (2), (6), (7), (13), and (PMVF1), we know

(14)
$$[\![P[\$J_1/I_1,\ldots,\$J_n/I_n]]\!] \\ (e[J_1 \mapsto \mathit{read}(s,I_1),\ldots,J_n \mapsto \mathit{read}(s,I_n)])(s',s)$$

From (1b), (14), and (PVF2), we know (c.2.a.1).

To show (c.2.a.2), we assume

(15) $[\![G[I_1/I_1\text{'},\ldots,I_n/I_n\text{'}][\$J_1/I_1,\ldots,\$J_n/I_n]]\!]$
$(e[J_1 \mapsto read(s,I_1),\ldots,J_n \mapsto read(s,I_n)])(s',s')$

and show

(c.2.a.2.a) $[\![G[\$J_1/I_1,\ldots,\$J_n/I_n,I_1/I_1\text{'},\ldots,I_n/I_n\text{'}]]\!]$
$(e[J_1 \mapsto read(s,I_1),\ldots,J_n \mapsto read(s,I_n)])(s',s')$

From (6), (7), (13), (15), and (PMVF1), we know

(17) $[\![G[I_1/I_1\text{'},\ldots,I_n/I_n\text{'}]]\!](e)(s,s)$

From (4), (5), (8), we can from the verification calculus derive

(18) `while (E) C` : $[\,!H[I_1\text{'}/I_1,\ldots,I_n\text{'}/I_n]$ `AND`
$(G[I_1/I_1\text{'},\ldots,I_n/I_n\text{'}]$ `=>` $G)\,]_{I_1,\ldots,I_n}$

From (18) and the soundness of the verification calculus, we thus know

(19) $[\![$ `while (E) C` $]\!](s,s') \Rightarrow [\![\,[\,!H[I_1\text{'}/I_1,\ldots,I_n\text{'}/I_n]$ `AND`
$(G[I_1/I_1\text{'},\ldots,I_n/I_n\text{'}]$ `=>` $G)\,]_{I_1,\ldots,I_n}]\!](e)(s,s')$

From (3), (19), and the definition of $[\![\_]\!]$, we know

(20) $\neg[\![H[I_1\text{'}/I_1,\ldots,I_n\text{'}/I_n]]\!](e)(s,s')$

(21) $[\![G[I_1/I_1\text{'},\ldots,I_n/I_n\text{'}]]\!](e)(s,s') \Rightarrow [\![G]\!](e)(s,s')$

(22) $s = s'$ EXCEPT $I_1,\ldots,I_n$

From (PPVF0), we know

(23) $I_1\text{'},\ldots,I_n\text{'}$ do not occur in $G[I_1/I_1\text{'},\ldots,I_n/I_n\text{'}]$

From (17), (22), (23) and (PVF4), we know

(24) $[\![G[I_1/I_1\text{'},\ldots,I_n/I_n\text{'}]]\!](e)(s,s')$

From (21) and (24), we know

(25) $[\![G]\!](e)(s,s')$

From (6), (7), (22), (25), and (PMVF1), we know

(26) $[\![G[\$J_1/I_1,\ldots,\$J_n/I_n]]\!]$
$(e[J_1 \mapsto read(s,I_1),\ldots,J_n \mapsto read(s,I_n)])(s',s')$

From (26) and (IDE), we know

(27) $[\![G[\$J_1/I_1,\ldots,\$J_n/I_n]]\!]$
$(e[J_1 \mapsto read(s,I_1),\ldots,J_n \mapsto read(s,I_n)])$
$(s', writes(s',I_1,read(s',I_1),\ldots,I_n,read(s',I_n)))$

From (27) and (PPVF2), we know (c.2.a.2.a). □

Command Language: Assertions

$C \in$ Command
$F \in$ Formula

$C ::= \ldots \mid \texttt{assert}\ F$

$[\![\texttt{assert}\ F]\!](s,s') \Leftrightarrow [\![F]\!](s) \wedge s' = s$

Figure 4.7: The Command Language Extended by Assertions

4.3 Computing Assertions

The goal of this section is to make for given command C and precondition P explicit which condition holds on the prestate of every subcommand of C. For this purpose, we introduce in Figure 4.7 a new command `assert` F where F is a formula called an *assertion*. If the prestate of this command satisfies the assertion, then the poststate equals the prestate (i.e. the command has no effect); otherwise, there is no poststate (i.e. the command "blocks"). Placing an assertion into the "control flow" of a program thus prevents all state sequences passing through states that violate the assertion.

Based on this idea, Figures 4.8 and 4.9 describe a calculus for deriving judgements of the form $\textsc{trans}(C,P) = C'$. The informal interpretation of this judgement is that C' is a duplicate of C where each (sub)command is preceded by an assertion in such a way that the semantics of C preserved. For example, from program

```
x = x+1;
if (x=0)
  y=y+1
else
  z=z+1
```

with precondition `x=a` the calculus constructs the program

```
assert x=a;
x = x+1;
assert x=a+1;
if (x=0)
   assert x=a+1 AND x=0; y=y+1
else
  assert x=a+1 AND x!=0; z=z+1
```

Assertion Calculus: Judgements

$\text{TRANS}(C,P) = C' \Leftrightarrow$
P has no primed program variables $\Rightarrow$
$\forall s,s' \in State : [\![P]\!](s) \wedge [\![C]\!](s,s') \Leftrightarrow [\![C']\!](s,s')$

Assertion Calculus: Rules for Non-Loops

$\text{TRANS}(I{=}E,P) = \texttt{assert}\ P\texttt{;}\ I{=}E$

$\$J$ does not occur in P
$\text{TRANS}(C, \texttt{EXISTS}\ \$J : P[\$J/I]) = C'$
$\text{POST}(\texttt{var}\ I\texttt{;}\ C,P) = \texttt{assert}\ P\texttt{;}\ \texttt{var}\ I\texttt{;}\ C'$

$E \simeq T$
$\$J$ does not occur in P and in T
$\text{TRANS}(C, \texttt{EXISTS}\ \$J : P[\$J/I]\ \texttt{AND}\ I = T[\$J/I]) = C'$
$\text{TRANS}(\texttt{var}\ I{=}E\texttt{;}\ C,P) = \texttt{assert}\ P\texttt{;}\ \texttt{var}\ I{=}E\texttt{;}\ C'$

$\text{POST}(C_1,P) = Q$
$\text{TRANS}(C_1,P) = C_1'$
$\text{TRANS}(C_2,Q) = C_2'$
$\text{TRANS}(C_1\texttt{;}\ C_2,P) = \texttt{assert}\ P\texttt{;}\ C_1'\texttt{;}\ C_2'$

$E \simeq F$
$\text{TRANS}(C, P\ \texttt{AND}\ F) = C'$
$\text{TRANS}(\texttt{if (}E\texttt{)}\ C,P) = \texttt{assert}\ P\texttt{;}\ \texttt{if (}E\texttt{)}\ C'$

$E \simeq F$
$\text{TRANS}(C_1, P\ \texttt{AND}\ F) = C_1'$
$\text{TRANS}(C_2, P\ \texttt{AND}\ \texttt{!}F) = C_2'$
$\text{TRANS}(\texttt{if (}E\texttt{)}\ C_1\ \texttt{else}\ C_2,P) =$
$\quad \texttt{assert}\ P\texttt{;}\ \texttt{if (}E\texttt{)}\ C_1'\ \texttt{else}\ C_2'$

Figure 4.8: The Assertion Calculus of the Command Language (Part 1/2)

Assertion Calculus: Rules for Loops

$$\frac{\begin{array}{l} E \simeq H \\ C : [F]_{I_1,\ldots,I_n} \\ J_1,\ldots,J_n \text{ is a renaming of } I_1,\ldots,I_n \\ \$J_1,\ldots,\$J_n \text{ do not occur in } P \\ \textsc{trans}(C, \\ \quad H \texttt{ AND EXISTS } \$J_1,\ldots,\$J_n : P[\$J_1/I_1,\ldots,\$J_n/I_n]) = C' \end{array}}{\textsc{trans}(\texttt{while(}E\texttt{) } C, P) = \texttt{assert } P\texttt{; while(}E\texttt{) } C'}$$

$$\frac{\begin{array}{l} E \simeq H \\ C : [F]_{I_1,\ldots,I_n} \\ J_1,\ldots,J_n \text{ is a renaming of } I_1,\ldots,I_n \\ \$J_1,\ldots,\$J_n \text{ do not occur in } P \text{ and } G \\ \mathit{Invariant}(G,H,F)_{I_1,\ldots,I_n} \\ \textsc{trans}(C, \\ \quad H \texttt{ AND} \\ \quad \texttt{EXISTS } \$J_1,\ldots,\$J_n : \\ \qquad P[\$J_1/I_1,\ldots,\$J_n/I_n] \texttt{ AND} \\ \qquad (G[I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}][\$J_1/I_1,\ldots,\$J_n/I_n] \texttt{ =>} \\ \qquad\quad G[\$J_1/I_1,\ldots,\$J_n/I_n,I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}])) = C' \end{array}}{\textsc{trans}(\texttt{while(}E\texttt{) } C, P) = \texttt{assert } P\texttt{; while(}E\texttt{) } C'}$$

Figure 4.9: The Assertion Calculus of the Command Language (Part 2/2)

The rules of the calculus are constructed in analogy to the rules for computing postconditions by propagating the knowledge about a prestate from a command to its subcommands (however, the rule for command sequences requires the explicit computation of the postcondition of the first command). Because of the rules for loops, the conditions described by the assertions are not necessarily the strongest ones (however, for programs without loops they actually are).

The following theorem states the soundness claim of this calculus.

Theorem (Soundness of the Assertion Calculus) Assume the condition denoted by *DifferentVariables*. If $\text{TRANS}(C,P) = C'$ can be derived from the rules of the assertion calculus of the command language, then it is true that

$$P \text{ has no primed program variables} \Rightarrow \\ \forall s,s' \in \mathit{State} : [\![P]\!](s) \wedge [\![C]\!](s,s') \Leftrightarrow [\![C']\!](s,s')$$

Assume

(1a) *DifferentVariables*

Take C, P, and C' such that $\text{TRANS}(C,P) = C'$ can be derived and assume

(1b) P has no primed program variables

Now take arbitrary $s,s' \in \mathit{State}$. We prove

$$[\![P]\!](s) \wedge [\![C]\!](s,s') \Leftrightarrow [\![C']\!](s,s')$$

The direction "right to left" of this proof is simple because by the rules of the calculus C' differs from C only by the introduction of additional assertions whose only effect (if any) is to prevent successor states s'.

The direction "left to right" of the proof is shown by induction on the derivation of $\text{TRANS}(C,P) = C'$. The following subsections cover all cases for the last step of such a derivation.

From (1b) and the rules, we can immediately deduce that in every derivation $\text{TRANS}(C',P') = C'$ that matches the premise of a rule which has conclusion $\text{TRANS}(C,P) = C$, the formula P' has no primed variables; we thus assume in the proofs that the induction hypothesis immediately implies the core claim $\forall s,s' \in \mathit{State} : [\![P']\!](s) \wedge [\![C']\!](s,s') \Rightarrow [\![C']\!](s',s')$. $\square$

4.3.1 Assignment

$$\text{TRANS}(I{=}E, P) = \texttt{assert}\ P\texttt{;}\ I{=}E$$

Soundness Proof We have to prove

(a) $[\![P]\!](s) \wedge [\![I{=}E]\!](s,s') \Rightarrow [\![\texttt{assert}\ P\texttt{;}\ I{=}E]\!](s,s')$

We assume

(2) $[\![P]\!](s)$
(3) $[\![I{=}E]\!](s,s')$

By the definition of $[\![\_]\!]$, it suffices to show

(b.1) $[\![P]\!](s)$
(b.2) $[\![I{=}E]\!](s,s')$

From (2), we know (b.1). From (3), we know (b.2). □

4.3.2 Variable Declaration

$$\frac{\$J \text{ does not occur in } P \qquad \text{TRANS}(C, \texttt{EXISTS}\ \$J : P[\$J/I]) = C'}{\text{POST}(\texttt{var}\ I\texttt{;}\ C, P) = \texttt{assert}\ P\texttt{;}\ \texttt{var}\ I\texttt{;}\ C'}$$

Soundness Proof We have to prove

(a) $[\![P]\!](s) \wedge [\![\texttt{var}\ I\texttt{;}\ C]\!](s,s') \Rightarrow [\![\texttt{assert}\ P\texttt{;}\ \texttt{var}\ I\texttt{;}\ C']\!](s,s')$

We assume

(2) $[\![P]\!](s)$
(3) $[\![\texttt{var}\ I\texttt{;}\ C]\!](s,s')$

By the definition of $[\![\_]\!]$, it suffices to show

(b.1) $[\![P]\!](s)$

(b.2) $\exists s_0, s_1 \in \mathit{State}:$
$s_0 = s\ \text{EXCEPT}\ I \wedge [\![C']\!](s_0,s_1) \wedge\ s' = \mathit{write}(s_1, I, \mathit{read}(s,I))$

From the premises and the induction hypothesis, we know

(4) $\$J$ does not occur in P

(5) $\forall s, s' \in \mathit{State} : [\![\texttt{EXISTS } \$J : P[\$J/I]]\!](s) \wedge [\![C]\!](s, s') \Rightarrow [\![C']\!](s, s')$

From (2), we know (b.1).

From (3) and the definition of $[\![\_]\!]$, we know for some $s_0, s_1 \in \mathit{State}$

(6) $s_0 = s$ EXCEPT I

(7) $[\![C]\!](s_0, s_1)$

(8) $s' = \mathit{write}(s_1, I, \mathit{read}(s, I))$

Let us assume

(b.2.a) $[\![C']\!](s_0, s_1)$

From (6), (8), and (b.2.a), we know (b.2).

To show (b.2.a), by (5) and (7), it suffices to show

(b.2.b) $[\![\texttt{EXISTS } \$J : P[\$J/I]]\!](s_0)$

From the definition of $[\![\_]\!]$, it suffices to show for arbitrary $e \in \mathit{Environment}$

(b.2.c) $\exists v \in \mathit{Value} : [\![P[\$J/I]]\!](e[J \mapsto v])(s_0, s_0)$

From (2) and the definition of $[\![\_]\!]$, we know

(9) $[\![P]\!](e)(s, s)$

From (4), (6), and (PMVF1), we know

(10) $[\![P[\$J/I]]\!](e[J \mapsto \mathit{read}(s, I)])(s_0, s)$

From (1b), (10), and (PVF2), we know (b.2.c). □

4.3.3 Variable Definition

$$\frac{\begin{array}{l} E \simeq T \\ \$J \text{ does not occur in } P \text{ and in } T \\ \textsc{trans}(C, \texttt{EXISTS } \$J : P[\$J/I] \texttt{ AND } I = T[\$J/I]) = C' \end{array}}{\textsc{trans}(\texttt{var } I\texttt{=}E\texttt{;}\, C, P) = \texttt{assert } P\texttt{; var } I\texttt{=}E\texttt{;}\, C'}$$

Soundness Proof We have to prove

(a) $[\![P]\!](s) \wedge [\![\texttt{var } I\texttt{=}E\texttt{;}\ C]\!](s,s') \Rightarrow [\![\texttt{assert } P\texttt{; var } I\texttt{=}E\texttt{;}\ C']\!](s,s')$

We assume

(2) $[\![P]\!](s)$

(3) $[\![\texttt{var } I\texttt{=}E\texttt{;}\ C]\!](s,s')$

By the definition of $[\![\_]\!]$, it suffices to show

(b.1) $[\![P]\!](s)$

(b.2) $\exists s_0, s_1 \in \mathit{State}:$
$s_0 = \mathit{write}(s, I, [\![E]\!](s)) \wedge [\![C']\!](s_0, s_1) \wedge\ s' = \mathit{write}(s_1, I, \mathit{read}(I, s))$

From the premises and the induction hypothesis, we know

(4) $E \simeq T$

(5) $\$J$ does not occur in P and in T

(6) $\forall s, s' \in \mathit{State}:$
$[\![\texttt{EXISTS } \$J : P[\$J/I] \texttt{ AND } I = T[\$J/I]]\!](s) \wedge [\![C]\!](s,s') \Rightarrow$
$[\![C']\!](s,s')$

From (2), we know (b.1).

From (3) and the definition of $[\![\_]\!]$, we know for some $s_0, s_1 \in \mathit{State}$

(7) $s_0 = \mathit{write}(s, I, [\![E]\!](s))$

(8) $[\![C]\!](s_0, s_1)$

(9) $s' = \mathit{write}(s_1, I, \mathit{read}(I, s))$

Let us assume

(b.2.a) $[\![C']\!](s_0, s_1)$

From (7), (9), and (b.2.a), we know (b.2).

To show (b.2.a), by (6) and (8), it suffices to show

(b.2.b) $[\![\texttt{EXISTS } \$J : P[\$J/I] \texttt{ AND } I = T[\$J/I]]\!](s_0)$

From the definition of $[\![\_]\!]$, it suffices to show for arbitrary $e \in \mathit{Environment}$

(b.2.c) $\exists v \in \mathit{Value}$:
$[\![P[\$J/I]]\!](e[J \mapsto v])(s_0, s_0) \wedge$
$\mathit{read}(s_0, I) = [\![T[\$J/I]]\!](e[J \mapsto v])(s_0, s_0)$

From (2) and the definition of $[\![\_]\!]$, we know

(10) $[\![P]\!](e)(s, s)$

From (1a), (7), and (WS), we know

(11) $s_0 = s$ EXCEPT I

From (5), (11), and (PMVF1), we know

(12) $[\![P[\$J/I]]\!](e[J \mapsto \mathit{read}(s, I)])(s_0, s)$

From (1b), (12), and (PVF2), we know

(13) $[\![P[\$J/I]]\!](e[J \mapsto \mathit{read}(s, I)])(s_0, s_0)$

From (6), (11), and (PMVT1), we know

(14) $[\![T[\$J/I]]\!](e[J \mapsto \mathit{read}(s, I)])(s_0, s_0) = [\![T]\!](e)(s, s_0)$

From (7) and (RW1), we know

(15) $\mathit{read}(s_0, I) = [\![E]\!](s)$

From (4), (15), and the definition of $\simeq$, we know

(16) T has no primed program variables
(17) $\mathit{read}(s_0, I) = [\![T]\!](e)(s, s)$

From (16), (17), and (PVT2), we know

(18) $\mathit{read}(s_0, I) = [\![T]\!](e)(s, s_0)$

From (13), (14), and (18), we know (b.2.c). □

4.3.4 Command Sequence

$$\frac{\begin{array}{l}\text{POST}(C_1, P) = Q \\ \text{TRANS}(C_1, P) = C_1' \\ \text{TRANS}(C_2, Q) = C_2'\end{array}}{\text{TRANS}(C_1 \texttt{;}\ C_2, P) = \texttt{assert}\ P \texttt{;}\ C_1' \texttt{;}\ C_2'}$$

Soundness Proof We have to prove

(a) $[\![P]\!](s) \wedge [\![C_1 ; C_2]\!](s,s') \Rightarrow [\![\texttt{assert}\ P;\ C_1' ; C_2']\!](s,s')$

We assume

(2) $[\![P]\!](s)$

(3) $[\![C_1 ; C_2]\!](s,s')$

By the definition of $[\![\_]\!]$, it suffices to show

(b.1) $[\![P]\!](s)$

(b.2) $\exists s_0 \in \mathit{State} : [\![C_1']\!](s,s_0) \wedge [\![C_2']\!](s,s_0)$

From the premises and the induction hypothesis, we know

(4) $\textsc{post}(C_1,P) = Q$

(5) $\forall s,s' \in \mathit{State} : [\![P]\!](s) \wedge [\![C_1]\!](s,s') \Rightarrow [\![C_1']\!](s,s')$

(6) $\forall s,s' \in \mathit{State} : [\![Q]\!](s) \wedge [\![C_2]\!](s,s') \Rightarrow [\![C_2']\!](s,s')$

From (2), we know (b.1).

From (3) and the definition of $[\![\_]\!]$, we know for some $s_0 \in \mathit{State}$

(7) $[\![C_1]\!](s,s_0)$

(8) $[\![C_2]\!](s_0,s')$

To show (b.2), it suffices to show

(b.2.a.1) $[\![C_1']\!](s,s_0)$

(b.2.a.2) $[\![C_2']\!](s_0,s')$

From (2), (5), and (7), we know (b.2.a.1).

Let us assume

(e) $[\![Q]\!](s_0)$

From (e), (6), and (8), we know (b.2.a.2).

It remains to show (e), i.e. for arbitrary $e \in \mathit{Environment}$

(f) $[\![Q]\!](e)(s_0,s_0)$

From (1b), (4), and the soundness of the postcondition calculus, we know

(9) $\forall e \in \mathit{Environment}, s,s' \in \mathit{State} :$
$[\![P]\!](e)(s,s) \wedge [\![C_1]\!](s,s') \Rightarrow [\![Q]\!](e)(s',s')$

From (2), (7), and (9), we know (f). □

4.3.5 One-Sided Conditional

$$\frac{E \simeq F \qquad \text{TRANS}(C, P \texttt{ AND } F) = C'}{\text{TRANS}(\texttt{if (}E\texttt{) }C, P) = \texttt{assert }P\texttt{; if (}E\texttt{) }C'}$$

Soundness Proof We have to prove

(a) $[\![P]\!](s) \wedge [\![\texttt{if (}E\texttt{) }C]\!](s,s') \Rightarrow [\![\texttt{assert }P\texttt{; if (}E\texttt{) }C']\!](s,s')$

We assume

(2) $[\![P]\!](s)$
(3) $[\![\texttt{if (}E\texttt{) }C]\!](s,s')$

By the definition of $[\![\_]\!]$, it suffices to show

(b.1) $[\![P]\!](s)$
(b.2) IF $[\![E]\!](s) =$ TRUE THEN $[\![C']\!](s,s')$ ELSE $s' = s$

From the premises and the induction hypothesis, we know

(4) $E \simeq F$
(5) $[\![P \texttt{ AND } F]\!](s) \wedge [\![C]\!](s,s') \Rightarrow [\![C']\!](s,s')$

From (3) and the definition of $[\![\_]\!]$, we know

(6) IF $[\![E]\!](s) =$ TRUE THEN $[\![C]\!](s,s')$ ELSE $s' = s$

From (2) we know (b.1). To show (b.2), we perform a case distinction.

- Case $[\![E]\!](s) =$ TRUE:

  From the case condition, (4), and the definition of $\simeq$, we know

  (7) $[\![F]\!](s)$

  From the case condition and (6), we know

  (8) $[\![C]\!](s,s')$

  From (2), (5), (7), (8), and the definition of $[\![\_]\!]$, we know

  (9) $[\![C']\!](s,s')$

From the case condition and (9), we know (b.2).

- Case $[\![E]\!](s) \neq \text{TRUE}$:

  From the case condition and (6), we know

  (10) $s' = s$

  From the case condition and (10), we know (b.2). □

4.3.6 Two-Sided Conditional

$$\frac{\begin{array}{l} E \simeq F \\ \text{TRANS}(C_1, P \texttt{ AND } F) = C_1' \\ \text{TRANS}(C_2, P \texttt{ AND !} F) = C_2' \end{array}}{\begin{array}{l} \text{TRANS}(\texttt{if (}E\texttt{) } C_1 \texttt{ else } C_2, P) = \\ \quad \texttt{assert } P\texttt{; if (}E\texttt{) } C_1' \texttt{ else } C_2' \end{array}}$$

Soundness Proof We have to prove

(a) $[\![P]\!](s) \wedge [\![\texttt{if (}E\texttt{) } C_1 \texttt{ else } C_2]\!](s,s') \Rightarrow$
$[\![\texttt{assert } P\texttt{; if (}E\texttt{) } C_1' \texttt{ else } C_2']\!](s,s')$

We assume

(2) $[\![P]\!](s)$

(3) $[\![\texttt{if (}E\texttt{) } C_1 \texttt{ else } C_2]\!](s,s')$

By the definition of $[\![\_]\!]$, it suffices to show

(b.1) $[\![P]\!](s)$

(b.2) IF $[\![E]\!](s) = \text{TRUE}$ THEN $[\![C_1']\!](s,s')$ ELSE $[\![C_2']\!](s,s')$

From the premises and the induction hypothesis, we know

(4) $E \simeq F$

(5) $[\![P \texttt{ AND } F]\!](s) \wedge [\![C_1]\!](s,s') \Rightarrow [\![C_1']\!](s,s')$

(6) $[\![P \texttt{ AND !} F]\!](s) \wedge [\![C_2]\!](s,s') \Rightarrow [\![C_2']\!](s,s')$

From (3) and the definition of $[\![\_]\!]$, we know

(7) IF $[\![E]\!](s) = \text{TRUE}$ THEN $[\![C_1]\!](s,s')$ ELSE $[\![C_2]\!](s,s')$

From (2) we know (b.1). To show (b.2), we perform a case distinction.

- Case $[\![E]\!](s) = \text{TRUE}$:

  From the case condition, (4), and the definition of $\simeq$, we know

  (8) $[\![F]\!](s)$

  From the case condition and (7), we know

  (9) $[\![C_1]\!](s,s')$

  From (2), (5), (8), (9), and the definition of $[\![\_]\!]$, we know

  (10) $[\![C_1']\!](s,s')$

  From the case condition and (10), we know (b.2).

- Case $[\![E]\!](s) \neq \text{TRUE}$:

  From the case condition, (4), and the definition of $\simeq$, we know

  (11) $\neg[\![F]\!](s)$

  From the case condition and (7), we know

  (12) $[\![C_2]\!](s,s')$

  From (2), (6), (11), (12), and the definition of $[\![\_]\!]$, we know

  (13) $[\![C_2']\!](s,s')$

  From the case condition and (13), we know (b.2). □

4.3.7 While Loop (Without Invariant)

$$\frac{\begin{array}{l} E \simeq H \\ C : [F]_{I_1,\ldots,I_n} \\ J_1,\ldots,J_n \text{ is a renaming of } I_1,\ldots,I_n \\ \$J_1,\ldots,\$J_n \text{ do not occur in } P \\ \text{TRANS}(C, \\ \quad H \texttt{ AND EXISTS } \$J_1,\ldots,\$J_n : P[\$J_1/I_1,\ldots,\$J_n/I_n]) = C' \end{array}}{\text{TRANS}(\texttt{while (}E\texttt{) } C, P) = \texttt{assert } P\texttt{; while (}E\texttt{) } C'}$$

Soundness Proof We have to prove

$$\text{(a)}\quad \begin{array}{l}[\![P]\!](s) \wedge [\![\texttt{while(}E\texttt{) } C]\!](s,s') \Rightarrow \\ \quad [\![\texttt{assert } P\texttt{; while(}E\texttt{) } C']\!](s,s')\end{array}$$

We assume

$$\text{(2)}\quad [\![P]\!](s)$$
$$\text{(3)}\quad [\![\texttt{while(}E\texttt{) } C]\!](s,s')$$

By the definition of $[\![\_]\!]$, it suffices to show

$$\text{(b.1)}\quad [\![P]\!](s)$$
$$\text{(b.2)}\quad \begin{array}{l}\exists k \in \mathbb{N}, t \in \mathit{State}^\infty : \\ \quad \mathit{finiteExecution}(k,t,s,[\![E]\!],[\![C']\!]) \wedge \\ \quad [\![E]\!](t(k)) \neq \text{TRUE} \wedge t(k) = s'\end{array}$$

From the premises and the induction hypothesis, we know

$$\text{(4)}\quad E \simeq H$$
$$\text{(5)}\quad C : [F]_{I_1,\ldots,I_n}$$
$$\text{(6)}\quad J_1,\ldots,J_n \text{ is a renaming of } I_1,\ldots,I_n$$
$$\text{(7)}\quad \$J_1,\ldots,\$J_n \text{ do not occur in } P$$
$$\text{(8)}\quad \begin{array}{l}\forall s,s' \in \mathit{State} : \\ \quad [\![H \texttt{ AND EXISTS } \$J_1,\ldots,\$J_n : P[\$J_1/I_1,\ldots,\$J_n/I_n]]\!](s) \wedge \\ \quad [\![C]\!](s,s') \Rightarrow \\ \quad\quad [\![C']\!](s,s')\end{array}$$

From (2), we know (b.1).

From (3) and the definition of $[\![\_]\!]$, we know for some $k \in \mathbb{N}, t \in \mathit{State}^\infty$

$$\text{(9)}\quad \mathit{finiteExecution}(k,t,s,[\![E]\!],[\![C]\!])$$
$$\text{(10)}\quad [\![E]\!](t(k)) \neq \text{TRUE}$$
$$\text{(11)}\quad t(k) = s'$$

To show (b.2), from (10) and (11), it suffices to show

$$\text{(b.2.a)}\quad \mathit{finiteExecution}(k,t,s,[\![E]\!],[\![C']\!])$$

i.e., by the definition of *finiteExecution*,

(b.2.b.1) $t(0) = s$

(b.2.b.2) $\forall i \in \mathbb{N}_k : [\![E]\!](t(i)) = \text{TRUE} \wedge [\![C']\!](t(i), t(i+1))$

From (9) and the definition of *finiteExecution*, we know

(12) $t(0) = s$

(13) $\forall i \in \mathbb{N}_k : [\![E]\!](t(i)) = \text{TRUE} \wedge [\![C]\!](t(i), t(i+1))$

From (12), we know (b.2.b.1). To show (b.2.b.2), we show for arbitrary $i \in \mathbb{N}_k$

(b.2.b.2.a.1) $[\![E]\!](t(i)) = \text{TRUE}$

(b.2.b.2.a.2) $[\![C']\!](t(i), t(i+1))$

From (13), we know (b.2.b.1.a.1) and

(14) $[\![C]\!](t(i), t(i+1))$

To show (b.2.b.1.a.2), from (8) and (14), it suffices to show

(b.2.b.1.a.1.a) $[\![H \text{ AND EXISTS } \$J_1, \ldots, \$J_n : P[\$J_1/I_1, \ldots, \$J_n/I_n]]\!](t(i))$

i.e. by the definition of $[\![\_]\!]$, for arbitrary $e \in \mathit{Environment}$,

(b.2.b.1.a.1.b.1) $[\![H]\!](e)(t(i), t(i))$

(b.2.b.1.a.1.b.2) $\exists v_1, \ldots, v_n \in \mathit{Value} :$
$[\![P[\$J_1/I_1, \ldots, \$J_n/I_n]]\!](e[J_1 \mapsto v_1, \ldots, J_n \mapsto v_n])(t(i), t(i))$

From (4), (13), and the definition of $\simeq$, we know (b.2.b.1.a.1.b.1).

From (5) and the soundness of the verification calculus, we know

(15) $\forall s, s' \in \mathit{Store} : [\![C]\!](s, s') \Rightarrow s = s' \text{ EXCEPT } I_1, \ldots, I_n$

From (13) and (15), we know

(16) $\forall i \in \mathbb{N}_k : t(i) = t(i+1) \text{ EXCEPT } I_1, \ldots, I_n$

From (12), (16), and (TRE), we know

(17) $s = t(i) \text{ EXCEPT } I_1, \ldots, I_n$

From (2), (6), (7), (17), and (PMVF1), we know

(18) $[\![P[\$J_1/I_1, \ldots, \$J_n/I_n]]\!]$
$(e[J_1 \mapsto \mathit{read}(s, I_1), \ldots, J_n \mapsto \mathit{read}(s, I_n)])(t(i), s)$

From (1b), (18), and (PVF2), we know

(19) $[\![P[\$J_1/I_1, \ldots, \$J_n/I_n]]\!]$
$(e[J_1 \mapsto \mathit{read}(s, I_1), \ldots, J_n \mapsto \mathit{read}(s, I_n)])(t(i), t(i))$

From (19), we know (b.2.b.1.a.1.b.2). □

4.3.8 While Loop (With Invariant)

$$\frac{\begin{array}{l} E \simeq H \\ C : [F]_{I_1,\ldots,I_n} \\ J_1,\ldots,J_n \text{ is a renaming of } I_1,\ldots,I_n \\ \$J_1,\ldots,\$J_n \text{ do not occur in } P \text{ and } G \\ \mathit{Invariant}(G,H,F)_{I_1,\ldots,I_n} \\ \textsc{trans}(C, \\ \quad H \texttt{ AND} \\ \quad \texttt{EXISTS } \$J_1,\ldots,\$J_n\texttt{:} \\ \quad\quad P[\$J_1/I_1,\ldots,\$J_n/I_n] \texttt{ AND} \\ \quad\quad (G[I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}][\$J_1/I_1,\ldots,\$J_n/I_n] \texttt{ =>} \\ \quad\quad\quad G[\$J_1/I_1,\ldots,\$J_n/I_n,I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}])) = C' \end{array}}{\textsc{trans}(\texttt{while (}E\texttt{) } C, P) = \texttt{assert } P\texttt{; while (}E\texttt{) } C'}$$

Soundness Proof We have to prove

(a) $[\![P]\!](s) \wedge [\![\texttt{while (}E\texttt{) } C]\!](s,s') \Rightarrow [\![\texttt{assert } P\texttt{; while (}E\texttt{) } C']\!](s,s')$

We assume

(2) $[\![P]\!](s)$

(3) $[\![\texttt{while (}E\texttt{) } C]\!](s,s')$

By the definition of $[\![\_]\!]$, it suffices to show

(b.1) $[\![P]\!](s)$

(b.2) $\exists k \in \mathbb{N}, t \in \mathit{State}^\infty :$
$\quad \mathit{finiteExecution}(k,t,s,[\![E]\!],[\![C']\!]) \wedge$
$\quad [\![E]\!](t(k)) \neq \textsc{true} \wedge t(k) = s'$

From the premises and the induction hypothesis, we know

(4) $E \simeq H$

(5) $C : [F]_{I_1,\ldots,I_n}$

(6) $J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$

(7) $\$J_1,\ldots,\$J_n$ do not occur in P and G

(8) $\mathit{Invariant}(G,H,F)_{I_1,\ldots,I_n}$

$$
\begin{array}{ll}
 & \forall s, s' \in \mathit{State} : \\
 & \quad [\![H \ \texttt{AND} \\
 & \quad \texttt{EXISTS} \ \$J_1, \ldots, \$J_n : \\
(9) & \qquad P[\$J_1/I_1, \ldots, \$J_n/I_n] \ \texttt{AND} \\
 & \qquad (G[I_1/I_1\texttt{'}, \ldots, I_n/I_n\texttt{'}][\$J_1/I_1, \ldots, \$J_n/I_n] \texttt{=>} \\
 & \qquad\quad G[\$J_1/I_1, \ldots, \$J_n/I_n, I_1/I_1\texttt{'}, \ldots, I_n/I_n\texttt{'}])[\![(s) \ \wedge [\![C]\!](s, s') \\
 & \quad \Rightarrow [\![C']\!](s, s')
\end{array}
$$

From (2), we know (b.1).

From (3) and the definition of $[\![\_]\!]$, we know for some $k \in \mathbb{N}, t \in \mathit{State}^\infty$

(10) $\mathit{finiteExecution}(k, t, s, [\![E]\!], [\![C]\!])$

(11) $[\![E]\!](t(k)) \neq \text{TRUE}$

(12) $t(k) = s'$

To show (b.2), from (11) and (12), it suffices to show

(b.2.a) $\mathit{finiteExecution}(k, t, s, [\![E]\!], [\![C']\!])$

i.e., by the definition of *finiteExecution*,

(b.2.b.1) $t(0) = s$

(b.2.b.2) $\forall i \in \mathbb{N}_k : [\![E]\!](t(i)) = \text{TRUE} \wedge [\![C']\!](t(i), t(i+1))$

From (10) and the definition of *finiteExecution*, we know

(13) $t(0) = s$

(14) $\forall i \in \mathbb{N}_k : [\![E]\!](t(i)) = \text{TRUE} \wedge [\![C]\!](t(i), t(i+1))$

From (13), we know (b.2.b.1). To show (b.2.b.2), we show for arbitrary $i \in \mathbb{N}_k$

(b.2.b.2.a.1) $[\![E]\!](t(i)) = \text{TRUE}$

(b.2.b.2.a.2) $[\![C']\!](t(i), t(i+1))$

From (14), we know (b.2.b.2.a.1) and

(15) $[\![C]\!](t(i), t(i+1))$

To show (b.2.b.2.a.2), from (9) and (15), it suffices to show

$$
\begin{array}{ll}
 & [\![H \texttt{ AND} \\
 & \quad \texttt{EXISTS } \$J_1,\ldots,\$J_n\texttt{:} \\
\text{(b.2.b.2.a.2.a)} & \quad\quad P[\$J_1/I_1,\ldots,\$J_n/I_n] \texttt{ AND} \\
 & \quad\quad (G[I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}][\$J_1/I_1,\ldots,\$J_n/I_n] \texttt{=>} \\
 & \quad\quad\quad G[\$J_1/I_1,\ldots,\$J_n/I_n,I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}])\,]\!](t(i))
\end{array}
$$

i.e. by the definition of $[\![\_]\!]$, for arbitrary $e \in \mathit{Environment}$,

$$
\begin{array}{ll}
\text{(b.2.b.1.a.2.b.1)} & [\![H]\!](e)(t(i),t(i)) \\
 & \exists v_1,\ldots,v_n \in \mathit{Value}: \\
 & \quad [\![P[\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e[J_1 \mapsto v_1,\ldots,J_n \mapsto v_n])(t(i),t(i)) \wedge \\
\text{(b.2.b.1.a.2.b.2)} & \quad ([\![G[I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}][\$J_1/I_1,\ldots,\$J_n/I_n]]\!] \\
 & \quad\quad (e[J_1 \mapsto v_1,\ldots,J_n \mapsto v_n])(t(i),t(i)) \Rightarrow \\
 & \quad\quad [\![G[\$J_1/I_1,\ldots,\$J_n/I_n,I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]]\!] \\
 & \quad\quad\quad (e[J_1 \mapsto v_1,\ldots,J_n \mapsto v_n])(t(i),t(i)))
\end{array}
$$

From (4), (14), and the definition of $\simeq$, we know (b.2.b.1.a.2.b.1).

We define

(16) $e_0 := e[J_1 \mapsto \mathit{read}(s,I_1),\ldots,J_n \mapsto \mathit{read}(s,I_n)]$

From (16), to show (b.2.b.1.a.2.b.2), it suffices to show

(b.2.b.1.a.2.b.2.a.1) $[\![P[\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e_0)(t(i),t(i))$

(b.2.b.1.a.2.b.2.a.2) $[\![G[I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}][\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e_0)(t(i),t(i)) \Rightarrow$
$[\![G[\$J_1/I_1,\ldots,\$J_n/I_n,I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]]\!](e_0)(t(i),t(i))$

From (5) and the soundness of the verification calculus, we know

(17) $\forall s,s' \in \mathit{Store} : [\![C]\!](s,s') \Rightarrow s = s' \text{ EXCEPT } I_1,\ldots,I_n$

From (14) and (17), we know

(18) $\forall i \in \mathbb{N}_k : t(i) = t(i+1) \text{ EXCEPT } I_1,\ldots,I_n$

From (13), (18), and (TRE), we know

(19) $s = t(i) \text{ EXCEPT } I_1,\ldots,I_n$

From (2), (6), (7), (16), (19), and (PMVF1), we know

(20) $[\![P[\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e_0)(t(i),s)$

From (1b), (20), and (PVF2), we know (b.2.b.1.a.2.b.2.a.1).

To show (b.2.b.1.a.2.b.2.a.2), we assume

(21) $[\![G[I_1/I_1{}', \ldots, I_n/I_n{}'][\$J_1/I_1, \ldots, \$J_n/I_n]]\!](e_0)(t(i),t(i))$

and show

(b.2.b.1.a.2.b.2.a.2.a) $[\![G[\$J_1/I_1, \ldots, \$J_n/I_n, I_1/I_1{}', \ldots, I_n/I_n{}']]\!](e_0)(t(i),t(i))$

From (6), (7), (16), (19), (21), and (PMVF1), we know

(22) $[\![G[I_1/I_1{}', \ldots, I_n/I_n{}']]\!](e)(s,t(i))$

From (22), (PPVF0), and (PVF4), we know

(23) $\forall s' \in \text{State} : s = s' \text{ EXCEPT } I_1, \ldots, I_n \Rightarrow$
$[\![G[I_1/I_1{}', \ldots, I_n/I_n{}']]\!](e)(s,s')$

From (23), we can show as demonstrated in the proof of the soundness of the invariant rule (Section 3.2.2)

(24) $\forall i \in \mathbb{N}_{k+1} : [\![G]\!](e)(s,t(i))$

From (24), we know

(25) $[\![G]\!](e)(s,t(i))$

From (6), (7), (16), (19), (22), and (PMVF1), we know

(26) $[\![G[\$J_1/I_1, \ldots, \$J_n/I_n]]\!](e_0)(t(i),t(i))$

From (26) and (IDE), we know

(27) $[\![G[\$J_1/I_1, \ldots, \$J_n/I_n]]\!](e_0)$
$(t(i), writes(t(i), I_1, read(t(i), I_1), \ldots, I_n, read(t(i), I_n)))$

From (27) and (PPVF2), we know (b.2.b.1.a.2.b.2.a.2.a). □

Chapter 5

Interrupting the Control Flow

In this chapter, we will model various kinds of commands that interrupt a program's normal control flow by jumping to the end of a loop body (`continue`), terminating the execution of a loop (`break`), returning from the execution of a program (`return`), or raising an exception (`throw`) that redirects the control flow to an error handler. These commands simplify the construction of programs that have to deal with special situations; however they also complicate the reasoning, because the static structure of a program does not directly reflect its dynamic behavior any more. Fortunately, those parts of a program that do not make use of these commands can be treated by the simpler calculus elaborated in the previous chapters such that the use of the more complex calculus presented in this chapter can be confined to those parts where interruptions actually play a role.

5.1 Programs with Interruptions

To adequately deal with the situation that a command may interrupt the normal program flow, we extend our notion of a program state as described in Figure 5.1. The state does not only consist of the store any more but also includes *control* data including a *flag* that indicates the current situation of the control flow: executing normally (E), continuing with the next iteration of a loop (C), breaking out of a loop (B), returning from the program (R) or throwing an exception (T). The control data may also carry a *key* (the kind of exception raised) and a *value* (a return/exception value). To abstract from the low-level representation of states and control data, also various auxiliary functions and predicates are introduced.

Because the notion of a state has changed, Figure 5.2 also redefines the fundamental operations on these states. With these modifications, all properties stated

$$
\begin{array}{l}
\mathit{Flag} := \{\mathrm{E},\mathrm{C},\mathrm{B},\mathrm{R},\mathrm{T}\} \\
\mathit{Key} := \mathit{Value} \\
\mathit{Control} := \mathit{Flag} \times \mathit{Key} \times \mathit{Value} \\
\mathit{State} := \mathit{Store} \times \mathit{Control} \\
\\
\mathit{store} : \mathit{State} \to \mathit{Store}, \mathit{store}(s,c) = s \\
\mathit{control} : \mathit{State} \to \mathit{Control}, \mathit{control}(s,c) = c \\
\\
\mathit{flag} : \mathit{Control} \to \mathit{Flag}, \mathit{flag}(f,k,v) = f \\
\mathit{key} : \mathit{Control} \to \mathit{Key}, \mathit{key}(f,k,v) = k \\
\mathit{value} : \mathit{Control} \to \mathit{Value}, \mathit{value}(f,k,v) = v \\
\\
\mathit{execute} : \mathit{State} \to \mathit{State} \\
\mathit{execute}(s) = \\
\quad \text{LET } c = \mathit{control}(s) \text{ IN } (\mathit{store}(s), (\mathrm{E}, \mathit{key}(c), \mathit{value}(c)) \\
\mathit{continue} : \mathit{State} \to \mathit{State} \\
\mathit{continue}(s) = \\
\quad \text{LET } c = \mathit{control}(s) \text{ IN } (\mathit{store}(s), (\mathrm{C}, \mathit{key}(c), \mathit{value}(c)) \\
\mathit{break} : \mathit{State} \to \mathit{State} \\
\mathit{break}(s) = \\
\quad \text{LET } c = \mathit{control}(s) \text{ IN } (\mathit{store}(s), (\mathrm{B}, \mathit{key}(c), \mathit{value}(c)) \\
\mathit{return} : \mathit{State} \times \mathit{Value} \to \mathit{State} \\
\mathit{return}(s,v) = \\
\quad \text{LET } c = \mathit{control}(s) \text{ IN } (\mathit{store}(s), (\mathrm{R}, \mathit{key}(c), v)) \\
\mathit{throw} : \mathit{State} \times \mathit{Key} \times \mathit{Value} \to \mathit{State} \\
\mathit{throw}(s,k,v) = \\
\quad \text{LET } c = \mathit{control}(s) \text{ IN } (\mathit{store}(s), (\mathrm{B}, k, v)) \\
\\
\mathit{executes} : \mathbb{P}(\mathit{Control}), \mathit{executes}(c) \Leftrightarrow \mathit{flag}(c) = \mathrm{E} \\
\mathit{continues} : \mathbb{P}(\mathit{Control}), \mathit{continues}(c) \Leftrightarrow \mathit{flag}(c) = \mathrm{C} \\
\mathit{breaks} : \mathbb{P}(\mathit{Control}), \mathit{breaks}(c) \Leftrightarrow \mathit{flag}(c) = \mathrm{B} \\
\mathit{returns} : \mathbb{P}(\mathit{Control}), \mathit{returns}(c) \Leftrightarrow \mathit{flag}(c) = \mathrm{R} \\
\mathit{throws} : \mathbb{P}(\mathit{Control}), \mathit{throws}(c) \Leftrightarrow \mathit{flag}(c) = \mathrm{T}
\end{array}
$$

Figure 5.1: States with Control Flow Flags (Part 1 of 2)

$$
\begin{array}{l}
\mathit{read} : \mathit{State} \times \text{Identifier} \to \mathit{Value} \\
\mathit{read}(s, I) = \mathit{store}(s)([\![I]\!])
\end{array}
$$

$$
\begin{array}{l}
\mathit{write} : \mathit{State} \times \text{Identifier} \times \mathit{Value} \to \mathit{State} \\
\mathit{write}(s, I, v) = (\mathit{store}(s)[[\![I]\!] \mapsto v], \mathit{control}(s))
\end{array}
$$

$$
\begin{array}{l}
\mathit{writes}(s, I_1, v_1, \ldots, I_n, v_n) \equiv \\
\quad (\mathit{store}(s)[[\![I_1]\!] \mapsto v_1] \ldots [[\![I_n]\!] \mapsto v_n], \mathit{control}(s))
\end{array}
$$

Figure 5.2: States with Control Flow Flags (Part 2 of 2)

in Appendix B.1 for states as plain stores still hold in the new setting (we omit the corresponding proofs). Also the properties stated in Appendix B.2 about phrases (formulas and terms) evaluated on such stores still hold, but only for the formula language introduced up to now, not for the extensions of the formula language which are going to be introduced in the next subsection.

Furthermore, we have the following property.

Lemma (State Control Predicates) The control predicates cover all situations and are mutually exclusive, i.e.:

$$
\begin{array}{l}
\forall c \in \mathit{Control} : \\
\quad (\mathit{executes}(c) \vee \mathit{continues}(c) \vee \mathit{breaks}(c) \vee \\
\qquad \mathit{returns}(c) \vee \mathit{throws}(c)) \wedge \\
\quad (\mathit{executes}(c) \Rightarrow \\
\qquad \neg(\mathit{continues}(c) \vee \mathit{breaks}(c) \vee \mathit{returns}(c) \vee \mathit{throws}(c))) \wedge \\
\quad (\mathit{continues}(c) \Rightarrow \\
\qquad \neg(\mathit{executes}(c) \vee \mathit{breaks}(c) \vee \mathit{returns}(c) \vee \mathit{throws}(c))) \wedge \\
\quad (\mathit{breaks}(c) \Rightarrow \\
\qquad \neg(\mathit{executes}(c) \vee \mathit{continues}(c) \vee \mathit{returns}(c) \vee \mathit{throws}(c))) \wedge \\
\quad (\mathit{returns}(c) \Rightarrow \\
\qquad \neg(\mathit{executes}(c) \vee \mathit{continues}(c) \vee \mathit{breaks}(c) \vee \mathit{throws}(c))) \wedge \\
\quad (\mathit{throws}(c) \Rightarrow \\
\qquad \neg(\mathit{executes}(c) \vee \mathit{continues}(c) \vee \mathit{breaks}(c) \vee \mathit{returns}(c)))
\end{array}
$$

We omit the corresponding proof. □

Based on these new definitions, Figure 5.3 presents an extended version of the command language with the semantics appropriately updated (the handling of loops is deferred until Section 5.4):

Command Language with Interruptions
An extension of the language of Figure 3.1.

Abstract Syntax

$C ::= \ldots$
$\quad \mid \texttt{continue} \mid \texttt{break} \mid \texttt{return}\ E \mid \texttt{throw}\ I\ E$
$\quad \mid \texttt{try}\ C_1\ \texttt{catch(}I_k\ I_v\texttt{)}\ C_2.$

Semantic Domains and Operations
See Figures 5.1 and 5.2

Valuation Functions

$\ldots$
$[\![\texttt{var}\ I\texttt{;}\ C]\!](s,s') \Leftrightarrow$
$\quad \exists s_0, s_1 \in \mathit{State}:$
$\qquad s_0 = s \text{ EXCEPT } I \wedge \mathit{control}(s_0) = \mathit{control}(s) \wedge$
$\qquad [\![C]\!](s_0,s_1) \wedge s' = \mathit{write}(s_1, I, \mathit{read}(s,I))$
$[\![C_1\texttt{;}\ C_2]\!](s,s') \Leftrightarrow$
$\quad \exists s_0 \in \mathit{State}:$
$\qquad [\![C_1]\!](s,s_0) \wedge$
$\qquad \text{IF } \mathit{executes}(\mathit{control}(s_0)) \text{ THEN } [\![C_2]\!](s_0,s') \text{ ELSE } s' = s_0$

$[\![\texttt{continue}]\!](s,s') \Leftrightarrow s' = \mathit{continue}(s)$
$[\![\texttt{break}]\!](s,s') \Leftrightarrow s' = \mathit{break}(s)$
$[\![\texttt{return}\ E]\!](s,s') \Leftrightarrow s' = \mathit{return}(s, [\![E]\!](s))$
$[\![\texttt{throw}\ I\ E]\!](s,s') \Leftrightarrow s' = \mathit{throw}(s, I, [\![E]\!](s))$
$[\![\texttt{try}\ C_1\ \texttt{catch(}I_k\ I_v\texttt{)}\ C_2]\!](s,s') \Leftrightarrow$
$\quad \exists s_0, s_1, s_2 \in \mathit{State}:$
$\qquad [\![C_1]\!](s,s_0) \wedge$
$\qquad \text{IF } \mathit{throws}(\mathit{control}(s_0)) \wedge \mathit{key}(\mathit{control}(s_0)) = I_k \text{ THEN}$
$\qquad\quad s_1 = \mathit{write}(\mathit{execute}(s_0), I_v, \mathit{value}(\mathit{control}(s_0))) \wedge$
$\qquad\quad [\![C_2]\!](s_1,s_2) \wedge$
$\qquad\quad s' = \mathit{write}(s_2, I_v, \mathit{read}(s_0, I_v))$
$\qquad \text{ELSE } s' = s_0$

Figure 5.3: Command Language with Interruptions

C_1 ; C_2 The semantics of command sequences is updated such that C_2 is only executed, if the state after C_1 is still "executing"; otherwise C_2 is ignored.

`continue` This command flags the poststate as "continuing".

`break` This command flags the poststate as "breaking".

`return` E This command flags the poststate as "returning" and sets the return value to the value of E.

`throw` I E This command flags the poststate as "throwing" and sets the exception key to I and the exception value to the value of E.

`try` C_1 `catch(`I_k I_v`)` C_2 This command first executes the "protected" command C_1. If the poststate of C_1 is not marked as "throwing exception with key I_k", this is also the ultimate poststate. Otherwise, the "exception handler" C_2 is executed with the parameter I_k set to the exception value.

We assume that a program starts in an "executing" prestate; by the commands above, however, its poststate, need not be "executing" any more. With the subsequently developed calculus it will be easy to detect which of "non-executing" states may emerge; thus it may be ensured that "continuing" or "breaking" states are captured by enclosing loops and that "throwing" states are caught by corresponding exception handlers. "Returning" states will be handled by the program method enclosing the command (see Chapter 6).

The new semantics of programs with interruptions is related to the old semantics of programs without interruptions in the following sense.

Theorem (Programs without Interruptions) Let C be a program that only contains constructs of the command language without interruptions. Then the following holds (where $[\![\_]\!]_o$ denotes the semantics of programs without interruptions):

$$\begin{aligned}&\forall s, s' \in State : executes(control(s)) \Rightarrow \\ &\quad ([\![C]\!](s,s') \Leftrightarrow \\ &\quad (control(s) = control(s') \wedge [\![C]\!]_o(store(s), store(s'))))\end{aligned}$$

We omit the corresponding proof. □

Formula Language with Control Predicates
An extension of the language of Figure 2.4.

Abstract Syntax

$S \in \text{State}$

$$
\begin{aligned}
F ::= \ & \ldots \\
& \mid \texttt{ALLSTATE}\ \#I_1, \ldots, \#I_n : F \\
& \mid \texttt{EXSTATE}\ \#I_1, \ldots, \#I_n : F \\
& \mid S_1 == S_2 \\
& \mid S\texttt{.executes} \mid S\texttt{.continues} \mid S\texttt{.breaks} \\
& \mid S\texttt{.returns} \mid S\texttt{.throws} \mid S\texttt{.throws}\ I
\end{aligned}
$$

$$T ::= \ldots \mid S\texttt{.value}$$

$$S ::= \texttt{now} \mid \texttt{next} \mid \#I$$

Figure 5.4: Formula Language with Control Predicates (Part 1 of 2)

5.2 Specifying Programs with Interruptions

To specify programs that interrupt the control flow, we extend the formula language as shown in Figure 5.4:

- We introduce a syntactic domain "State" with constants `now` and `next` to refer to the (control data of) the prestate and the poststate of a command, respectively.
- We introduce a term S`.value` which denotes the return/exception value of the state denoted by S.
- We introduce the formulas S`.executes`, S`.continues`, S`.breaks`, S`.returns`, S`.throws`, S`.throws` I, that are true if the control flow of the state denoted by S is executing, continuing with the next loop iteration, breaking out of a loop, returning from the program, throwing an exception, and throwing an exception with key I, respectively.
- We introduce state variables of the form $\#I$ ranging over states; correspondingly we introduce quantified formulas `ALLSTATE` and `EXSTATE` that bind state variables $\#I_1, \ldots, \#I_n$.

Formula Language with Control Predicates

Semantic Domains and Operations

$ControlEnv := \text{Identifier} \to Control$
$Environment := ValueEnv \times ControlEnv$

$(e_v, e_c)(I) \equiv e_v(I)$
$(e_v, e_c)(I)_c \equiv e_c(I)$
$(e_v, e_c)[I_1 \mapsto v_1, \ldots, I_n \mapsto v_n] \equiv (e_v[I_1 \mapsto v_1, \ldots, I_n \mapsto v_n], e_c)$
$(e_v, e_c)[I_1 \mapsto c_1, \ldots, I_n \mapsto c_n]_c \equiv (e_v, e_c[I_1 \mapsto c_1, \ldots, I_n \mapsto c_n])$

Valuation Functions

$[\![\_]\!] : \textbf{Formula} \to \textbf{\textit{Environment}} \to \textbf{\textit{StateRelation}}$
$\ldots$
$[\![\texttt{ALLSTATE}\ \#I_1, \ldots, \#I_n : F]\!](e)(s,s') \Leftrightarrow$
$\quad \forall c_1, \ldots, c_n \in Control : [\![F]\!](e[I_1 \mapsto c_1, \ldots, I_n \mapsto c_n]_c)(s,s')$
$[\![\texttt{EXSTATE}\ \#I_1, \ldots, \#I_n : F]\!](e)(s,s') \Leftrightarrow$
$\quad \exists c_1, \ldots, c_n \in Control : [\![F]\!](e[I_1 \mapsto c_1, \ldots, I_n \mapsto c_n]_c)(s,s')$
$[\![S_1 == S_2]\!](e)(s,s') \Leftrightarrow [\![S_1]\!](e)(s,s') = [\![S_2]\!](e)(s,s')$
$[\![S\texttt{.executes}]\!](e)(s,s') \Leftrightarrow executes([\![S]\!](e)(s,s'))$
$[\![S\texttt{.continues}]\!](e)(s,s') \Leftrightarrow continues([\![S]\!](e)(s,s'))$
$[\![S\texttt{.breaks}]\!](e)(s,s') \Leftrightarrow breaks([\![S]\!](e)(s,s'))$
$[\![S\texttt{.returns}]\!](e)(s,s') \Leftrightarrow returns([\![S]\!](e)(s,s'))$
$[\![S\texttt{.throws}]\!](e)(s,s') \Leftrightarrow throws([\![S]\!](e)(s,s'))$
$[\![S\texttt{.throws}\ I]\!](e)(s,s') \Leftrightarrow$
$\quad \text{LET } c = [\![S]\!](e)(s,s') \text{ IN } throws(c) \wedge key(c) = I$

$[\![\_]\!] : \textbf{Term} \to \textbf{\textit{Environment}} \to (\textbf{\textit{State}} \times \textbf{\textit{State}}) \to \textbf{\textit{Value}}$
$\ldots$
$[\![S\texttt{.value}]\!](e)(s,s') = value([\![S]\!](e)(s,s'))$

$[\![\_]\!] : \textbf{State} \to \textbf{\textit{Environment}} \to (\textbf{\textit{State}} \times \textbf{\textit{State}}) \to \textbf{\textit{Control}}$
$[\![\texttt{now}]\!](e)(s,s') = control(s)$
$[\![\texttt{next}]\!](e)(s,s') = control(s')$
$[\![\#I]\!](e)(s,s') = e(I)_c$

Figure 5.5: Formula Language with Control Predicates (Part 2 of 2)

- We introduce formulas $S_1 == S_2$ that are true if the (control data of) the states S_1 and S_2 are identical.

With these extensions, it will be (as shown in the next subsection) possible to adequately specify the behavior of any command by a formula. As an example, the formula

```
int(x) AND int(y) =>
  writesonly x AND int(x') AND
  IF y = 0
    THEN next.throws DivisionByZero AND
         next.value = x AND x' = x
    ELSE next.executes AND
         EXISTS $r: x = x'*y+$r AND
           0 <= $r AND $r < abs(y)
```

is an appropriate specification of a program

```
if (y == 0) throw DivisionByZero x;
x = x/y;
```

that sets integer variable x to the quotient of x and the value of another integer variable y, but raises an exception "Division By Zero" which carries the exception value x, if y is 0.

With these extensions, the semantics of the formula language is updated as shown in Figure 5.5. We introduce the semantic domain *ControlEnv* of control environments mapping identifiers to control data and update *Environment* to contain tuples of value environments and control environments. Corresponding access operations are defined such that $e(I)$ provides (as before) the value of I in the value environment denoted by e while $e(I)_c$ provides the control object of I in the corresponding control environment; likewise the update operation $e[I \mapsto v]$ provides a new value v for I in the value environment while $e[I \mapsto c]_c$ provides a new control object c for I in the control environment. Based on these definitions, the semantics of the new formulas and new terms and of the state designators are defined.

The following lemma describes useful knowledge for reasoning about formulas with control predicates.

Lemma (Control Predicates) For all $e \in \mathit{Environment}, s, s' \in \mathit{State}$ and for all I, the following statements hold:

$$
\begin{array}{l}
[\![(\#I\texttt{.executes OR } \#I\texttt{.continues OR} \\
\quad \#I\texttt{.breaks OR } \#I\texttt{.returns OR } \#I\texttt{.throws}) \texttt{ AND} \\
(\#I\texttt{.executes => !(}\#I\texttt{.continues OR} \\
\quad \#I\texttt{.breaks OR } \#I\texttt{.returns OR } \#I\texttt{.throws)) AND} \\
(\#I\texttt{.continues => !(}\#I\texttt{.executes OR} \\
\quad \#I\texttt{.breaks OR } \#I\texttt{.returns OR } \#I\texttt{.throws)) AND} \\
(\#I\texttt{.breaks => !(}\#I\texttt{.executes OR} \\
\quad \#I\texttt{.continues OR } \#I\texttt{.returns OR } \#I\texttt{.throws)) AND} \\
(\#I\texttt{.returns => !(}\#I\texttt{.executes OR} \\
\quad \#I\texttt{.continues OR } \#I\texttt{.breaks OR } \#I\texttt{.throws)) AND} \\
(\#I\texttt{.throws => !(}\#I\texttt{.executes OR} \\
\quad \#I\texttt{.continues OR } \#I\texttt{.breaks OR } \#I\texttt{.returns}))]\!] \\
(e)(s,s')
\end{array}
$$

(i.e. the control predicates cover all possibilities and are mutually disjoint) and

$$
\begin{array}{l}
\forall I_k \in \text{Identifier} : \\
\quad [\![\#I\texttt{.throws } I_k \texttt{ => } \#I\texttt{.throws}]\!](e)(s,s')
\end{array}
$$

(i.e. if a specific exception is raised, then also the general exception predicate must hold) and also

$$
\begin{array}{l}
[\![\#I\texttt{.throws}]\!](e)(s,s') \Rightarrow \\
\quad \exists I_k \in \text{Identifier} : [\![\#I\texttt{.throws } I_k]\!](e)(s,s')
\end{array}
$$

(i.e. if an exception is raised, it has some specific key).

We omit the corresponding proof. □

It should be noted, that with the extensions to the formula language introduced in this subsection the properties on formulas and terms stated in Appendix B.2 do not necessarily hold any more; some of them have to be rephrased to take into account the control data of states respectively the potential of formulas to differentiate between states even if their store contents are identical. Appendix B.3 thus lists properties that are true for the extended formula language.

5.3 Verifying Programs with Interruptions

We are now going to update the calculus for deriving judgements $C : F$ where the interpretation of a judgement has been slightly changed as shown in Figure 5.6: the prerequisite now also includes the proposition $[\![now.executes]\!](e)(s,s')$ which boils down to $executes(s)$, i.e. we assume that C starts in an executing state.

Verification Calculus with Interruptions

Definitions

$[F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \equiv$
(F) `AND writesonly` $I_1,\ldots,I_n$ `AND`
`(next.continues =>` (F_c)`) AND`
`(next.breaks =>` (F_b)`) AND`
`(next.returns =>` (F_r)`) AND`
`(next.throws =>`
`(next.throws` K_1 `OR ... OR next.throws` K_n`))`

$\llcorner \simeq \llcorner : \mathbb{P}(\text{Expression} \times \text{Term})$
$E \simeq T \Leftrightarrow$
T has no free (mathematical or state) variables $\wedge$
T has no primed program variables $\wedge$
T has no occurrence of `next` $\wedge$
$\forall s, s' \in \mathit{Store}, e \in \mathit{Environment} : [\![E]\!](s) = [\![T]\!](e)(s,s')$

$\llcorner \simeq \llcorner : \mathbb{P}(\text{Expression} \times \text{Formula})$
$E \simeq F \Leftrightarrow$
F has no free (mathematical or state) variables $\wedge$
F has no primed program variables $\wedge$
F has no occurrence of `next` $\wedge$
$\forall s, s' \in \mathit{Store}, e \in \mathit{Environment}$:
$[\![E]\!](s) = \text{TRUE} \Leftrightarrow [\![F]\!](e)(s,s')$

Judgements

$C : F \Leftrightarrow$
$\forall s, s' \in \mathit{State}, e \in \mathit{Environment}$:
$[\![$`now.executes`$]\!](e)(s,s') \wedge [\![C]\!](s,s') \Rightarrow$
$[\![F]\!](e)(s,s')$

Figure 5.6: The Verification Calculus with Interruptions (Part 1 of 5)

Actually the calculus only derives judgements of the form $C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}$ where $[F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}$ extends $[F]_{I_1,\ldots,I_n}$ by boolean constants F_c, F_b, F_r and an identifier set $K_1, \ldots, K_m$ with the following interpretation:

- Only if F_c is `TRUE`, the command may result in a continuing state.
- Only if F_b is `TRUE`, the command may result in a breaking state.
- Only if F_r is `TRUE`, the command may result in a returning state.
- An exception may be raised only, if its key is in $\{K_1, \ldots, K_m\}$ (thus $m = 0$ implies that no exception may be raised).

This interpretation is indicated by the expansion of $[F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}$ to a formula as shown in Figure 5.6.

Figure 5.6 also strengthens the relation $\simeq$ to rule out free occurrences of state variables and of the state constant `next` (which will become relevant in the verification rules for loops given in Section 5.4).

Figures 5.7, 5.8, 5.9, and 5.10, give rules for deriving these judgements for all commands of the command language with interruptions (except for loops which are treated in the next section).

From these rules only special kinds of judgements can be derived.

Lemma (Closed Specifications) If a judgement $C : F$ can be derived by the rules of the verification calculus of the command language with interruptions, then F is closed, i.e. does not contain free occurrences of mathematical variables and not free occurrences of state variables.

Proof Analogous to the proof of the corresponding lemma in the verification calculus for commands without interruptions. □

Lemma (Constant Formulas) If $C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}$ can be derived by the rules of the verification calculus of the command language with interruptions, then the values of F_c, F_b, F_r do not depend on environments and states:

$$\begin{aligned}
&\forall e_0, e_1 \in \mathit{Environment}, s_0, s_1, s_0', s_1' : \\
&\quad (\llbracket F_c \rrbracket(e_0)(s_0, s_0') \Leftrightarrow \llbracket F_c \rrbracket(e_1)(s_1, s_1')) \wedge \\
&\quad (\llbracket F_b \rrbracket(e_0)(s_0, s_0') \Leftrightarrow \llbracket F_b \rrbracket(e_1)(s_1, s_1')) \wedge \\
&\quad (\llbracket F_r \rrbracket(e_0)(s_0, s_0') \Leftrightarrow \llbracket F_r \rrbracket(e_1)(s_1, s_1'))
\end{aligned}$$

Rules

$$\frac{C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \quad \text{p is a permutation of } \{1,\ldots,n\}}{C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_{p(1)},\ldots,I_{p(n)}}}$$

$$\frac{C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \quad I \neq I_1 \wedge \ldots \wedge I \neq I_n}{C : [F \texttt{ AND } I' = I]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n,I}}$$

$$\frac{C : [F]^{\texttt{FALSE},\texttt{FALSE},\texttt{FALSE},\emptyset}_{I_1,\ldots,I_n}}{C : [F]_{I_1,\ldots,I_n}}$$

$$\frac{C \text{ is a program without interruptions} \quad C : [F]_{I_1,\ldots,I_n}}{C : [F]^{\texttt{FALSE},\texttt{FALSE},\texttt{FALSE},\emptyset}_{I_1,\ldots,I_n}}$$

Figure 5.7: The Verification Calculus with Interruptions (Part 2 of 5)

Proof From the rules of the calculus, it is easy to see that only disjunctions of `TRUE` and `FALSE` can be derived. □

Theorem (Soundness of Verification Calculus) Under the assumption denoted by *DifferentVariables*,the following is true: if a judgement $C : F$ can be derived from the rules of the verification calculus of the command language with interruptions, then

$$\begin{aligned}&\forall s, s' \in \mathit{State}, e \in \mathit{Environment} :\\ &\quad [\![\texttt{now.executes}]\!](e)(s,s') \wedge [\![C]\!](s,s') \Rightarrow\\ &\qquad [\![F]\!](e)(s,s')\end{aligned}$$

Proof We assume

(1) *DifferentVariables*

Take C and F such that $C : F$ can be derived and take arbitrary $s, s' \in \mathit{State}, e \in \mathit{Environment}$. We prove

$$[\![\texttt{now.executes}]\!](e)(s,s') \wedge [\![C]\!](s,s') \Rightarrow [\![F]\!](e)(s,s')$$

by induction on the derivation of $C : F$. The following subsections cover all cases for the last step of such a derivation; each subsection essentially shows the soundness of one derivation rule, i.e. that the interpretation of its conclusion is true under the assumption that the interpretations of its premises are true. □

Actually, we will only show detailed proofs for those cases where interruptions play a role; the other proofs are analogous to those of the calculus for programs without interruptions.

5.3.1 Frame Permutation

$$\frac{C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \quad \text{p is a permutation of } \{1,\ldots,n\}}{C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_{p(1)},\ldots,I_{p(n)}}}$$

Soundness Proof Analogous to the proof in the verification calculus without interruptions. □

5.3.2 Frame Extension

$$\frac{C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \quad I \neq I_1 \wedge \ldots \wedge I \neq I_n}{C : [F \texttt{ AND } I' = I]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n,I}}$$

Soundness Proof Analogous to the proof in the verification calculus without interruptions. □

5.3.3 No Interruptions (Part 1)

$$\frac{C : [F]^{\texttt{FALSE},\texttt{FALSE},\texttt{FALSE},\emptyset}_{I_1,\ldots,I_n}}{C : [F]_{I_1,\ldots,I_n}}$$

This rule allows us to embed commands that prevent the "escape" of their interruptions into the original verification calculus for programs without interruptions.

Verification Calculus with Interruptions: Rules

$$\frac{E \simeq T}{I = E : [I' = T \texttt{ AND next.executes}]_I^{\texttt{FALSE,FALSE,FALSE,}\emptyset}}$$

$$\frac{\begin{array}{l} C : [F]_{I_1,\ldots,I_n,I}^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}} \\ I_a \neq I_b \\ \$I_a \text{ and } \$I_b \text{ do not occur in } F \end{array}}{\begin{array}{l} \texttt{var } I\texttt{; } C : \\ \quad [\texttt{EXISTS } \$I_a\texttt{, } \$I_b : F[\$I_a/I, \$I_b/I']]_{I_1,\ldots,I_n}^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}} \end{array}}$$

$$\frac{\begin{array}{l} C : [F]_{I_1,\ldots,I_n,I}^{F_c,F_b,F_r,\{K_1,\ldots,K_m,\}} \\ I_a \neq I_b \\ \$I_a \text{ and } \$I_b \text{ do not occur in } F \\ E \simeq T \end{array}}{\begin{array}{l} \texttt{var } I\texttt{=}E\texttt{; } C : \\ \quad [\texttt{EXISTS } \$I_a\texttt{, } \$I_b : \\ \qquad \$I_a\texttt{=}T \texttt{ AND } F[\$I_a/I, \$I_b/I']]_{I_1,\ldots,I_n}^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}} \end{array}}$$

$$\frac{\begin{array}{l} C_1 : [F_1]_{I_1,\ldots,I_n}^{\texttt{FALSE,FALSE,FALSE,}\emptyset} \\ C_2 : [F_2]_{I_1,\ldots,I_n}^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}} \\ \$I_1,\ldots,\$I_n, \#I_s \text{ do not occur in } F_1 \text{ and } F_2 \end{array}}{\begin{array}{l} C_1\texttt{; }C_2 : \\ \quad [\texttt{EXISTS } \$I_1,\ldots,\$I_n\texttt{ : EXSTATE } \#I_s\texttt{ :} \\ \qquad F_1[\#I_s/\texttt{next}][\$I_1/I_1', \ldots, \$I_n/I_n'] \texttt{ AND} \\ \qquad F_2[\#I_s/\texttt{now}][\$I_1/I_1,\ldots,\$I_n/I_n]]_{I_1,\ldots,I_n}^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}} \end{array}}$$

Figure 5.8: The Verification Calculus with Interruptions (Part 3 of 5)

Verification Calculus with Interruptions: Rules

$$\frac{C_1 : [F_1]^{F_{c1},F_{b1},F_{r1},\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \quad C_2 : [F_2]^{F_{c1},F_{b1},F_{r1},\{L_1,\ldots,L_o\}}_{I_1,\ldots,I_n} \quad \$I_1,\ldots,\$I_n,\#I_s \text{ do not occur in } F_1 \text{ and } F_2}{\begin{array}{l} C_1\,;C_2 : \\ \quad [\texttt{EXISTS } \$I_1,\ldots,\$I_n\texttt{: EXSTATE } \#I_s\texttt{:} \\ \qquad F_1[\#I_s/\texttt{next}][\$I_1/I_1\texttt{'},\ldots,\$I_n/I_n\texttt{'}]\ \texttt{AND} \\ \qquad \texttt{IF } \#I_s\texttt{.executes THEN} \\ \qquad\quad F_2[\#I_s/\texttt{now}][\$I_1/I_1,\ldots,\$I_n/I_n] \\ \qquad \texttt{ELSE} \\ \qquad\quad I_1\texttt{'}=\$I_1\ \texttt{AND}\ \ldots\ \texttt{AND}\ I_n\texttt{'}=\$I_n\ \texttt{AND next==}\#I_s \\ \quad]^{F_{c1}\ \texttt{OR}\ F_{c2},F_{b1}\ \texttt{OR}\ F_{b2},F_{r1}\ \texttt{OR}\ F_{r2},\{K_1,\ldots,K_m,L_1,\ldots,L_o\}}_{I_1,\ldots,I_n} \end{array}}$$

$$\frac{C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \quad E \simeq F_0}{\texttt{if (}E\texttt{) } C : [\texttt{IF } F_0 \texttt{ THEN } F \texttt{ ELSE readsonly}]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}}$$

$$\frac{C_1 : [F_1]^{F_{c1},F_{b1},F_{r1},\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \quad C_2 : [F_2]^{F_{c2},F_{b2},F_{r2},\{L_1,\ldots,L_o\}}_{I_1,\ldots,I_n} \quad E \simeq F_0}{\begin{array}{l} \texttt{if (}E\texttt{) } C_1 \texttt{ else } C_2 : \\ \quad [\texttt{IF } F_0 \texttt{ THEN } F_1 \\ \qquad \texttt{ELSE } F_2]^{F_{c1}\ \texttt{OR}\ F_{c2},F_{b1}\ \texttt{OR}\ F_{b2},F_{r1}\ \texttt{OR}\ F_{r2},\{K_1,\ldots,K_m,L_1,\ldots,L_o\}}_{I_1,\ldots,I_n} \end{array}}$$

Figure 5.9: The Verification Calculus with Interruptions (Part 4 of 5)

Verification Calculus with Interruptions: Rules

$$\texttt{continue} : [\texttt{next.continues}]^{\texttt{TRUE,FALSE,FALSE,}\emptyset}_{\sqcup}$$

$$\texttt{break} : [\texttt{next.breaks}]^{\texttt{FALSE,TRUE,FALSE,}\emptyset}_{\sqcup}$$

$$\frac{T \simeq E}{\begin{array}{l}\texttt{return}\ E : \\ \quad [\texttt{next.returns} \\ \quad \texttt{AND next.value=}T]^{\texttt{FALSE,FALSE,TRUE,}\emptyset}_{\sqcup}\end{array}}$$

$$\frac{T \simeq E}{\begin{array}{l}\texttt{throw}\ I\ E : \\ \quad [\texttt{next.throws}\ I \\ \quad \texttt{AND next.value=}T]^{\texttt{FALSE,FALSE,FALSE,}\{I\}}_{\sqcup}\end{array}}$$

$$\frac{\begin{array}{l}C_1 : [F_1]^{F_{c1},F_{b1},F_{r1},\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \\ C_2 : [F_2]^{F_{c2},F_{b2},F_{r2},\{L_1,\ldots,L_o\}}_{I_1,\ldots,I_n} \\ \$I_1,\ldots,\$I_n, \#I_s \text{ do not occur in } F_1 \text{ and } F_2 \\ I_a \neq I_b \\ \{I_a, I_b\} \cap \{I_1,\ldots,I_n\} = \emptyset \\ I_s \neq I_t \\ \$I_a, \$I_b, \#I_t \text{ do not occur in } F_2\end{array}}{\begin{array}{l}\texttt{try}\ C_1\ \texttt{catch}\ (I_k\ I_v)\ C_2 : \\ \quad [\texttt{EXISTS}\ \$I_1,\ldots,\$I_n\texttt{:}\ \texttt{EXSTATE}\ \#I_s\texttt{:} \\ \qquad F_1[\#I_s/\texttt{next}][\$I_1/I_1\texttt{'},\ldots,\$I_n/I_n\texttt{'}]\ \texttt{AND} \\ \qquad \texttt{IF}\ \#I_s\texttt{.throws}\ I_k\ \texttt{THEN} \\ \qquad\quad \texttt{EXISTS}\ \$I_a,\$I_b\texttt{:}\ \texttt{EXSTATE}\ \#I_t\texttt{:} \\ \qquad\qquad \$I_a = \#I_s\texttt{.value AND}\ \#I_t\texttt{.executes AND} \\ \qquad\qquad F_2[\#I_t/\texttt{now}][\$I_a/I_v][\$I_1/I_1,\ldots,\$I_n/I_n][\$I_b/I_v\texttt{'}] \\ \qquad \texttt{ELSE} \\ \qquad\quad I_1\texttt{'} = \$I_1\ \texttt{AND}\ \ldots\ \texttt{AND}\ I_n\texttt{'} = \$I_n\ \texttt{AND}\ \texttt{next==}\#I_s \\ \quad]^{F_{c1}\ \texttt{OR}\ F_{c2},F_{b1}\ \texttt{OR}\ F_{b2},F_{r1}\ \texttt{OR}\ F_{r2},(\{K_1,\ldots,K_m\}\setminus\{I_k\})\cup\{L_1,\ldots,L_o\}}_{I_1,\ldots,I_n}\end{array}}$$

Figure 5.10: The Verification Calculus with Interruptions (Part 5 of 5)

Soundness Proof We have to show

$$\text{(a)} \quad \begin{array}{l} [\![\texttt{now.executes}]\!](e)(s,s') \wedge [\![C]\!](s,s') \Rightarrow \\ \quad [\![[F]_{I_1,\ldots,I_n}]\!](e)(s,s') \end{array}$$

We assume

$$\text{(2)} \quad [\![\texttt{now.executes}]\!](e)(s,s')$$

$$\text{(3)} \quad [\![C]\!](s,s')$$

By the definitions of $[\_]_\_$ and $[\![\_]\!]$, it suffices to show

$$\text{(a.1)} \quad [\![F]\!](e)(s,s')$$

$$\text{(a.2)} \quad s = s' \text{ EXCEPT } I_1,\ldots,I_n$$

From the premise and the soundness of the verification calculus, we know

$$\text{(5)} \quad \begin{array}{l} [\![\texttt{now.executes}]\!](e)(s,s') \wedge [\![C]\!](s,s') \Rightarrow \\ \quad [\![[F]_{I_1,\ldots,I_n}^{\texttt{FALSE},\texttt{FALSE},\texttt{FALSE},\emptyset}]\!](e)(s,s') \end{array}$$

From (2), (3) and (5), we know

$$\text{(6)} \quad [\![[F]_{I_1,\ldots,I_n}^{\texttt{FALSE},\texttt{FALSE},\texttt{FALSE},\emptyset}]\!](e)(s,s')$$

From (6) and the definition of $[\![\_]\!]$, we know (a.1) and (a.2). □

5.3.4 No Interruptions (Part 2)

$$\frac{\begin{array}{l} C \text{ is a program without interruptions} \\ C : [F]_{I_1,\ldots,I_n} \end{array}}{C : [F]_{I_1,\ldots,I_n}^{\texttt{FALSE},\texttt{FALSE},\texttt{FALSE},\emptyset}}$$

This rule allows to apply to a part of a program that does not use interruption commands the verification calculus for programs without interruptions and embed the judgement derived into the verification calculus for programs with interruptions.

Soundness Proof We have to show

(a) $$[\![\texttt{now.executes}]\!](e)(s,s') \wedge [\![C]\!](s,s') \Rightarrow [\![[F]^{\texttt{FALSE,FALSE,FALSE},\emptyset}_{I_1,\ldots,I_n}]\!](e)(s,s')$$

We assume

(2) $[\![\texttt{now.executes}]\!](e)(s,s')$

(3) $[\![C]\!](s,s')$

By the definitions of $[\,\_\,]_{\_}$ and $[\![\,\_\,]\!]$, it suffices to show

(a.1) $[\![F]\!](e)(s,s')$

(a.2) $s = s'$ EXCEPT $I_1,\ldots,I_n$

(a.3) $\neg \mathit{continues}(\mathit{control}(s'))$

(a.4) $\neg \mathit{breaks}(\mathit{control}(s'))$

(a.5) $\neg \mathit{returns}(\mathit{control}(s'))$

(a.6) $\neg \mathit{throws}(\mathit{control}(s'))$

From the premise and the soundness of the verification calculus, we know

(4) C is a program without interruptions

(5) $$[\![\texttt{now.executes}]\!](e)(s,s') \wedge [\![C]\!](s,s') \Rightarrow [\![[F]_{I_1,\ldots,I_n}]\!](e)(s,s')$$

From (2), (3) and (5), we know

(6) $[\![[F]_{I_1,\ldots,I_n}]\!](e)(s,s')$

From (6) and the definitions of $[\,\_\,]_{\_}$ and $[\![\,\_\,]\!]$, we know (a.1) and (a.2).

From (2) and the definition of $[\![\,\_\,]\!]$, we know

(7) $\mathit{executes}(\mathit{control}(s))$

From (3), (4), (7) and Theorem (Programs without Interruptions), we know

(8) $\mathit{executes}(\mathit{control}(s'))$

From (8) and Lemma “State Control Predicates”, we know (a.3), (a.4), (a.5), and (a.6). □

5.3.5 Assignment

$$\frac{E \simeq T}{I = E : [I' = T \texttt{ AND next.executes}]_I^{\texttt{FALSE,FALSE,FALSE,}\emptyset}}$$

Soundness Proof We have to show

(a) $[\![\texttt{now.executes}]\!](e)(s,s') \wedge [\![I = E]\!](s,s') \Rightarrow [\![[I' = T \texttt{ AND next.executes}]_I^{\texttt{FALSE,FALSE,FALSE,}\emptyset}]\!](e)(s,s')$

We assume

(2) $[\![\texttt{now.executes}]\!](e)(s,s')$

(3) $[\![I = E]\!](s,s')$

i.e. by the definition of $[\![\_]\!]$

(4) $executes(control(s))$

(5) $s' = write(s,I,[\![E]\!](s))$

By the definitions of $[\_]_\_$ and $[\![\_]\!]$, it suffices to show

(a.1) $read(s',I) = [\![T]\!](e)(s,s')$

(a.2) $executes(control(s'))$

(a.3) $s = s'$ EXCEPT I

(a.4) $\neg continues(control(s'))$

(a.5) $\neg breaks(control(s'))$

(a.6) $\neg returns(control(s'))$

(a.7) $\neg throws(control(s'))$

We can show (a.1) and (a.3) as in the verification calculus without interruptions. From (4), (5), and (CW), we know (a.2). From (a.2) and Lemma "State Control Predicates", we know (a.4), (a.5), (a.6), and (a.7). □

5.3.6 Variable Declaration

$$\frac{\begin{array}{l} C : [F]_{I_1,\ldots,I_n,I}^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}} \\ I_a \neq I_b \\ \$I_a \text{ and } \$I_b \text{ do not occur in } F \end{array}}{\begin{array}{l} \texttt{var } I\texttt{; } C : \\ \quad [\texttt{EXISTS } \$I_a\texttt{, } \$I_b : F[\$I_a/I, \$I_b/I']]_{I_1,\ldots,I_n}^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}} \end{array}}$$

Soundness Proof Analogous to the proof in the verification calculus without interruptions. □

5.3.7 Variable Definition

$$\frac{\begin{array}{l} C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m,\}}_{I_1,\ldots,I_n,I} \\ I_a \neq I_b \\ \$I_a \text{ and } \$I_b \text{ do not occur in } F \\ E \simeq T \end{array}}{\begin{array}{l} \texttt{var}\ I{=}E\texttt{;}\ C : \\ \quad [\texttt{EXISTS}\ \$I_a\texttt{,}\ \$I_b : \\ \quad\quad \$I_a{=}T\ \texttt{AND}\ F[\$I_a/I, \$I_b/I\texttt{'}]]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \end{array}}$$

Soundness Proof Analogous to the proof in the verification calculus without interruptions. □

5.3.8 Command Sequence (Without Interruptions)

$$\frac{\begin{array}{l} C_1 : [F_1]^{\texttt{FALSE},\texttt{FALSE},\texttt{FALSE},\emptyset}_{I_1,\ldots,I_n} \\ C_2 : [F_2]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \\ \$I_1,\ldots,\$I_n, \#I_s \text{ do not occur in } F_1 \text{ and } F_2 \end{array}}{\begin{array}{l} C_1\texttt{;}\ C_2 : \\ \quad [\texttt{EXISTS}\ \$I_1,\ldots,\$I_n\texttt{:}\ \texttt{EXSTATE}\ \#I_s\texttt{:} \\ \quad\quad F_1[\#I_s/\texttt{next}][\$I_1/I_1\texttt{'},\ldots,\$I_n/I_n\texttt{'}]\ \texttt{AND} \\ \quad\quad F_2[\#I_s/\texttt{now}][\$I_1/I_1,\ldots,\$I_n/I_n]]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \end{array}}$$

This rule for a sequence of two commands handles the special situation where the first command does not interrupt the control flow such that all interruptions are those caused by the second command. In the resulting specification formula, analogous to the existentially quantified mathematical variables $\$I_1,\ldots,\$I_n$ which denote the values of the program variables in the intermediate state, the existentially quantified state variable $\#I_s$ denotes the intermediate state's control data. This variable correspondingly takes the role of `next` in the specification of the first command and of `now` in the specification of the second one.

Soundness Proof We have to show

$$\text{(a)}\quad \begin{array}{l} [\![\texttt{now.executes}]\!](e)(s,s') \wedge [\![C_1 \,;\, C_2]\!](s,s') \Rightarrow \\ \quad [\![[\texttt{EXISTS } \$I_1,\ldots,\$I_n\texttt{: EXSTATE } \#I_s\texttt{:} \\ \qquad F_1[\#I_s/\texttt{next}][\$I_1/I_1\texttt{'},\ldots,\$I_n/I_n\texttt{'}]\ \texttt{AND} \\ \qquad F_2[\#I_s/\texttt{now}][\$I_1/I_1,\ldots,\$I_n/I_n]] \,]\!]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}(e)(s,s') \end{array}$$

We assume

(2) $[\![\texttt{now.executes}]\!](e)(s,s')$

(3) $[\![C_1 \,;\, C_2]\!](s,s')$

i.e. by the definition of $[\![\_]\!]$ for some $s_0 \in \mathit{State}$

(4) $\mathit{executes}(\mathit{control}(s))$

(5) $[\![C_1]\!](s,s_0)$

(6) IF $\mathit{executes}(\mathit{control}(s_0))$ THEN $[\![C_2]\!](s_0,s')$ ELSE $s' = s_0$

By the definitions of $[\_]_\_$ and $[\![\_]\!]$, it suffices to show

$$\text{(a.1)}\quad \begin{array}{l} \exists v_1,\ldots,v_n \in \mathit{Value}, c \in \mathit{Control}: \\ \quad [\![F_1[\#I_s/\texttt{next}][\$I_1/I_1\texttt{'},\ldots,\$I_n/I_n\texttt{'}]]\!] \\ \qquad (e[I_s \mapsto c]_c[I_1 \mapsto v_1,\ldots,I_n \mapsto v_n])(s,s') \wedge \\ \quad [\![F_2[\#I_s/\texttt{now}][\$I_1/I_1,\ldots,\$I_n/I_n]]\!] \\ \qquad (e[I_s \mapsto c]_c[I_1 \mapsto v_1,\ldots,I_n \mapsto v_n])(s,s') \end{array}$$

(a.2) $s = s'$ EXCEPT $I_1,\ldots,I_n$

(a.3) $\mathit{continues}(\mathit{control}(s')) \Rightarrow [\![F_c]\!](e)(s,s')$

(a.4) $\mathit{breaks}(\mathit{control}(s')) \Rightarrow [\![F_b]\!](e)(s,s')$

(a.5) $\mathit{returns}(\mathit{control}(s')) \Rightarrow [\![F_r]\!](e)(s,s')$

(a.6) $\mathit{throws}(\mathit{control}(s')) \Rightarrow \mathit{key}(\mathit{control}(s')) \in \{K_1,\ldots,K_m\}$

From the premises, we know

$$\text{(7)}\quad \begin{array}{l} [\![\texttt{now.executes}]\!](e)(s,s_0) \wedge [\![C_1]\!](s,s_0) \Rightarrow \\ \quad [\![[F_1]^{\texttt{FALSE},\texttt{FALSE},\texttt{FALSE},\emptyset}_{I_1,\ldots,I_n}]\!](e)(s,s_0) \end{array}$$

$$\text{(8)}\quad \begin{array}{l} [\![\texttt{now.executes}]\!](e)(s_0,s') \wedge [\![C_2]\!](s_0,s') \Rightarrow \\ \quad [\![[F_2]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}]\!](e)(s_0,s') \end{array}$$

(8a) $\$I_1,\ldots,\$I_n, \#I_s$ do not occur in F_1 and F_2

From (4), (5), (7), and the definition of $[\![\_]\!]$, we know

(9) $[\![F_1]\!](e)(s,s_0)$

(10) $s = s_0$ EXCEPT $I_1,\ldots,I_n$

(11) $\neg continues(control(s_0))$

(12) $\neg breaks(control(s_0))$

(13) $\neg returns(control(s_0))$

(14) $\neg throws(control(s_0))$

From (11), (12), (13), (14), and Lemma "State Control Predicates", we know

(15) $executes(control(s_0))$

From (6), (8), (15), and the definition of $[\![\_]\!]$, we know

(16) $[\![F_2]\!](e)(s_0,s')$

(17) $s_0 = s'$ EXCEPT $I_1,\ldots,I_n$

(18) $continues(control(s')) \Rightarrow [\![F_c]\!](e)(s_0,s')$

(19) $breaks(control(s')) \Rightarrow [\![F_b]\!](e)(s_0,s')$

(20) $returns(control(s')) \Rightarrow [\![F_r]\!](e)(s_0,s')$

(21) $throws(control(s')) \Rightarrow key(control(s')) \in \{K_1,\ldots,K_m\}$

From (10), (17), and (TRE), we have (a.2).

From the second assumption and Lemma "Constant Formulas", we know

(22) $[\![F_c]\!](e)(s,s') \Leftrightarrow [\![F_c]\!](e)(s_0,s')$

(23) $[\![F_b]\!](e)(s,s') \Leftrightarrow [\![F_b]\!](e)(s_0,s')$

(24) $[\![F_r]\!](e)(s,s') \Leftrightarrow [\![F_r]\!](e)(s_0,s')$

From (18), (19), (20), (22), (23), and (24), we we know (a.3), (a.4), and (a.5). From (21), we know (a.6).

To show (a.2), it suffices to show

(a.1.a.1) $[\![F_1[\#I_s/\texttt{next}][\$I_1/I_1\texttt{'},\ldots,\$I_n/I_n\texttt{'}]]\!]$
$(e[I_s \mapsto control(s_0)]_c[I_1 \mapsto read(s_0,I_1),\ldots,I_n \mapsto read(s_0,I_n)])$
(s,s')

(a.1.a.2) $[\![F_2[\#I_s/\texttt{now}][\$I_1/I_1,\ldots,\$I_n/I_n]]\!]$
$(e[I_s \mapsto control(s_0)]_c[I_1 \mapsto read(s_0,I_1),\ldots,I_n \mapsto read(s_0,I_n)])$
(s,s')

From (8a), (9), and (CNEF2), we know

(25) $[\![F_1[\#I_s/\texttt{next}]]\!](e[I_s \mapsto \mathit{control}(s_0)]_c)(s, s_0)$

From (8a), (17), (25), (CNEF0), and (PMVF2'), we know (a.1.a.1).

From (8a), (16), and (CNOF2), we know

(26) $[\![F_2[\#I_s/\texttt{now}]]\!](e[I_s \mapsto \mathit{control}(s_0)]_c)(s_0, s')$

From (8a), (10), (26), (CNOF0), and (PMVF1'), we know (a.1.a.2). □

5.3.9 Command Sequence (With Interruptions)

$C_1 : [F_1]^{F_{c1},F_{b1},F_{r1},\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}$
$C_2 : [F_2]^{F_{c1},F_{b1},F_{r1},\{L_1,\ldots,L_o\}}_{I_1,\ldots,I_n}$
$\$I_1, \ldots, \$I_n, \#I_s$ do not occur in F_1 and F_2

$C_1 ; C_2$:
[EXISTS $\$I_1, \ldots, \$I_n$: EXSTATE $\#I_s$:
$F_1[\#I_s/\texttt{next}][\$I_1/I_1', \ldots, \$I_n/I_n']$ AND
IF $\#I_s$.executes THEN
$F_2[\#I_s/\texttt{now}][\$I_1/I_1, \ldots, \$I_n/I_n]$
ELSE
$I_1' = \$I_1$ AND ... AND $I_n' = \$I_n$ AND next==$\#I_s$
$]^{F_{c1} \text{ OR } F_{c2},F_{b1} \text{ OR } F_{b2},F_{r1} \text{ OR } F_{r2},\{K_1,\ldots,K_m,L_1,\ldots,L_o\}}_{I_1,\ldots,I_n}$

This rule handles the general case of a sequence of commands where the first one may interrupt the control flow such that the second one is not executed. Correspondingly, the post-state either (if the first command does not trigger an interrupt) is determined by the second command or (if the first command indeed triggers an interrupt) equals the poststate of the first command. For instance, the command sequence

```
if (x=0) throw DivByZero y else y = y/x;
z=z+y;
```

is specified by $[F]^{\texttt{FALSE,FALSE,FALSE,\{DivByZero\}}}_{\texttt{y,z}}$ where F is the formula

```
EXISTS $y, $z: EXSTATE #s:
    IF x=0 THEN
```

```
      $y=y AND $z=z AND
      #s.throws DivByZero AND #s.value = y
    ELSE
      $y=y/x AND $z=z AND #s.executes
  AND
    IF #s.executes THEN
      y'=$y AND z'=$z+$y AND next.executes
    ELSE
      y'=$y AND z'=$z AND next==#s
```

which can be simplified to

```
IF x=0 THEN
  y'=y AND z'=z AND
  next.throws DivByZero AND next.value=y
ELSE
  y'=y/x AND z'=z+y/x AND
  next.executes
```

which succinctly describes the overall effect of the program sniplet.

Soundness Proof We have to show

$$
\begin{array}{ll}
 & [\![\texttt{now.executes}]\!](e)(s,s') \wedge [\![C_1\,;C_2]\!](s,s') \Rightarrow \\
 & \quad [\![\texttt{[EXISTS \$}I_1,\ldots,\texttt{\$}I_n\texttt{: EXSTATE \#}I_s\texttt{:} \\
 & \qquad F_1[\#I_s/\texttt{next}][\$I_1/I_1\texttt{'},\ldots,\$I_n/I_n\texttt{'}]\ \texttt{AND} \\
 & \qquad \texttt{IF \#}I_s\texttt{.executes THEN} \\
\text{(a)} & \qquad\quad F_2[\#I_s/\texttt{now}][\$I_1/I_1,\ldots,\$I_n/I_n] \\
 & \qquad \texttt{ELSE} \\
 & \qquad\quad I_1\texttt{'}=\$I_1\ \texttt{AND}\ \ldots\ \texttt{AND}\ I_n\texttt{'}=\$I_n\ \texttt{AND next==\#}I_s \\
 &]\!]^{F_{c1}\ \texttt{OR}\ F_{c2},F_{b1}\ \texttt{OR}\ F_{b2},F_{r1}\ \texttt{OR}\ F_{r2},\{K_1,\ldots,K_m,L_1,\ldots,L_o\}}_{I_1,\ldots,I_n}(e)(s,s')
\end{array}
$$

We assume

(2) $[\![\texttt{now.executes}]\!](e)(s,s')$

(3) $[\![C_1\,;C_2]\!](s,s')$

i.e. by the definition of $[\![\,\_\,]\!]$ for some $s_0 \in \mathit{State}$

(4) $\mathit{executes}(\mathit{control}(s))$

(5) $[\![C_1]\!](s, s_0)$

(6) IF $\mathit{executes}(\mathit{control}(s_0))$ THEN $[\![C_2]\!](s_0, s')$ ELSE $s' = s_0$

By the definitions of $[\,\_\,]_\_$ and $[\![\,\_\,]\!]$, it suffices to show

(a.1)
$$\begin{array}{l}\exists v_1, \ldots, v_n \in \mathit{Value}, c \in \mathit{Control}: \\ \quad \text{LET} \\ \quad\quad e_0 = e[I_s \mapsto c]_c[I_1 \mapsto v_1, \ldots, I_n \mapsto v_n] \\ \quad \text{IN} \\ \quad\quad [\![F_1[\#I_s/\texttt{next}][\$I_1/I_1\texttt{'}, \ldots, \$I_n/I_n\texttt{'}]]\!](e_0)(s, s') \wedge \\ \quad\quad \text{IF } \mathit{executes}(e_0(I_s)) \text{ THEN} \\ \quad\quad\quad [\![F_2[\#I_s/\texttt{now}][\$I_1/I_1, \ldots, \$I_n/I_n]]\!](e_0)(s, s') \wedge \\ \quad\quad \text{ELSE} \\ \quad\quad\quad \mathit{read}(s', I_1) = e_0(I_1) \wedge \ldots \wedge \mathit{read}(s', I_n) = e_0(I_n) \wedge \\ \quad\quad\quad \mathit{control}(s') = e_0(I_s)\end{array}$$

(a.2) $s = s'$ EXCEPT $I_1, \ldots, I_n$

(a.3) $\mathit{continues}(\mathit{control}(s')) \Rightarrow [\![F_{c1}\ \texttt{OR}\ F_{c2}]\!](e)(s, s')$

(a.4) $\mathit{breaks}(\mathit{control}(s')) \Rightarrow [\![F_{b1}\ \texttt{OR}\ F_{b2}]\!](e)(s, s')$

(a.5) $\mathit{returns}(\mathit{control}(s')) \Rightarrow [\![F_{r1}\ \texttt{OR}\ F_{r2}]\!](e)(s, s')$

(a.6) $\mathit{throws}(\mathit{control}(s')) \Rightarrow \mathit{key}(\mathit{control}(s')) \in \{K_1, \ldots, K_m, L_1, \ldots, L_o\}$

We define

(7) $e_0 := e[I_s \mapsto \mathit{control}(s_0)]_c[I_1 \mapsto \mathit{read}(s_0, I_1), \ldots, I_n \mapsto \mathit{read}(s_0, I_n)]$

To show (a.1), it suffices to show

(a.1.a.1) $[\![F_1[\#I_s/\texttt{next}][\$I_1/I_1\texttt{'}, \ldots, \$I_n/I_n\texttt{'}]]\!](e_0)(s, s')$

$$\begin{array}{l}\text{IF } \mathit{executes}(e_0(I_s)) \text{ THEN} \\ \quad [\![F_2[\#I_s/\texttt{now}][\$I_1/I_1, \ldots, \$I_n/I_n]]\!](e_0)(s, s') \wedge \\ \text{ELSE} \\ \quad \mathit{read}(s', I_1) = e_0(I_1) \wedge \ldots \wedge \mathit{read}(s', I_n) = e_0(I_n) \wedge \\ \quad \mathit{control}(s') = e_0(I_s)\end{array}$$

(a.1.a.2)

From the premises, we know

(8) $$[\![\texttt{now.executes}]\!](e)(s, s_0) \wedge [\![C_1]\!](s, s_0) \Rightarrow [\![\,[F_1]_{I_1, \ldots, I_n}^{F_{c1}, F_{b1}, F_{r1}, \{K_1, \ldots, K_m\}}\,]\!](e)(s, s_0)$$

(9) $$[\![\texttt{now.executes}]\!](e)(s_0, s') \wedge [\![C_2]\!](s_0, s') \Rightarrow [\![\,[F_2]_{I_1, \ldots, I_n}^{F_{c2}, F_{b2}, F_{r2}, \{L_1, \ldots, L_o\}}\,]\!](e)(s_0, s')$$

(9a) $\$I_1, \ldots, \$I_n, \#I_s$ do not occur in F_1 and F_2

From (2) and the definition of $[\![\_]\!]$, we know

(10) $[\![\texttt{now.executes}]\!](e)(s, s_0)$

From (5), (8), and (10), we know

(11) $[\![[F_1]_{I_1,\ldots,I_n}^{F_{c1},F_{b1},F_{r1},\{K_1,\ldots,K_m\}}]\!](e)(s, s_0)$

i.e. from the definitions of $[\_]\_$ and $[\![\_]\!]$

(12) $[\![F_1]\!](e)(s, s_0)$
(13) $s = s_0$ EXCEPT $I_1, \ldots, I_n$
(14) $\mathit{continues}(\mathit{control}(s_0)) \Rightarrow [\![F_{c1}]\!](e)(s, s_0)$
(15) $\mathit{breaks}(\mathit{control}(s_0)) \Rightarrow [\![F_{b1}]\!](e)(s, s_0)$
(16) $\mathit{returns}(\mathit{control}(s_0)) \Rightarrow [\![F_{r1}]\!](e)(s, s_0)$
(17) $\mathit{throws}(\mathit{control}(s_0)) \Rightarrow \mathit{key}(\mathit{control}(s_0)) \in \{K_1, \ldots, K_m\}$

From (9a), (12), and (CNEF2), we know

(20) $[\![F_1[\#I_s/\texttt{next}]]\!](e[I_s \mapsto \mathit{control}(s_0)]_c)(s, s0)$

We proceed by case distinction:

- Case $\mathit{executes}(\mathit{control}(s_0))$: from the case condition and the definition of $[\![\_]\!]$, we know

  (21) $[\![\texttt{now.executes}]\!](e)(s_0, s')$

  From the case condition, (6), (9), and (21), we know

  (22) $[\![[F_2]_{I_1,\ldots,I_n}^{F_{c2},F_{b2},F_{r2},\{L_1,\ldots,L_o\}}]\!](e)(s_0, s')$

  i.e. from the definitions of $[\_]\_$ and $[\![\_]\!]$

  (23) $[\![F_2]\!](e)(s_0, s')$
  (24) $s_0 = s'$ EXCEPT $I_1, \ldots, I_n$
  (25) $\mathit{continues}(\mathit{control}(s')) \Rightarrow [\![F_{c2}]\!](e)(s_0, s')$
  (26) $\mathit{breaks}(\mathit{control}(s')) \Rightarrow [\![F_{b2}]\!](e)(s_0, s')$
  (27) $\mathit{returns}(\mathit{control}(s')) \Rightarrow [\![F_{r2}]\!](e)(s_0, s')$
  (28) $\mathit{throws}(\mathit{control}(s')) \Rightarrow \mathit{key}(\mathit{control}(s')) \in \{L_1, \ldots, L_o\}$

From (7), (9a), (20), (24), (CNEF0), and (PMVF2'), we know (a.1.a.1).

From the case condition and (7), we know

(29) $\mathit{executes}(e_0(I_s))$

From (29), to show (a.1.a.2), it suffices to show

(a.1.a.2.a) $[\![F_2[\#I_s/\mathtt{now}][\$I_1/I_1,\ldots,\$I_n/I_n]]\!](e_0)(s,s')$

From (9a), (23), and (CNOF2), we know

(30) $[\![F_2[\#I_s/\mathtt{now}]]\!](e[I_s \mapsto \mathit{control}(s_0)]_c)(s_0,s')$

From (7), (9a), (13), (30), (CNOF0), and (PMVF1'), we know (a.1.a.2.a).

From (13), (24), and (TRE), we know (a.2). From (25) and the definition of $[\![\_]\!]$, we know (a.3). From (26) and the definition of $[\![\_]\!]$, we know (a.4). From (27) and the definition of $[\![\_]\!]$, we know (a.5). From (28) and the definition of $[\![\_]\!]$, we know (a.6).

- Case $\neg\mathit{executes}(\mathit{control}(s_0))$: from the case condition and (6), we know

  (31) $s' = s_0$

  and thus from (REE)

  (32) $s_0 = s' \text{ EXCEPT } I_1,\ldots,I_n$

  From (7), (9a), (20), (32), (CNEF0), and (PMVF2'), we know (a.1.a.1).

  From the case condition and (7), we know

  (33) $\neg\mathit{executes}(e_0(I_s))$

  From (33), to show (a.1.a.2), it suffices to show

  (a.1.a.2.a) $\mathit{read}(s',I_1) = e_0(I_1) \wedge \ldots \wedge \mathit{read}(s',I_n) = e_0(I_n) \wedge \mathit{control}(s') = e_0(I_s)$

  From (7) and (31), we have (a.1.a.2.a).

  From (13), (32), and (TRE), we know (a.2). From (14), (31), and the definition of $[\![\_]\!]$, we know (a.3). From (15), (31), and the definition of $[\![\_]\!]$, we know (a.3). From (16), (31), and the definition of $[\![\_]\!]$, we know (a.3). From (17), (31), and the definition of $[\![\_]\!]$, we know (a.3). □

5.3.10 One-Sided Conditional

$$\frac{C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \qquad E \simeq F_0}{\texttt{if (}E\texttt{) } C : [\texttt{IF } F_0 \texttt{ THEN } F \texttt{ ELSE readsonly}]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}}$$

Soundness Proof We have to show

(a) $$[\![\texttt{now.executes}]\!](e)(s,s') \wedge [\![\texttt{if (}E\texttt{) } C]\!](s,s') \Rightarrow [\![[\texttt{IF } F_0 \texttt{ THEN } F \texttt{ ELSE readsonly}]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}]\!](e)(s,s')$$

We assume

(2) $[\![\texttt{now.executes}]\!](e)(s,s')$

(3) $[\![\texttt{if (}E\texttt{) } C]\!](s,s')$

i.e. by the definition of $[\![\_]\!]$

(4) IF $[\![E]\!](s) =$ TRUE THEN $[\![C]\!](s,s')$ ELSE $s' = s$

By the definitions of $[\_]_\_$ and $[\![\_]\!]$, it suffices to show

(a.1) IF $[\![F_0]\!](e)(s,s')$ THEN $[\![F]\!](e)(s,s')$ ELSE $s = s'$

(a.2) $s = s'$ EXCEPT $I_1,\ldots,I_n$

(a.3) $\mathit{continues}(\mathit{control}(s')) \Rightarrow [\![F_c]\!](e)(s,s')$

(a.4) $\mathit{breaks}(\mathit{control}(s')) \Rightarrow [\![F_b]\!](e)(s,s')$

(a.5) $\mathit{returns}(\mathit{control}(s')) \Rightarrow [\![F_r]\!](e)(s,s')$

(a.6) $\mathit{throws}(\mathit{control}(s')) \Rightarrow \mathit{key}(\mathit{control}(s')) \in \{K_1,\ldots,K_m\}$

We proceed by case distinction:

- Case $[\![E]\!](s) =$ TRUE: from the case condition, the second premise and the definition of $\simeq$, we know

  (5) $[\![F_0]\!](e)(s,s')$

  From the case condition and (4), we know

  (6) $[\![C]\!](s,s')$

  From the first premise, we know

$$(7)\quad \begin{array}{l}[\![\texttt{now.executes}]\!](e)(s,s') \wedge [\![C]\!](s,s') \Rightarrow \\ \quad [\![[F]_{I_1,\ldots,I_n}^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}]\!](e)(s,s')\end{array}$$

From (2), (6), and (7), we know

$$(8)\quad [\![[F]_{I_1,\ldots,I_n}^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}]\!](e)(s,s')$$

From (5), (8), and the definitions of $[\llcorner]_{\llcorner}$ and $[\![\llcorner]\!]$, we know (a.1), (a.2), (a.3), (a.4), (a.5), and (a.6).

- Case $[\![E]\!](s) \neq \text{TRUE}$: from the case condition, the second premise and the definition of $\simeq$, we know

  $$(9)\quad \neg[\![F_0]\!](e)(s,s')$$

  From the case condition and (4), we know

  $$(10)\quad s' = s$$

  From (9) and (10), we know (a.1). From (10) and (REE), we know (a.2). From (2), (10), and the definition of $[\![\llcorner]\!]$, we know

  $$(11)\quad \mathit{executes}(\mathit{control}(s'))$$

  From (11) and Lemma "State Control Predicates", we know (a.3), (a.4), (a.5), and (a.6). □

5.3.11 Two-Sided Conditional

$$\frac{\begin{array}{l}C_1 : [F_1]_{I_1,\ldots,I_n}^{F_{c1},F_{b1},F_{r1},\{K_1,\ldots,K_m\}} \\ C_2 : [F_2]_{I_1,\ldots,I_n}^{F_{c2},F_{b2},F_{r2},\{L_1,\ldots,L_o\}} \\ E \simeq F_0\end{array}}{\begin{array}{l}\texttt{if (}E\texttt{)}\ C_1\ \texttt{else}\ C_2 : \\ \quad [\texttt{IF}\ F_0\ \texttt{THEN}\ F_1 \\ \qquad \texttt{ELSE}\ F_2]_{I_1,\ldots,I_n}^{F_{c1}\ \texttt{OR}\ F_{c2},F_{b1}\ \texttt{OR}\ F_{b2},F_{r1}\ \texttt{OR}\ F_{r2},\{K_1,\ldots,K_m,L_1,\ldots,L_o\}}\end{array}}$$

Soundness Proof We have to show

$$(a)\quad \begin{array}{l}[\![\texttt{now.executes}]\!](e)(s,s') \wedge [\![\texttt{if (}E\texttt{)}\ C_1\ \texttt{else}\ C_2]\!](s,s') \Rightarrow \\ \quad [\![[\texttt{IF}\ F_0\ \texttt{THEN}\ F_1 \\ \qquad \texttt{ELSE}\ F_2]_{I_1,\ldots,I_n}^{F_{c1}\ \texttt{OR}\ F_{c2},F_{b1}\ \texttt{OR}\ F_{b2},F_{r1}\ \texttt{OR}\ F_{r2},\{K_1,\ldots,K_m,L_1,\ldots,L_o\}} \\ \quad]\!](e)(s,s')\end{array}$$

We assume

(2) $[\![\texttt{now.executes}]\!](e)(s,s')$
(3) $[\![\texttt{if (}E\texttt{) }C_1\texttt{ else }C_2]\!](s,s')$

i.e. by the definition of $[\![\_]\!]$

(4) IF $[\![E]\!](s) =$ TRUE THEN $[\![C_1]\!](s,s')$ ELSE $[\![C_2]\!](s,s')$

By the definitions of $[\_]_\_$ and $[\![\_]\!]$, it suffices to show

(a.1) IF $[\![F_0]\!](e)(s,s')$ THEN $[\![F_1]\!](e)(s,s')$ ELSE $[\![F_2]\!](e)(s,s')$
(a.2) $s = s'$ EXCEPT $I_1,\ldots,I_n$
(a.3) $\mathit{continues}(\mathit{control}(s')) \Rightarrow [\![F_{c1}\ \texttt{OR}\ F_{c2}]\!](e)(s,s')$
(a.4) $\mathit{breaks}(\mathit{control}(s')) \Rightarrow [\![F_{b1}\ \texttt{OR}\ F_{b2}]\!](e)(s,s')$
(a.5) $\mathit{returns}(\mathit{control}(s')) \Rightarrow [\![F_{r1}\ \texttt{OR}\ F_{r2}]\!](e)(s,s')$
(a.6) $\mathit{throws}(\mathit{control}(s')) \Rightarrow \mathit{key}(\mathit{control}(s')) \in \{K_1,\ldots,K_m,L_1,\ldots,L_o\}$

We proceed by case distinction:

- Case $[\![E]\!](s) =$ TRUE: from the case condition, the third premise and the definition of $\simeq$, we know

  (5) $[\![F_0]\!](e)(s,s')$

  From the case condition and (4), we know

  (6) $[\![C_1]\!](s,s')$

  From the first premise, we know

  (7) $[\![\texttt{now.executes}]\!](e)(s,s') \wedge [\![C_1]\!](s,s') \Rightarrow [\![\,[F_1]_{I_1,\ldots,I_n}^{F_{c1},F_{b1},F_{r1},\{K_1,\ldots,K_m\}}\,]\!](e)(s,s')$

  From (2), (6), and (7), we know

  (8) $[\![\,[F_1]_{I_1,\ldots,I_n}^{F_{c1},F_{b1},F_{r1},\{K_1,\ldots,K_m\}}\,]\!](e)(s,s')$

  i.e. from the definitions of $[\![\_]\!]$ and $[\_]_\_$,

  (9) $[\![F_1]\!](e)(s,s')$
  (10) $s = s'$ EXCEPT $I_1,\ldots,I_n$
  (11) $\mathit{continues}(\mathit{control}(s')) \Rightarrow [\![F_{c1}]\!](e)(s,s')$
  (12) $\mathit{breaks}(\mathit{control}(s')) \Rightarrow [\![F_{b1}]\!](e)(s,s')$
  (13) $\mathit{returns}(\mathit{control}(s')) \Rightarrow [\![F_{r1}]\!](e)(s,s')$
  (14) $\mathit{throws}(\mathit{control}(s')) \Rightarrow \mathit{key}(\mathit{control}(s')) \in \{K_1,\ldots,K_m\}$

From (5) and (9), we know (a.1). From (10), we know (a.2). From (11) and the definition of $[\![\_]\!]$, we know (a.3). From (12) and the definition of $[\![\_]\!]$, we know (a.4). From (13) and the definition of $[\![\_]\!]$, we know (a.5). From (14) and the definition of $[\![\_]\!]$, we know (a.6).

- Case $[\![E]\!](s) \neq \text{TRUE}$: from the case condition, the third premise and the definition of $\simeq$, we know

  (15) $\neg[\![F_0]\!](e)(s,s')$

  From the case condition and (4), we know

  (16) $[\![C_2]\!](s,s')$

  From the second premise, we know

  (17) $[\![\texttt{now.executes}]\!](e)(s,s') \wedge [\![C_2]\!](s,s') \Rightarrow [\![[F_2]_{I_1,\ldots,I_n}^{F_{c2},F_{b2},F_{r2},\{L_1,\ldots,L_o\}}]\!](e)(s,s')$

  From (2), (16), and (17), we know

  (18) $[\![[F_2]_{I_1,\ldots,I_n}^{F_{c2},F_{b2},F_{r2},\{L_1,\ldots,L_o\}}]\!](e)(s,s')$

  i.e. from the definitions of $[\![\_]\!]$ and $[\,\_\,]_\_$,

  (19) $[\![F_2]\!](e)(s,s')$
  (20) $s = s'$ EXCEPT $I_1,\ldots,I_n$
  (21) $\mathit{continues}(\mathit{control}(s')) \Rightarrow [\![F_{c2}]\!](e)(s,s')$
  (22) $\mathit{breaks}(\mathit{control}(s')) \Rightarrow [\![F_{b2}]\!](e)(s,s')$
  (23) $\mathit{returns}(\mathit{control}(s')) \Rightarrow [\![F_{r2}]\!](e)(s,s')$
  (24) $\mathit{throws}(\mathit{control}(s')) \Rightarrow \mathit{key}(\mathit{control}(s')) \in \{L_1,\ldots,L_o\}$

  From (15) and (19), we know (a.1). From (20), we know (a.2). From (21) and the definition of $[\![\_]\!]$, we know (a.3). From (22) and the definition of $[\![\_]\!]$, we know (a.4). From (23) and the definition of $[\![\_]\!]$, we know (a.5). From (24) and the definition of $[\![\_]\!]$, we know (a.6). □

5.3.12 Continue Loop

$$\texttt{continue} : [\texttt{next.continues}]_\_^{\texttt{TRUE,FALSE,FALSE,}\emptyset}$$

Soundness Proof We have to show

(a) $[\![\texttt{now.executes}]\!](e)(s,s') \wedge [\![\texttt{continue}]\!](s,s') \Rightarrow [\![[\texttt{next.continues}]_\_^{\texttt{TRUE,FALSE,FALSE,}\emptyset}]\!](e)(s,s')$

We assume

(2) $[\![\texttt{now.executes}]\!](e)(s,s')$
(3) $[\![\texttt{continue}]\!](s,s')$

By the definitions of $[\![\_]\!]_\_$ and $[\![\_]\!]$, it suffices to show

(a.1) $\mathit{continues}(\mathit{control}(s'))$
(a.2) $s = s'$ EXCEPT $\_$
(a.3) $\mathit{continues}(\mathit{control}(s')) \Rightarrow$ TRUE
(a.4) $\mathit{breaks}(\mathit{control}(s')) \Rightarrow$ FALSE
(a.5) $\mathit{returns}(\mathit{control}(s')) \Rightarrow$ FALSE
(a.6) $\mathit{throws}(\mathit{control}(s')) \Rightarrow \mathit{key}(\mathit{control}(s')) \in \emptyset$

From (3) and the definition of $[\![\_]\!]$, we know

(4) $s' = \mathit{continue}(s)$

From (4) and (CD1), we know (a.1). From (4), (NEQ), and (CD2), we know (a.2). From (a.1) and Lemma "State Control Predicates", we know (a.3), (a.4), (a.5), and (a.6). □

5.3.13 Break Loop

$$\texttt{break} : [\texttt{next.breaks}]_\_^{\texttt{FALSE},\texttt{TRUE},\texttt{FALSE},\emptyset}$$

Soundness Proof We have to show

(a) $[\![\texttt{now.executes}]\!](e)(s,s') \wedge [\![\texttt{break}]\!](s,s') \Rightarrow$
$[\![[\texttt{next.breaks}]_\_^{\texttt{FALSE},\texttt{TRUE},\texttt{FALSE},\emptyset}]\!](e)(s,s')$

We assume

(2) $[\![\texttt{now.executes}]\!](e)(s,s')$
(3) $[\![\texttt{break}]\!](s,s')$

By the definitions of $[\![\_]\!]_\_$ and $[\![\_]\!]$, it suffices to show

(a.1) $\mathit{breaks}(\mathit{control}(s'))$

(a.2) $s = s'$ EXCEPT $\llcorner$

(a.3) $\mathit{continues}(\mathit{control}(s')) \Rightarrow$ FALSE

(a.4) $\mathit{breaks}(\mathit{control}(s')) \Rightarrow$ TRUE

(a.5) $\mathit{returns}(\mathit{control}(s')) \Rightarrow$ FALSE

(a.6) $\mathit{throws}(\mathit{control}(s')) \Rightarrow \mathit{key}(\mathit{control}(s')) \in \emptyset$

From (3) and the definition of $[\![\llcorner]\!]$, we know

(4) $s' = \mathit{break}(s)$

From (4) and (CD1), we know (a.1). From (4), (NEQ), and (CD2), we know (a.2). From (a.1) and Lemma "State Control Predicates", we know (a.3), (a.4), (a.5), and (a.6). □

5.3.14 Return Result

$$\frac{T \simeq E}{\begin{array}{l}\texttt{return}\ E: \\ \quad [\texttt{next.returns} \\ \quad \texttt{AND next.value=}T]_{\llcorner}^{\texttt{FALSE,FALSE,TRUE,}\emptyset}\end{array}}$$

Soundness Proof We have to show

(a) $[\![\texttt{now.executes}]\!](e)(s,s') \wedge [\![\texttt{return}\ E]\!](s,s') \Rightarrow$
$[\![[\texttt{next.returns}$
$\texttt{AND next.value=}T]_{\llcorner}^{\texttt{FALSE,FALSE,TRUE,}\emptyset}]\!](e)(s,s')$

We assume

(2) $[\![\texttt{now.executes}]\!](e)(s,s')$

(3) $[\![\texttt{return}\ E]\!](s,s')$

By the definitions of $[\llcorner]_{\llcorner}$ and $[\![\llcorner]\!]$, it suffices to show

(a.1) $\mathit{returns}(\mathit{control}(s'))$

(a.2) $\mathit{value}(\mathit{control}(s')) = [\![T]\!](e)(s,s')$

(a.3) $s = s'$ EXCEPT $\llcorner$

(a.4) $\mathit{continues}(\mathit{control}(s')) \Rightarrow$ FALSE

(a.5) $\mathit{breaks}(\mathit{control}(s')) \Rightarrow$ FALSE

(a.6) $\mathit{returns}(\mathit{control}(s')) \Rightarrow \text{TRUE}$

(a.7) $\mathit{throws}(\mathit{control}(s')) \Rightarrow \mathit{key}(\mathit{control}(s')) \in \emptyset$

From (3) and the definition of $[\![\_]\!]$, we know

(4) $s' = \mathit{return}(s, [\![E]\!](s))$

From (4) and (CD1), we know (a.1) and also

(5) $\mathit{value}(\mathit{control}(s')) = [\![E]\!](s)$

From (5), the premise, and the definition of $\simeq$, we know (a.2). From (4), (NEQ), and (CD2), we know (a.3). From (a.1) and Lemma "State Control Predicates", we know (a.4), (a.5), (a.6), and (a.7). □

5.3.15 Throw Exception

$$\frac{T \simeq E}{\begin{array}{l}\texttt{throw}\ I\ E: \\ \quad [\texttt{next.throws}\ I \\ \quad \texttt{AND next.value=}T]_{\_}^{\texttt{FALSE,FALSE,FALSE,}\{I\}}\end{array}}$$

Soundness Proof We have to show

(a) $$[\![\texttt{now.executes}]\!](e)(s,s') \wedge [\![\texttt{throw}\ I\ E]\!](s,s') \Rightarrow [\![[\texttt{next.throws}\ I \quad \texttt{AND next.value=}T]_{\_}^{\texttt{FALSE,FALSE,FALSE,}\{I\}}]\!](e)(s,s')$$

We assume

(2) $[\![\texttt{now.executes}]\!](e)(s,s')$

(3) $[\![\texttt{throw}\ I\ E]\!](s,s')$

By the definitions of $[\_]_{\_}$ and $[\![\_]\!]$, it suffices to show

(a.1) $\mathit{throws}(\mathit{control}(s'))$

(a.2) $\mathit{key}(\mathit{control}(s')) = I$

(a.3) $\mathit{value}(\mathit{control}(s')) = [\![T]\!](e)(s,s')$

(a.4) $s = s'$ EXCEPT $\_$

(a.5) $\mathit{continues}(\mathit{control}(s')) \Rightarrow \text{FALSE}$

(a.6) $breaks(control(s')) \Rightarrow \text{FALSE}$

(a.7) $returns(control(s')) \Rightarrow \text{FALSE}$

(a.8) $throws(control(s')) \Rightarrow key(control(s')) \in \{I\}$

From (3) and the definition of $[\![\_]\!]$, we know

(4) $s' = throw(s, I, [\![E]\!](s))$

From (4) and (CD1), we know (a.1), (a.2), (a.8), and also

(5) $value(control(s')) = [\![E]\!](s)$

From (5), the premise, and the definition of $\simeq$, we know (a.3). From (4), (NEQ), and (CD2), we know (a.4). From (a.1) and Lemma "State Control Predicates", we know (a.5), (a.6), and (a.7). □

5.3.16 Catch Exception

$C_1 : [F_1]^{F_{c1},F_{b1},F_{r1},\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}$
$C_2 : [F_2]^{F_{c2},F_{b2},F_{r2},\{L_1,\ldots,L_o\}}_{I_1,\ldots,I_n}$
$\$I_1, \ldots, \$I_n, \#I_s$ do not occur in F_1 and F_2
$I_a \neq I_b$
$\{I_a, I_b\} \cap \{I_1, \ldots, I_n\} = \emptyset$
$I_s \neq I_T$
$\$I_a, \$I_b, \#I_t$ do not occur in F_2

`try` C_1 `catch(`I_k I_v`)` C_2 :
  [`EXISTS` $\$I_1, \ldots, \$I_n$`:` `EXSTATE` $\#I_s$`:`
    $F_1[\#I_s/\texttt{next}][\$I_1/I_1', \ldots, \$I_n/I_n']$ `AND`
    `IF` $\#I_s$`.throws` I_k `THEN`
      `EXISTS` $\$I_a, \$I_b$`:` `EXSTATE` $\#I_t$`:`
        $\$I_a = \#I_s$`.value AND` $\#I_t$`.executes AND`
        $F_2[\#I_t/\texttt{now}][\$I_a/I_v][\$I_1/I_1, \ldots, \$I_n/I_n][\$I_b/I_v']$
    `ELSE`
      $I_1' = \$I_1$ `AND ... AND` $I_n' = \$I_n$ `AND next==`$\#I_s$
  $]^{F_{c1}\ \text{OR}\ F_{c2},F_{b1}\ \text{OR}\ F_{b2},F_{r1}\ \text{OR}\ F_{r2},(\{K_1,\ldots,K_m\}\setminus\{I_k\})\cup\{L_1,\ldots,L_o\}}_{I_1,\ldots,I_n}$

This rule is dual (in a generalized form) to the rule for command sequences: the second command C_2 is executed only if the first command C_1 raises an exception of the kind I_k specified in the `catch` clause: the execution of C_2 then takes place in a context where the local variable I_v is bound to the exception value.

For instance, the command

```
try
  if (x=0) throw DivByZero y else y=y/x
catch(DivByZero e)
  z=z+e
```

is specified by $[F]^{\texttt{FALSE,FALSE,FALSE,}\emptyset\}}_{\texttt{y,z}}$ where F is the formula

```
EXISTS $y, $z: EXSTATE $s:
    IF x=0 THEN
      $y=y AND $z=z AND
      #s.throws DivByZero AND #s.value = y
    ELSE
      $y=y/x AND $z=z AND #s.executes
  AND
    IF #s.throws DivByZero THEN
      EXISTS $a,$b: EXSTATE #t:
        $a=#s.value AND #t.executes AND
        y'=$y AND z'=$z+$a AND next.executes
    ELSE
      y'=$y AND z'=$z AND next==#s
```

which can be simplified to

```
IF x=0 THEN
  y'=y AND z'=z+y AND next.executes
ELSE
  y'=y/x AND z'=z AND next.executes
```

which succinctly describes the overall effect of the program snippet.

Soundness Proof We have to show

$$
\begin{aligned}
&[\![\texttt{now.executes}]\!](e)(s,s') \wedge [\![\texttt{try } C_1 \texttt{ catch } (I_k\ I_v)\ C_2]\!](s,s') \Rightarrow \\
&[\![\texttt{try } C_1 \texttt{ catch } (I_k\ I_v)\ C_2 : \\
&\quad [\texttt{EXISTS } \$I_1,\ldots,\$I_n\texttt{: EXSTATE } \#I_s\texttt{:} \\
&\qquad F_1[\#I_s/\texttt{next}][\$I_1/I_1\texttt{'},\ldots,\$I_n/I_n\texttt{'}]\ \texttt{AND} \\
&\qquad \texttt{IF } \#I_s\texttt{.throws } I_k \texttt{ THEN} \\
\text{(a)}\quad &\qquad\quad \texttt{EXISTS } \$I_a,\$I_b\texttt{: EXSTATE } \#I_t\texttt{:} \\
&\qquad\qquad \$I_a = \#I_s\texttt{.value AND } \#I_t\texttt{.executes AND} \\
&\qquad\qquad F_2[\#I_t/\texttt{now}][\$I_a/I_v][\$I_1/I_1,\ldots,\$I_n/I_n][\$I_b/I_v\texttt{'}] \\
&\qquad \texttt{ELSE} \\
&\qquad\quad I_1\texttt{'} = \$I_1 \texttt{ AND } \ldots \texttt{ AND } I_n\texttt{'} = \$I_n \texttt{ AND next==} \#I_s \\
&]^{F_{c1}\ \texttt{OR}\ F_{c2},F_{b1}\ \texttt{OR}\ F_{b2},F_{r1}\ \texttt{OR}\ F_{r2},(\{K_1,\ldots,K_m\}\setminus\{I_k\})\cup\{L_1,\ldots,L_o\}}_{I_1,\ldots,I_n} [\![(e)(s,s')
\end{aligned}
$$

We assume

(2) $[\![\texttt{now.executes}]\!](e)(s,s')$

(3) $[\![\texttt{try}\ C_1\ \texttt{catch}\,(I_k\ I_v)\ C_2]\!](s,s')$

i.e. by the definition of $[\![\_]\!]$ for some $s_0, s_1, s_2 \in \mathit{State}$

(4) $\mathit{executes}(\mathit{control}(s))$

(5) $[\![C_1]\!](s,s_0)$

(6)
$$\begin{array}{l}\text{IF } \mathit{throws}(\mathit{control}(s_0)) \wedge \mathit{key}(\mathit{control}(s_0)) = I_k \text{ THEN}\\ \quad s_1 = \mathit{write}(\mathit{execute}(s_0), I_v, \mathit{value}(\mathit{control}(s_0))) \wedge\\ \quad [\![C_2]\!](s_1,s_2) \wedge\\ \quad s' = \mathit{write}(s_2, I_v, \mathit{read}(s_0, I_v))\\ \text{ELSE } s' = s_0\end{array}$$

By the definitions of $[\_]_\_$ and $[\![\_]\!]$, it suffices to show

(a.1)
$$\begin{array}{l}\exists v_1,\ldots,v_n \in \mathit{Value}, c_s \in \mathit{Control}:\\ \quad \text{LET}\\ \quad\quad e_0 = e[I_s \mapsto c_s]_c,\\ \quad\quad e_1 = e_0[I_1 \mapsto v_1,\ldots,I_n \mapsto v_n]\\ \quad \text{IN}\\ \quad\quad [\![F_1[\#I_s/\texttt{next}][\$I_1/I_1\texttt{'},\ldots,\$I_n/I_n\texttt{'}]]\!](e_1)(s,s') \wedge\\ \quad\quad \text{IF } \mathit{throws}(e_1(I_s)) \wedge \mathit{key}(e_1(I_s)) = I_k \text{ THEN}\\ \quad\quad\quad [\![\texttt{EXISTS } \$I_a, \$I_b\texttt{:}\ \texttt{EXSTATE } \#I_t\texttt{:}\\ \quad\quad\quad\quad \$I_a = \#I_s\texttt{.value AND } \#I_t\texttt{.executes AND}\\ \quad\quad\quad\quad F_2[\#I_t/\texttt{now}][\$I_a/I_v][\$I_1/I_1,\ldots,\$I_n/I_n][\$I_b/I_v\texttt{'}]]\!]\\ \quad\quad\quad (e_1)(s,s')\\ \quad\quad \text{ELSE}\\ \quad\quad\quad \mathit{read}(s',I_1) = e_1(I_1) \wedge \ldots \wedge \mathit{read}(s',I_n) = e_1(I_n) \wedge\\ \quad\quad\quad \mathit{control}(s') = e_1(I_s)\end{array}$$

(a.2) $s = s'$ EXCEPT $I_1,\ldots,I_n$

(a.3) $\mathit{continues}(\mathit{control}(s')) \Rightarrow [\![F_{c1}\ \texttt{OR}\ F_{c2}]\!](e)(s,s')$

(a.4) $\mathit{breaks}(\mathit{control}(s')) \Rightarrow [\![F_{b1}\ \texttt{OR}\ F_{b2}]\!](e)(s,s')$

(a.5) $\mathit{returns}(\mathit{control}(s')) \Rightarrow [\![F_{r1}\ \texttt{OR}\ F_{r2}]\!](e)(s,s')$

(a.6) $\mathit{throws}(\mathit{control}(s')) \Rightarrow \mathit{key}(\mathit{control}(s')) \in \{K_1,\ldots,K_m\}\backslash\{I_k\}) \cup \{L_1,\ldots,L_o\}$

We define

(7) $e_0 := e[I_s \mapsto control(s_0)]_c$

(8) $e_1 := e_0[I_1 \mapsto read(s_0, I_1), \ldots, I_n \mapsto read(s_0, I_n)]$

To show (a.1), it suffices to show

(a.1.a.1) $[\![F_1[\#I_s/\texttt{next}][\$I_1/I_1', \ldots, \$I_n/I_n']]\!](e_1)(s,s')$

IF $throws(e_1(I_s)) \wedge key(e_1(I_s)) = I_k)$ THEN

$[\![$`EXISTS` $\$I_a, \$I_b$`:` `EXSTATE` $\#I_t$`:`

$\$I_a =$ $\#I_s$`.value AND` $\#I_t$`.executes AND`

(a.1.a.2) $F_2[\#I_t/\texttt{now}][\$I_a/I_v][\$I_1/I_1, \ldots, \$I_n/I_n][\$I_b/I_v']]\!](e_1)(s,s')$

ELSE

$read(s', I_1) = e_1(I_1) \wedge \ldots \wedge read(s', I_n) = e_1(I_n) \wedge$

$control(s') = e_1(I_s)$

From the first two premises, we know

(9) $[\![$`now.executes`$]\!](e)(s,s_0) \wedge [\![C_1]\!](s,s_0) \Rightarrow$
$[\![[F_1]_{I_1,\ldots,I_n}^{F_{c1},F_{b1},F_{r1},\{K_1,\ldots,K_m\}}]\!](e)(s,s_0)$

(10) $[\![$`now.executes`$]\!](e_1)(s_1,s_2) \wedge [\![C_2]\!](s_1,s_2) \Rightarrow$
$[\![[F_2]_{I_1,\ldots,I_n}^{F_{c2},F_{b2},F_{r2},\{L_1,\ldots,L_o\}}]\!](e_1)(s_1,s_2)$

From (2) and the definition of $[\![\_]\!]$, we know

(11) $[\![$`now.executes`$]\!](e)(s,s_0)$

From (5), (9), and (11), we know

(12) $[\![[F_1]_{I_1,\ldots,I_n}^{F_{c1},F_{b1},F_{r1},\{K_1,\ldots,K_m\}}]\!](e)(s,s_0)$

i.e. from the definitions of $[\_]_\_$ and $[\![\_]\!]$

(13) $[\![F_1]\!](e)(s,s_0)$

(14) $s = s_0$ EXCEPT $I_1, \ldots, I_n$

(15) $continues(control(s_0)) \Rightarrow [\![F_{c1}]\!](e)(s,s_0)$

(16) $breaks(control(s_0)) \Rightarrow [\![F_{b1}]\!](e)(s,s_0)$

(17) $returns(control(s_0)) \Rightarrow [\![F_{r1}]\!](e)(s,s_0)$

(18) $throws(control(s_0)) \Rightarrow key(control(s_0)) \in \{K_1, \ldots, K_m\}$

From the third, sixth, and seventh premise, we know

(19) $\$I_1, \ldots, \$I_n, \#I_s$ do not occur in F_1 and F_2

(19a) $I_s \neq I_T$

(20) $\$I_a, \$I_b, \#I_t$ do not occur in F_2

From (7), (13), (19), and (CNEF2), we know

(21) $[\![F_1[\#I_s/\texttt{next}]]\!](e_0)(s,s0)$

We proceed by case distinction:

- Case $throws(control(s_0)) \wedge key(control(s_0)) = I_k$: from the case condition and (6), we know

  (22) $s_1 = write(execute(s_0), I_v, value(control(s_0)))$
  (23) $[\![C_2]\!](s_1, s_2)$
  (24) $s' = write(s_2, I_v, read(s_0, I_v))$

  From (22), (CD1), and (CW), we know

  (25) $executes(control(s_1))$

  From (22), (CD2), (AVE), and (WS), we know

  (26) $s_1 = s_0$ EXCEPT I_v

  From (22), (RW1), and (CR), we know

  (27) $read(s_1, I_v) = value(control(s_0))$

  From (24) and (RW1), we know

  (28) $read(s', I_v) = read(s_0, I_v)$

  From (24) and (WS), we know

  (29) $s' = s_2$ EXCEPT I_v

  From (25) and the definition of $[\![\,\_\,]\!]$, we know

  (30) $[\![\texttt{now.executes}]\!](e)(s_1, s_2)$

  From (10), (23), and (30), we know

  (31) $[\![[F_2]^{F_{c2},F_{b2},F_{r2},\{L_1,\ldots,L_o\}}_{I_1,\ldots,I_n}]\!](e_1)(s_1, s_2)$

  i.e. from the definitions of $[\,\_\,]_{\_}$ and $[\![\,\_\,]\!]$

  (32) $[\![F_2]\!](e_1)(s_1, s_2)$
  (33) $s_1 = s_2$ EXCEPT $I_1, \ldots, I_n$
  (34) $continues(control(s_2)) \Rightarrow [\![F_{c2}]\!](e_1)(s_1, s_2)$

(35) $breaks(control(s_2)) \Rightarrow [\![F_{b2}]\!](e_1)(s_1,s_2)$
(36) $returns(control(s_2)) \Rightarrow [\![F_{r2}]\!](e_1)(s_1,s_2)$
(37) $throws(control(s_2)) \Rightarrow key(control(s_2)) \in \{L_1,\ldots,L_o\}$

From (26), (29), (33), (AVE), and (TRE), we know

(38) $s' = s_0$ EXCEPT $I_1,\ldots,I_n,I_v$

From (28), (38), and (RVE), we know

(39) $s' = s_0$ EXCEPT $I_1,\ldots,I_n$

From (7), (19), (21), (39), (CNEF0), and (PMVF2'), we know (a.1.a.1).

From the case condition, (7), and (8), we know

(40) $throws(e_1(I_s)) \wedge key(e_1(I_s)) = I_k$

From (40), to show (a.1.a.2), it suffices to show

(a.1.a.2.a) $[\![$`EXISTS` $\$I_a,\$I_b$`:` `EXSTATE` $\#I_t$`:`
$\$I_a =$ $\#I_s$`.value` `AND` $\#I_t$`.executes` `AND`
$F_2[\#I_t/\texttt{now}][\$I_a/I_v][\$I_1/I_1,\ldots,\$I_n/I_n][\$I_b/I_v{}']]](e_1)(s,s')$

We define

(41) $e_2 := e_1[I_t \mapsto control(s_1)]_c$
(42) $e_3 := e_2[I_a \mapsto read(s_1,I_v)]$
(43) $e_4 := e_3[I_b \mapsto read(s_2,I_v)]$

To show (a.1.a.2.a), it suffices to show

(a.1.a.2.a.1) $e_4(I_a) = value(e_4(I_s))$
(a.1.a.2.a.2) $executes(e_4(I_t))$
(a.1.a.2.a.3) $[\![F_2[\#I_t/\texttt{now}][\$I_a/I_v][\$I_1/I_1,\ldots,\$I_n/I_n][\$I_b/I_v{}']]](e_4)(s,s')$

From (7), (8), (19a), (27), (41), (42), and (43), we know (a.1.a.2.a.1).

From (25), (41), (42), and (43), we know (a.1.a.2.a.2).

From (19), (32), (41), and (CNOF2), we know

(44) $[\![F_2[\#I_t/\texttt{now}]]\!](e_2)(s_1,s_2)$

From (20), (26), (42), (44), (CNOF0) and (PMVF1'), we know

(45) $[\![F_2[\#I_t/\texttt{now}][\$I_a/I_v]]\!](e_3)(s_0,s_2)$

From (8), (41), (42), and the fifth premise we know

(46) $e_3 = e_3[I_1 \mapsto read(s_0,I_1),\ldots,I_n \mapsto read(s_0,I_n)]$

From (14), (19), (20), (45), (46), the fifth premise, (CNOF0), and finally (PMVF1'), we know

(47) $[\![F_2[\#I_t/\mathtt{now}][\$I_a/I_v][\$I_1/I_1,\ldots,\$I_n/I_n]]\!](e_3)(s,s_2)$

From (19), (20), (29), (43), (47), the fourth premise, (CNEF0), and finally (PMVF2'), we know (a.1.a.2.a.3).

From (14), (39), and (TRE), we know (a.2).

From (24) and (CW), we know

(48) $control(s') = control(s_2)$

From (34), (48), the second premise, Lemma "Constant Formulas", and the definition of $[\![\_]\!]$, we know (a.3). From (35), (48), the second premise, Lemma "Constant Formulas", and the definition of $[\![\_]\!]$, we know (a.4). From (36), (48), the second premise, Lemma "Constant Formulas", and the definition of $[\![\_]\!]$, we know (a.5). From (37), (48) and the definition of $[\![\_]\!]$, we know (a.6).

- Case $\neg(throws(control(s_0)) \wedge key(control(s_0)) = I_k)$: from the case condition and (6), we know

  (49) $s' = s_0$

  and thus from (REE)

  (50) $s_0 = s' \text{ EXCEPT } I_1,\ldots,I_n$

  From (8), (19), (21), (50), (CNEF0), and (PMVF2'), we know (a.1.a.1).

  From the case condition, (7), and (8), we know

  (33) $\neg(throws(control(e_1(I_s))) \wedge key(control(e_1(I_s))) = I_k)$

  From (33), to show (a.1.a.2), it suffices to show

  (a.1.a.2.a) $read(s',I_1) = e_1(I_1) \wedge \ldots \wedge read(s',I_n) = e_1(I_n) \wedge control(s') = e_1(I_s)$

  From (7), (8) and (49), we have (a.1.a.2.a).

  From (14), (50), and (TRE), we know (a.2). From (15), (49), and the definition of $[\![\_]\!]$, we know (a.3). From (16), (49), and the definition of $[\![\_]\!]$, we know (a.4). From (17), (49), and the definition of $[\![\_]\!]$, we know (a.5). From the case condition, (18), (49), and the definition of $[\![\_]\!]$, we know (a.6). □

5.4 Loops and Interruptions

Taking into account control flow interruptions, the semantics of the command `while(`E`)` C is redefined as shown in Figure 5.11 by making use of two state sequences $t(0) = s, t(1), \ldots, t(k)$ and $u(0) = s, u(1), \ldots, u(k)$:

- $t(i)$ is the state before the $(i+1)$-th execution of the loop body C, $u(i+1)$ is the state after that execution.
- If $u(i+1)$ is "continuing" or "breaking" then $t(i+1)$ is a variant of $u(i+1)$ which is set to "executing". Otherwise $t(i+1)$ is identical to $u(i+1)$.
- If $u(i)$ is "breaking", or "returning", or "throwing", then i is k, i.e. the loop is terminated after that state; (only) in that case the loop condition E may be true on $t(k)$.

As a consequence of these rules, a loop can be terminated in a poststate that is "executing" (if the execution of the loop body has yielded a state that is "breaking" or makes the loop condition E "false") or "returning" or "throwing" (if the execution of the loop body has yielded such a state); it cannot be terminated in a "continuing" or a "breaking" state.

Furthermore, the notion of an invariant is generalized to take into account that a specification formula may refer via `now` and `next` to the control data of the prestate and of the poststate, respectively. The basic idea is that an invariant describes the relationship between the initial state $u(0) = s$ and the state $u(k)$ immediately after the k-th execution of the loop body (please note that this is different from the state $t(k)$ immediately before the $k+1$-th iteration):

- A state variable $\#I_s$ is introduced to capture the control data of state $u(k)$ whose program variables carry the values denoted by the mathematical variables $\$I_1, \ldots, \$I_n$ (substitutions of occurrences of the poststate variables $I_1\texttt{'}, \ldots, I_n\texttt{'}$ in the formula by the mathematical variables are mirrored by a substitution of `next` by $\#I_s$); to allow another loop iteration, $\#I_s$ must be continuing or executing.
- A state variable $\#I_t$ is introduced to capture the control data of state $t(k)$ where $t(k)$ must be "executing" to allow another loop iteration (both $u(k)$ and $t(k)$) hold the same values in the program variables which are denoted by $I_1\texttt{'}, \ldots, I_n\texttt{'}$: $\#I_s$ and $\#I_t$ are related in that, if $\#I_s$ is "executing" or "continuing", then $\#I_t$ is "executing", otherwise $\#I_t$ equals $\#I_s$.

Verification Calculus with Interruptions and Loops
A variation of the language of Figure 5.3.

Semantic Operations

$$
\begin{array}{l}
\mathit{finiteExecution}: \\
\mathbb{P}(\mathbb{N} \times \mathit{State}^{\infty} \times \mathit{State}^{\infty} \times \mathit{State} \times \mathit{StateFunction} \times \mathit{StateRelation}) \\
\mathit{finiteExecution}(k,t,u,s,E,C) \Leftrightarrow \\
\quad t(0) = s \wedge u(0) = s \wedge \\
\quad \forall i \in \mathbb{N}_k : \\
\qquad \neg \mathit{breaks}(\mathit{control}(u(i))) \wedge \mathit{executes}(\mathit{control}(t(i))) \wedge \\
\qquad E(t(i)) = \text{TRUE} \wedge C(t(i), u(i+1)) \wedge \\
\qquad \text{IF } \mathit{continues}(\mathit{control}(u(i+1))) \vee \mathit{breaks}(\mathit{control}(u(i+1))) \\
\qquad\quad \text{THEN } t(i+1) = \mathit{execute}(u(i+1)) \\
\qquad\quad \text{ELSE } t(i+1) = u(i+1)
\end{array}
$$

Valuation Function

$$
\begin{array}{l}
[\![\texttt{while (}E\texttt{) } C]\!](s,s') \Leftrightarrow \\
\quad \exists k \in \mathbb{N}, t, u \in \mathit{State}^{\infty} : \\
\qquad \mathit{finiteExecution}(k,t,u,s,[\![E]\!],[\![C]\!]) \wedge \\
\qquad ([\![E]\!](t(k)) \neq \text{TRUE} \vee \\
\qquad\quad \neg(\mathit{executes}(\mathit{control}(u(k))) \vee \\
\qquad\qquad \mathit{continues}(\mathit{control}(u(k))))) \wedge \\
\qquad t(k) = s'
\end{array}
$$

Definitions

$$
\begin{array}{l}
\mathit{Invariant}(G,H,F)_{I_1,\ldots,I_n} \equiv \\
\quad G \text{ has no free (mathematical or state) variables} \wedge \\
\quad \$I_1, \ldots, \$I_n, \#I_s, \#I_t \text{ do not occur in } G, H, \text{ and } F \wedge \\
\quad \forall e \in \mathit{ValueEnv}, s, s' \in \mathit{Store} : \\
\qquad [\![\texttt{FORALL } \$I_1, \ldots, \$I_n \texttt{: ALLSTATE } \#I_s\texttt{, } \#I_t \texttt{:} \\
\qquad\quad (G[\#I_s/\texttt{next}][\$I_1/I_1\texttt{'}, \ldots, \$I_n/I_n\texttt{'}] \\
\qquad\qquad \texttt{AND } (\#I_s\texttt{.executes OR } \#I_s\texttt{.continues}) \\
\qquad\qquad \texttt{AND } \#I_t\texttt{.executes} \\
\qquad\qquad \texttt{AND } H[\#I_t/\texttt{now}][\$I_1/I_1, \ldots, \$I_n/I_n] \\
\qquad\qquad \texttt{AND } F[\#I_t/\texttt{now}][\$I_1/I_1, \ldots, \$I_n/I_n]) \\
\qquad\quad \texttt{=> } G)]\!](e)(s,s')
\end{array}
$$

Figure 5.11: The Language with Interruptions and Loops (Part 1 of 3)

Basic Rules

$$\frac{C : [F]^{F_c,F_b,F_r;K_1,\ldots,K_m}_{I_1,\ldots,I_n}}{\begin{array}{l}\texttt{while (}E\texttt{) }C\texttt{:}\\ \quad [\texttt{!next.continues AND}\\ \qquad \texttt{!next.breaks}]^{\texttt{FALSE,FALSE},F_r;K_1,\ldots,K_m}_{I_1,\ldots,I_n}\end{array}}$$

$$\frac{\begin{array}{l}C : [F]^{F_c,\texttt{FALSE},F_r;K_1,\ldots,K_m}_{I_1,\ldots,I_n}\\ E \simeq H\end{array}}{\begin{array}{l}\texttt{while (}E\texttt{) }C\texttt{:}\\ \quad [\texttt{!next.continues AND !next.breaks AND}\\ \quad (\texttt{next.executes =>}\\ \qquad !H[\texttt{next}/\texttt{now}][I_1\texttt{'}/I_1,\ldots,I_n\texttt{'}/I_n])\\ \quad]^{\texttt{FALSE,FALSE},F_r;K_1,\ldots,K_m}_{I_1,\ldots,I_n}\end{array}}$$

Figure 5.12: The Language with Interruptions and Loops (Part 2 of 3)

Figures 5.12 and 5.13 show that the two verification rules for loops (with and without invariant) now come in two flavors:

- The general form takes into account that the loop by may terminated by an execution of the `break` statement in the loop body C (rather than by the loop condition E becoming "false"); we thus cannot say anything about the value of E in the poststate of the loop.
- A more special form can be applied if the loop body C does not involve a `break` statement (as indicated by the specification of C); in this case, if the loop yields an "executing" poststate (rather than a "returning" or a "throwing" one), the loop condition E (respectively its mathematical counterpart H) does not hold in that state.

The proofs of the soundness of these rules follow the basic structure of the soundness proofs for programs without control flow interruptions; nevertheless, they become substantially more complicated in detail.

Invariant Rules

$C : [F]^{F_c,F_b,F_r;K_1,\ldots,K_m}_{I_1,\ldots,I_n}$
$Invariant(G,H,F)_{I_1,\ldots,I_n}$
$\#I_s$ does not occur in G

```
while (E) C :
   [!next.continues AND !next.breaks AND
   (G[now/next][I1/I1',...,In/In'] =>
      EXISTS #Is : G[#Is/next] AND
         IF #Is.continues OR #Is.breaks
            THEN next.executes
            ELSE next == #Is)
   ]^{FALSE,FALSE,Fr;K1,...,Km}_{I1,...,In}
```

$C : [F]^{F_c,\mathtt{FALSE},F_r;K_1,\ldots,K_m}_{I_1,\ldots,I_n}$
$E \simeq H$
$Invariant(G,H,F)_{I_1,\ldots,I_n}$
$\#I_s$ does not occur in G

```
while (E) C :
   [!next.continues AND !next.breaks AND
   (next.executes =>
      !H[next/now][I1'/I1,...,In'/In]) AND
   (G[now/next][I1/I1',...,In/In'] =>
      EXISTS #Is : G[#Is/next] AND
         IF #Is.continues OR #Is.breaks
            THEN next.executes
            ELSE next == #Is)
   ]^{FALSE,FALSE,Fr;K1,...,Km}_{I1,...,In}
```

Figure 5.13: The Language with Interruptions and Loops (Part 3 of 3)

5.4.1 Basic Rule (With Breaks)

$$\frac{C : [F]^{F_c,F_b,F_r;K_1,\ldots,K_m}_{I_1,\ldots,I_n}}{\begin{array}{l}\texttt{while (}E\texttt{) }C\texttt{:}\\ \quad [\texttt{!next.continues}\\ \qquad \texttt{AND !next.breaks}]^{\texttt{FALSE,FALSE,}F_r;K_1,\ldots,K_m}_{I_1,\ldots,I_n}\end{array}}$$

Soundness Proof We have to show

(a) $[\![\texttt{now.executes}]\!](e)(s,s') \wedge [\![\texttt{while (}E\texttt{) }C]\!](s,s') \Rightarrow$
$[\![[\texttt{!next.continues}$
$\texttt{AND !next.breaks}]^{\texttt{FALSE,FALSE,}F_r;K_1,\ldots,K_m}_{I_1,\ldots,I_n}]\!](e)(s,s')$

We assume

(2) $[\![\texttt{now.executes}]\!](e)(s,s')$

(3) $[\![\texttt{while (}E\texttt{) }C]\!](s,s')$

By the definitions of $[\_]_\_$ and $[\![\_]\!]$, it suffices to show

(a.1) $\neg \mathit{continues}(\mathit{control}(s'))$

(a.2) $\neg \mathit{breaks}(\mathit{control}(s'))$

(a.3) $s = s'$ EXCEPT $I_1,\ldots,I_n$

(a.4) $\mathit{continues}(\mathit{control}(s')) \Rightarrow$ FALSE

(a.5) $\mathit{breaks}(\mathit{control}(s')) \Rightarrow$ FALSE

(a.6) $\mathit{returns}(\mathit{control}(s')) \Rightarrow [\![F_r]\!](e)(s,s')$

(a.7) $\mathit{throws}(\mathit{control}(s')) \Rightarrow \mathit{key}(\mathit{control}(s')) \in \{K_1,\ldots,K_m\}$

From (2), (3), and the definition of $[\![\_]\!]$, we know for some $k \in \mathbb{N}, t, u \in \mathit{State}^\infty$

(4) $\mathit{executes}(\mathit{control}(s))$

(5) $t(0) = s$

(6) $u(0) = s$

(7)
$\forall i \in \mathbb{N}_k :$
$\quad \neg \mathit{breaks}(\mathit{control}(u(i))) \wedge \mathit{executes}(\mathit{control}(t(i))) \wedge$
$\quad [\![E]\!](t(i)) = \text{TRUE} \wedge [\![C]\!](t(i),u(i+1)) \wedge$
$\quad$ IF $\mathit{continues}(\mathit{control}(u(i+1))) \vee \mathit{breaks}(\mathit{control}(u(i+1)))$
$\qquad$ THEN $t(i+1) = \mathit{execute}(u(i+1))$
$\qquad$ ELSE $t(i+1) = u(i+1)$

$$(8)\quad \begin{array}{l}[\![E]\!](t(k)) \neq \text{TRUE} \vee \\ \quad \neg(\mathit{executes}(\mathit{control}(u(k))) \vee \mathit{continues}(\mathit{control}(u(k))))\end{array}$$

$$(9)\quad t(k) = s'$$

From the premise and the definition of $[\![\_]\!]$, we know

$$(10)\quad \begin{array}{l}\forall s, s' \in \mathit{State}, e \in \mathit{Environment}: \\ \quad \mathit{executes}(\mathit{control}(s)) \wedge [\![C]\!](s, s') \Rightarrow \\ \quad\quad [\![[F]_{I_1,\ldots,I_n}^{F_c,F_b,F_r;K_1,\ldots,K_m}]\!](e)(s, s')\end{array}$$

From (7), (10), and the definition of $[\![\_]\!]$, we know

$$(11)\quad \begin{array}{l}\forall i \in \mathbb{N}_k: \\ \quad \neg\mathit{breaks}(\mathit{control}(u(i))) \wedge \mathit{executes}(\mathit{control}(t(i))) \wedge \\ \quad [\![E]\!](t(i)) = \text{TRUE} \wedge [\![F]\!](e)(t(i), u(i)+1) \wedge \\ \quad t(i) = u(i+1) \text{ EXCEPT } I_1, \ldots, I_n \wedge \\ \quad (\mathit{continues}(\mathit{control}(u(i+1))) \Rightarrow [\![F_c]\!](e)(t(i), u(i+1))) \wedge \\ \quad (\mathit{breaks}(\mathit{control}(u(i+1))) \Rightarrow [\![F_b]\!](e)(t(i), u(i+1))) \wedge \\ \quad (\mathit{returns}(\mathit{control}(u(i+1))) \Rightarrow [\![F_r]\!](e)(t(i), u(i+1))) \wedge \\ \quad (\mathit{throws}(\mathit{control}(u(i+1))) \Rightarrow \\ \quad\quad \mathit{key}(\mathit{control}(u(i+1))) \in \{K_1, \ldots, K_m\}) \wedge \\ \quad \text{IF } \mathit{continues}(\mathit{control}(u(i+1))) \vee \mathit{breaks}(\mathit{control}(u(i+1))) \\ \quad\quad \text{THEN } t(i+1) = \mathit{execute}(u(i+1)) \\ \quad\quad \text{ELSE } t(i+1) = u(i+1)\end{array}$$

From (11) and Lemma "State Control Predicates", we can conclude

$$(12)\quad \begin{array}{l}k > 0 \Rightarrow \\ \quad t(k) = \mathit{execute}(u(k)) \vee \\ \quad (\mathit{executes}(\mathit{control}(t(k))) \vee \\ \quad (\mathit{returns}(\mathit{control}(t(k))) \wedge [\![F_r]\!](e)(t(k-1), t(k)))) \vee \\ \quad (\mathit{throws}(\mathit{control}(t(k))) \wedge \mathit{key}(\mathit{control}(t(k)))) \in \{K_1, \ldots, K_m\}))\end{array}$$

From (4) and (5), we know

$$(13)\quad \mathit{executes}(\mathit{control}(t(0)))$$

From (9), (12), (13), (CD1), Lemma "Constant Formulas", and Lemma "State Control Predicates", we know (a.1), (a.2), (a.4), (a.5), (a.6), and (a.7).

To prove (a.3), we proceed by case distinction:

- Case $k = 0$: from the case condition, (5), (9), and (REE), we have (a.3).

- Case $k > 0$: from (11), we can conclude

$$\begin{array}{ll} & \forall i \in \mathbb{N}_k : \\ (14) & \quad t(i) = u(i+1) \text{ EXCEPT } I_1, \ldots, I_n \wedge \\ & \quad (t(i+1) = u(i+1) \vee t(i+1) = \mathit{execute}(u(i+1))) \end{array}$$

From (14), (CD2), (NEQ), and (AVE), we know

$$(15) \quad \forall i \in \mathbb{N}_k : t(i) = t(i+1) \text{ EXCEPT } I_1, \ldots, I_n$$

From (5), (9), (15), and (TRE), we know (a.3). □

5.4.2 Basic Rule (Without Breaks)

$$\frac{\begin{array}{l} C : [F]^{F_c,\texttt{FALSE},F_r;K_1,\ldots,K_m}_{I_1,\ldots,I_n} \\ E \simeq H \end{array}}{\begin{array}{l} \texttt{while (}E\texttt{) }C\texttt{:} \\ \quad [\texttt{!next.continues AND !next.breaks AND} \\ \quad (\texttt{next.executes =>} \\ \quad\quad !H[\texttt{next/now}][I_1\texttt{'}/I_1, \ldots, I_n\texttt{'}/I_n]) \\ \quad]^{\texttt{FALSE,FALSE},F_r;K_1,\ldots,K_m}_{I_1,\ldots,I_n} \end{array}}$$

Soundness Proof We have to show

$$\begin{array}{ll} & [\![\texttt{now.executes}]\!](e)(s,s') \wedge [\![\texttt{while (}E\texttt{) }C]\!](s,s') \Rightarrow \\ & [\![[\texttt{!next.continues AND !next.breaks AND} \\ (a) & \quad (\texttt{next.executes =>} \\ & \quad\quad !H[\texttt{next/now}][I_1\texttt{'}/I_1, \ldots, I_n\texttt{'}/I_n]) \\ & \quad]^{\texttt{FALSE,FALSE},F_r;K_1,\ldots,K_m}_{I_1,\ldots,I_n}]\!](e)(s,s') \end{array}$$

We assume

$$(2) \quad [\![\texttt{now.executes}]\!](e)(s,s')$$

$$(3) \quad [\![\texttt{while (}E\texttt{) }C]\!](s,s')$$

By the definitions of $[\_]_\_$ and $[\![\_]\!]$, it suffices to show

$$(a.1) \quad \neg \mathit{continues}(\mathit{control}(s'))$$

$$(a.2) \quad \neg \mathit{breaks}(\mathit{control}(s'))$$

$$\begin{array}{ll} (a.3) & \mathit{executes}(\mathit{control}(s')) \Rightarrow \\ & \quad \neg [\![H[\texttt{next/now}][I_1\texttt{'}/I_1, \ldots, I_n\texttt{'}/I_n]]\!](e)(s,s') \end{array}$$

(a.4) $s = s'$ EXCEPT $I_1, \ldots, I_n$

(a.5) $\mathit{continues}(\mathit{control}(s')) \Rightarrow$ FALSE

(a.6) $\mathit{breaks}(\mathit{control}(s')) \Rightarrow$ FALSE

(a.7) $\mathit{returns}(\mathit{control}(s')) \Rightarrow [\![F_r]\!](e)(s, s')$

(a.8) $\mathit{throws}(\mathit{control}(s')) \Rightarrow \mathit{key}(\mathit{control}(s')) \in \{K_1, \ldots, K_m\}$

The proofs of all but (a.3) proceed as shown in the proof of the basic rule with breaks. To show (a.3), we assume

(4) $\mathit{executes}(\mathit{control}(s'))$

and show

(a.3.a) $\neg[\![H[\mathtt{next}/\mathtt{now}][I_1\mathtt{'}/I_1, \ldots, I_n\mathtt{'}/I_n]]\!](e)(s, s')$

From (2), (3), and the definition of $[\![\_]\!]$, we know for some $k \in \mathbb{N}, t, u \in \mathit{State}^\infty$

(5) $\mathit{executes}(\mathit{control}(s))$

(6) $t(0) = s$

(7) $u(0) = s$

(8)
$$\begin{array}{l} \forall i \in \mathbb{N}_k : \\ \quad \neg \mathit{breaks}(\mathit{control}(u(i))) \wedge \mathit{executes}(\mathit{control}(t(i))) \wedge \\ \quad [\![E]\!](t(i)) = \text{TRUE} \wedge [\![C]\!](t(i), u(i+1)) \wedge \\ \quad \text{IF } \mathit{continues}(\mathit{control}(u(i+1))) \vee \mathit{breaks}(\mathit{control}(u(i+1))) \\ \qquad \text{THEN } t(i+1) = \mathit{execute}(u(i+1)) \\ \qquad \text{ELSE } t(i+1) = u(i+1) \end{array}$$

(9)
$$\begin{array}{l} [\![E]\!](t(k)) \neq \text{TRUE} \vee \\ \quad \neg(\mathit{executes}(\mathit{control}(u(k))) \vee \mathit{continues}(\mathit{control}(u(k)))) \end{array}$$

(10) $t(k) = s'$

From the premises and the definition of $[\![\_]\!]$, we also know

(11)
$$\begin{array}{l} \forall s, s' \in \mathit{State}, e \in \mathit{Environment} : \\ \quad \mathit{executes}(\mathit{control}(s)) \wedge [\![C]\!](s, s') \Rightarrow \\ \qquad [\![[F]_{I_1,\ldots,I_n}^{F_c, \mathtt{FALSE}, F_r; K_1, \ldots, K_m}]\!](e)(s, s') \end{array}$$

(12) $E \simeq H$

From (8), (11), and the definition of $[\![\_]\!]$, we know

$$
\begin{array}{l}
\forall i \in \mathbb{N}_k : \\
\quad \neg \mathit{breaks}(\mathit{control}(u(i)))) \wedge \mathit{executes}(\mathit{control}(t(i))) \wedge \\
\quad [\![E]\!](t(i)) = \text{ TRUE} \wedge [\![F]\!](e)(t(i), u(i+1)) \wedge \\
\quad t(i) = u(i+1) \text{ EXCEPT } I_1, \ldots, I_n \wedge \\
\quad (\mathit{continues}(\mathit{control}(u(i+1))) \Rightarrow [\![F_c]\!](e)(t(i), u(i+1))) \wedge \\
\quad \neg \mathit{breaks}(\mathit{control}(u(i+1))) \wedge \\
\quad (\mathit{returns}(\mathit{control}(u(i+1))) \Rightarrow [\![F_r]\!](e)(t(i), u(i+1))) \wedge \\
\quad (\mathit{throws}(\mathit{control}(u(i+1))) \Rightarrow \\
\qquad \mathit{key}(\mathit{control}(u(i+1))) \in \{K_1, \ldots, K_m\}) \wedge \\
\quad \text{IF } \mathit{continues}(\mathit{control}(u(i+1))) \vee \mathit{breaks}(\mathit{control}(u(i+1))) \\
\qquad \text{THEN } t(i+1) = \mathit{execute}(u(i+1)) \\
\qquad \text{ELSE } t(i+1) = u(i+1)
\end{array} \tag{13}
$$

We now show as an intermediate goal

(a.3.b) $[\![E]\!](t(k)) \neq \text{TRUE}$

From (9), it suffices to assume

(14) $\neg \mathit{executes}(\mathit{control}(u(k)))$

(15) $\neg \mathit{continues}(\mathit{control}(u(k))))$

and show a contradiction.

From (5) and (7), we know

(16) $\mathit{executes}(\mathit{control}(u(0)))$

From (16) and Lemma “State Control Predicates”, we know

(17) $\neg \mathit{breaks}(\mathit{control}(u(0)))$

(18) $\neg \mathit{returns}(\mathit{control}(u(0)))$

(19) $\neg \mathit{throws}(\mathit{control}(u(0)))$

From (13) and (17), we can conclude

(20) $\neg \mathit{breaks}(\mathit{control}(u(k)))$

From (4) and (10), we know

(21) $\mathit{executes}(\mathit{control}(t(k))$

From (13), (18), (19), and (21), we know

(22) $\neg returns(control(u(k)))$

(23) $\neg throws(control(u(k)))$

But (14), (15), (20), (22), and (23) contradicts Lemma "State Control Predicates".

We now proceed with the proof of (a.3.a). From (12), (a.3.b), and the definition of $\simeq$, we know

(24) $\neg[\![H]\!](e)(s',s')$

From (24), (CD0), and (PNNF1), we know

(25) $\neg[\![H[\texttt{next/now}]]\!](e)(s',s')$

We define

(26) $s'' := writes(s',I_1,read(s',I_1),\ldots,s,I_n,read(s',I_n))$

From (26) and (WSE), we know

(27) $s'' = s'$ EXCEPT $I_1,\ldots,I_n$

From *DifferentVariables*, (26), and (RWE), we know

(28) $read(s'',I_1) = read(s',I_1) \wedge \ldots read(s'',I_n) = read(s',I_n)$

From (27), (28), (RVE), and (NEQ), we know

(29) s'' EQUALS s'

From (26) and (CWE), we know

(30) $control(s'') = control(s')$

From (25), (29), (30), (REE), and (ESF'), we know

(31) $\neg[\![H[\texttt{next/now}]]\!](e)(s'',s')$

From (26), (31), and (PPVF1'), we know

(32) $\neg[\![H[\texttt{next/now}][I_1\texttt{'}/I_1,\ldots,I_n\texttt{'}/I_n]]\!](e)(s',s')$

From (32), (CNOF1), and (PVFNO), we know

(33) $\neg[\![H[\texttt{next/now}][I_1\texttt{'}/I_1,\ldots,I_n\texttt{'}/I_n]]\!](e)((store(s'),control(s)),s')$

From (CD3), (NEQ), and (AVE), we know

(34) $s' = (store(s'),control(s))$ EXCEPT $I_1,\ldots,I_n$

From (a.4), (34), and (TRE), we know

(35) $s = (store(s'),control(s))$ EXCEPT $I_1,\ldots,I_n$

From (29), (35), (PPVG0'), and (PVF3'), we have (a.3.a). □

5.4.3 Invariant Rule (With Breaks)

$$\frac{\begin{array}{l} C : [F]^{F_c,F_b,F_r;K_1,\ldots,K_m}_{I_1,\ldots,I_n} \\ \mathit{Invariant}(G,H,F)_{I_1,\ldots,I_n} \\ \#I_s \text{ does not occur in } G \end{array}}{\begin{array}{l} \texttt{while (}E\texttt{) } C\texttt{:} \\ \quad [\texttt{!next.continues AND !next.breaks AND} \\ \quad (G[\texttt{now/next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}] \texttt{ =>} \\ \qquad \texttt{EXISTS } \#I_s : G[\#I_s/\texttt{next}] \texttt{ AND} \\ \qquad\quad \texttt{IF } \#I_s\texttt{.continues OR } \#I_s\texttt{.breaks} \\ \qquad\qquad \texttt{THEN next.executes} \\ \qquad\qquad \texttt{ELSE next == } \#I_s) \\ \quad]^{\texttt{FALSE,FALSE},F_r;K_1,\ldots,K_m}_{I_1,\ldots,I_n} \end{array}}$$

Soundness Proof We have to show

$$\begin{array}{ll} & [\![\texttt{now.executes}]\!](e)(s,s') \wedge [\![\texttt{while (}E\texttt{) } C]\!](s,s') \Rightarrow \\ & [\![[\texttt{!next.continues AND !next.breaks AND} \\ & \quad (G[\texttt{now/next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}] \texttt{ =>} \\ & \qquad \texttt{EXISTS } \#I_s : G[\#I_s/\texttt{next}] \texttt{ AND} \\ (a) & \qquad\quad \texttt{IF } \#I_s\texttt{.continues OR } \#I_s\texttt{.breaks} \\ & \qquad\qquad \texttt{THEN next.executes} \\ & \qquad\qquad \texttt{ELSE next == } \#I_s) \\ &]^{\texttt{FALSE,FALSE},F_r;K_1,\ldots,K_m}_{I_1,\ldots,I_n}]\!](e)(s,s') \end{array}$$

We assume

(2) $[\![\texttt{now.executes}]\!](e)(s,s')$

(3) $[\![\texttt{while (}E\texttt{) } C]\!](s,s')$

By the definitions of $[\_]_{\_}$ and $[\![\_]\!]$, it suffices to show

(a.1) $\neg \mathit{continues}(\mathit{control}(s'))$

(a.2) $\neg \mathit{breaks}(\mathit{control}(s'))$

(a.3)
$$\begin{array}{l} [\![G[\texttt{now/next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]]\!](e)(s,s') \Rightarrow \\ [\![\texttt{EXISTS } \#I_s : G[\#I_s/\texttt{next}] \texttt{ AND} \\ \quad \texttt{IF } \#I_s\texttt{.continues OR } \#I_s\texttt{.breaks} \\ \qquad \texttt{THEN next.executes} \\ \qquad \texttt{ELSE next == } \#I_s]\!](e)(s,s') \end{array}$$

(a.4) $s = s'$ EXCEPT $I_1,\ldots,I_n$

(a.5) $\mathit{continues}(\mathit{control}(s')) \Rightarrow \text{FALSE}$

(a.6) $\mathit{breaks}(\mathit{control}(s')) \Rightarrow \text{FALSE}$

(a.7) $\mathit{returns}(\mathit{control}(s')) \Rightarrow [\![F_r]\!](e)(s,s')$

(a.8) $\mathit{throws}(\mathit{control}(s')) \Rightarrow \mathit{key}(\mathit{control}(s')) \in \{K_1,\ldots,K_m\}$

The proofs of all parts except for (a.3) proceed as shown in the soundness proof of the basic rule with breaks.

To show (a.3), we assume

(4) $[\![G[\texttt{now}/\texttt{next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]]\!](e)(s,s')$

and show

(a.3.a)
$$[\![\texttt{EXISTS \#}I_s : G[\texttt{\#}I_s/\texttt{next}]\ \texttt{AND}\ \texttt{IF \#}I_s\texttt{.continues OR \#}I_s\texttt{.breaks}\ \texttt{THEN next.executes}\ \texttt{ELSE next == \#}I_s]\!](e)(s,s')$$

From (2), (3), and the definition of $[\![\_]\!]$, we know for some $k \in \mathbb{N}, t, u \in \mathit{State}^\infty$

(5) $\mathit{executes}(\mathit{control}(s))$

(6) $t(0) = s$

(7) $u(0) = s$

(8)
$$\begin{array}{l}\forall i \in \mathbb{N}_k : \\ \quad \neg\mathit{breaks}(\mathit{control}(u(i)))) \wedge \mathit{executes}(\mathit{control}(t(i))) \wedge \\ \quad [\![E]\!](t(i)) = \text{TRUE} \wedge [\![C]\!](t(i),u(i+1)) \wedge \\ \quad \text{IF } \mathit{continues}(\mathit{control}(u(i+1))) \vee \mathit{breaks}(\mathit{control}(u(i+1))) \\ \qquad \text{THEN } t(i+1) = \mathit{execute}(u(i+1)) \\ \qquad \text{ELSE } t(i+1) = u(i+1)\end{array}$$

(9)
$$\begin{array}{l}[\![E]\!](t(k)) \neq \text{TRUE} \vee \\ \quad \neg(\mathit{executes}(\mathit{control}(u(k))) \vee \mathit{continues}(\mathit{control}(u(k))))\end{array}$$

(10) $t(k) = s'$

From the premises and the definition of $[\![\_]\!]$, we know

(11)
$$\begin{array}{l}\forall s,s' \in \mathit{State}, e \in \mathit{Environment} : \\ \quad \mathit{executes}(\mathit{control}(s)) \wedge [\![C]\!](s,s') \Rightarrow \\ \qquad [\![[F]_{I_1,\ldots,I_n}^{F_c,F_b,F_r;K_1,\ldots,K_m}]\!](e)(s,s')\end{array}$$

(12) $\mathit{Invariant}(G,H,F)_{I_1,\ldots,I_n}$

(13) $\#I_s$ does not occur in G

From (8), (11), and the definition of $[\![\_]\!]$, we know

$$
\begin{array}{l}
\forall i \in \mathbb{N}_k : \\
\quad \neg \mathit{breaks}(\mathit{control}(u(i))) \wedge \mathit{executes}(\mathit{control}(t(i))) \wedge \\
\quad [\![E]\!](t(i)) = \text{ TRUE} \wedge [\![F]\!](e)(t(i), u(i+1)) \wedge \\
\quad t(i) = u(i+1) \text{ EXCEPT } I_1, \ldots, I_n \wedge \\
\quad (\mathit{continues}(\mathit{control}(u(i+1))) \Rightarrow [\![F_c]\!](e)(t(i), u(i+1))) \wedge \\
\quad (\mathit{breaks}(\mathit{control}(u(i+1))) \Rightarrow [\![F_b]\!](e)(t(i), u(i+1))) \wedge \\
\quad (\mathit{returns}(\mathit{control}(u(i+1))) \Rightarrow [\![F_r]\!](e)(t(i), u(i+1))) \wedge \\
\quad (\mathit{throws}(\mathit{control}(u(i+1))) \Rightarrow \\
\qquad \mathit{key}(\mathit{control}(u(i+1))) \in \{K_1, \ldots, K_m\}) \wedge \\
\quad \text{IF } \mathit{continues}(\mathit{control}(u(i+1))) \vee \mathit{breaks}(\mathit{control}(u(i+1))) \\
\qquad \text{THEN } t(i+1) = \mathit{execute}(u(i+1)) \\
\qquad \text{ELSE } t(i+1) = u(i+1)
\end{array}
\tag{14}
$$

From (14), we can conclude

$$
\begin{array}{l}
\forall i \in \mathbb{N}_k : \\
\quad [\![F]\!](e)(t(i), u(i+1)) \wedge \\
\quad t(i) = u(i+1) \text{ EXCEPT } I_1, \ldots, I_n \wedge \\
\quad (t(i+1) = u(i+1) \vee t(i+1) = \mathit{execute}(u(i+1)))
\end{array}
\tag{15}
$$

From (6), (7), (15), (CD2), (NEQ), and (AVE), we know

(16) $\forall i \in \mathbb{N}_k : u(i) = u(i+1) \text{ EXCEPT } I_1, \ldots, I_n$

From (12) and the definition of *Invariant*, we know

(17) G has no free (mathematical or state) variables

(17a) $\$I_1, \ldots, \$I_n, \#I_s, \#I_t$ do not occur in $G, H,$ and F

$$
\begin{array}{l}
\forall e \in \mathit{ValueEnv}, s, s' \in \mathit{Store} : \\
\quad [\![\texttt{FORALL } \$I_1, \ldots, \$I_n \texttt{: ALLSTATE } \#I_s \texttt{,} \#I_t \texttt{:} \\
\qquad (G[\#I_s/\texttt{next}][\$I_1/I_1\texttt{'}, \ldots, \$I_n/I_n\texttt{'}] \\
\qquad\quad \texttt{AND } (\#I_s\texttt{.executes OR } \#I_s\texttt{.continues}) \\
\qquad\quad \texttt{AND } \#I_t\texttt{.executes} \\
\qquad\quad \texttt{AND } H[\#I_t/\texttt{now}][\$I_1/I_1, \ldots, \$I_n/I_n] \\
\qquad\quad \texttt{AND } F[\#I_t/\texttt{now}][\$I_1/I_1, \ldots, \$I_n/I_n]) \\
\qquad \texttt{=> } G)\,]\!](e)(s, s')
\end{array}
\tag{18}
$$

We define

(19) $e_0 := e[I_s \mapsto \mathit{control}(u(k))]_c$

To show (a.3.a), from (19) and the definition of $[\![\_]\!]$, it suffices to show

(a.3.b.1) $[\![G[\#I_s/\texttt{next}]]\!](e_0)(s,s')$
IF $\mathit{continues}(\mathit{control}(u(k))) \vee \mathit{breaks}(\mathit{control}(u(k)))$
(a.3.b.2) THEN $\mathit{executes}(\mathit{control}(s'))$
ELSE $\mathit{control}(s') = \mathit{control}(u')$

From (10), (14), and (CD1), we know (a.3.b.2).

Assume we can show

(a.3.c) $\forall i \in \mathbb{N}_{k+1} : [\![G]\!](e)(s,u(i))$

From (a.3.c), we know

(20) $[\![G]\!](e)(s,u(k))$

From (13), (19), (20), the definition of *Invariant*, and (CNEF2), we know

(21) $[\![G[\#I_s/\texttt{next}]]\!](e_0)(s,u(k))$

From (10) and (15), we know

(22) $s' = u(k) \vee s' = \mathit{execute}(u(k))$

If $s' = u(k)$, from (21) we know (a.3.b.1). Otherwise, from (22) we may assume

(23) $s' = \mathit{execute}(u(k))$

From (21), (CNEF0), and (PVFNE), we know

(24) $[\![G]\!][\#I_s/\texttt{next}](e_0)(s,(\mathit{store}(u(k)),\mathit{control}(s')))$

From (23) and (CD3), we know

(25) s' EQUALS $u(k)$

From (CD3), we know

(26) $u(k)$ EQUALS $(\mathit{store}(u(k)),\mathit{control}(s'))$

From (25), (26), and (TRE), we know

(27) s' EQUALS $(\mathit{store}(u(k)),\mathit{control}(s'))$

From (24), (27) and (ESF'), we know (a.3.c).

The proof of (a.3.c) proceeds by induction on i with bound k.

Induction Base We show

(a.3.c.1) $[\![G]\!](e)(s,u(0))$

By (7), it suffices to show

(a.3.c.1.a) $[\![G]\!](e)(s,s)$

We define

(28) $s'' := writes(s', I_1, read(s,I_1), \ldots, I_n, read(s,I_n))$

From (4), (28), and (PPVF1'), we know

(29) $[\![G[\texttt{now/next}]]\!](e)(s,s'')$

From (29) and (PNNF2), we know

(30) $[\![G]\!](e)(s,(store(s''),control(s)))$

From (CD3), we know

(31) s'' EQUALS $(store(s''),control(s))$

From (28) and (WSE), we know

(32) $s'' = s'$ EXCEPT $I_1,\ldots,I_n$

From (a.4), (32), and (TRE), we know

(33) $s'' = s$ EXCEPT $I_1,\ldots,I_n$

From (28) and (RWE), we know

(34) $read(s'',I_1) = read(s,I_1) \wedge \ldots \wedge read(s'',I_n) = read(s,I_n)$

From (33), (34), (RVE), and (NEQ), we know

(35) s'' EQUALS s

From (31), (35), (NEQ), and (TRE), we know

(36) s EQUALS $(store(s''),control(s))$

From (REE) and (NEQ), we know

(37) s EQUALS s

From (30), (36), (37), and (ESF'), we know (a.3.c.1.a).

Induction Step Take arbitrary $i \in \mathbb{N}_k$. We assume

(38) $[\![G]\!](e)(s, u(i))$

and show

(a.3.c.2) $[\![G]\!](e)(s, u(i+1))$

From (18) and the definition of $[\![\_]\!]$, we know

$$
\begin{array}{l}
\forall v_1, \ldots, v_n \in \mathit{Value}, c_s, c_t \in \mathit{Control} : \\
\quad \text{LET} \\
\quad\quad e_0 := e[I_s \mapsto c_s, I_t \mapsto c_t]_c[I_1 \mapsto v_1, \ldots, I_n \mapsto v_n] \\
\quad \text{IN} \\
\quad\quad [\![G[\#I_s/\texttt{next}][\$I_1/I_1\texttt{'}, \ldots, \$I_n/I_n\texttt{'}]]\!](e_0)(s, u(i+1)) \wedge \\
\quad\quad (\mathit{executes}(c_s) \vee \mathit{continues}(c_s)) \wedge \\
\quad\quad \mathit{executes}(c_t) \wedge \\
\quad\quad [\![H[\#I_t/\texttt{now}][\$I_1/I_1, \ldots, \$I_n/I_n]]\!](e_0)(s, u(i+1)) \wedge \\
\quad\quad [\![F[\#I_t/\texttt{now}][\$I_1/I_1, \ldots, \$I_n/I_n]]\!](e_0)(s, u(i+1)) \;\Rightarrow \\
\quad\quad\quad [\![G]\!](e_0)(s, u(i+1))
\end{array}
\tag{39}
$$

We define

$$
\begin{array}{l}
e_0 := e[I_s \mapsto \mathit{control}(u(i))]_c \\
e_1 := e_0[I_t \mapsto \mathit{control}(t(i))]_c \\
e_2 := e_1[I_1 \mapsto \mathit{read}(u(i), I_1), \ldots, I_n \mapsto \mathit{read}(u(i), I_n)]
\end{array}
\tag{40}
$$

To prove (a.3.c.2), by (17) and (MVF'), it suffices to prove

(a.3.c.2.a) $[\![G]\!](e_2)(s, u(i+1))$

To prove (a.3.c.2.a), by (39), it suffices to prove

(a.3.c.2.a.1) $[\![G[\#I_s/\texttt{next}][\$I_1/I_1\texttt{'}, \ldots, \$I_n/I_n\texttt{'}]]\!](e_2)(s, u(i+1))$

(a.3.c.2.a.2) $\mathit{executes}(\mathit{control}(u(i))) \vee \mathit{continues}(\mathit{control}(u(i)))$

(a.3.c.2.a.3) $\mathit{executes}(\mathit{control}(t(i)))$

(a.3.c.2.a.4) $[\![H[\#I_t/\texttt{now}][\$I_1/I_1, \ldots, \$I_n/I_n]]\!](e_2)(s, u(i+1))$

(a.3.c.2.a.5) $[\![F[\#I_t/\texttt{now}][\$I_1/I_1, \ldots, \$I_n/I_n]]\!](e_2)(s, u(i+1))$

From (13), (38), (40), and (CNEF2), we know

(41) $[\![G[\#I_s/\texttt{next}]]\!](e_0)(s, u(i))$

From (13), (40), (41), we also know

(42) $[\![G[\#I_s/\texttt{next}]]\!](e_1)(s,u(i))$

From (16), we know

(43) $u(i) = u(i+1)$ EXCEPT $I_1,\ldots,I_n$

From (17a), (40), (42), (43), (CNEF0), and (PMVF2'), we know (a.3.c.2.a.1).

To show (a.3.c.2.a.2), we make a case distinction:

- Case $i = 0$: from (5) and (7), we know (a.3.c.2.a.2).
- Case $i > 0$: from the case condition and (14), we know

  (44) $\neg \mathit{breaks}(\mathit{control}(u(i))$
  IF $\mathit{continues}(\mathit{control}(u(i))) \vee \mathit{breaks}(\mathit{control}(u(i)))$
  (45) THEN $t(i) = \mathit{execute}(u(i))$
  ELSE $t(i) = u(i)$

  From (44) and (45), we know

  IF $\mathit{continues}(\mathit{control}(u(i)))$
  (46) THEN $t(i) = \mathit{execute}(u(i))$
  ELSE $t(i) = u(i)$

  From (a.3.c.2.a.3) (shown below) and (46), we know (a.3.c.2.a.2).

From (14), we know (a.3.c.2.a.3).

From (8), (12), and the definition of $\simeq$, we know

(47) $[\![H]\!](e)(t(i),u(i+1))$

(48) H does not have free (mathematical or state) variables

From (17a), (40), (47), and (CNOF2), we know

(49) $[\![H[\#I_t/\texttt{now}]]\!](e_1)(t(i),u(i+1))$

From (7), (16), and (TRE), we know

(50) $s = u(i)$ EXCEPT $I_1,\ldots,I_n$

From (6), (7), and (14), we know

(51) $t(i) = u(i) \vee t(i) = execute(u(i))$

From (51), (NEQ), (REE), and (CD2), we know

(52) $t(i)$ EQUALS $u(i)$

From (50), (52), (NEQ), (AVE), and (TRE), we know

(53) $s = t(i)$ EXCEPT $I_1, \ldots, I_n$

From (17a), (40), (49), (53), (CNOF0), and (PMVF1'), we know (a.3.c.2.a.4).

From (14), we know

(54) $[\![F]\!](e)(t(i), u(i+1))$

(55) $t(i) = u(i+1)$ EXCEPT $I_1, \ldots, I_n$

(56) $t(i+1) = u(i+1) \vee t(i+1) = execute(u(i+1))$

From (17a), (40), (54), and (CNOF2), we know

(57) $[\![F[\#I_t/\texttt{now}]]\!](e_1)(t(i), u(i+1))$

From (52), (NEQ), (AVE), and (RSE), we know

(58) $read(t(i), I_1) = read(u(i), I_1) \wedge \ldots \wedge read(t(i), I_n) = read(u(i), I_n)$

From (17a), (40), (53), (58), (59), (CNOF0), and (PMVF1'), we finally know (a.3.c.2.a.5). □

5.4.4 Invariant Rule (Without Breaks)

$C : [F]^{F_c,\texttt{FALSE},F_r;K_1,\ldots,K_m}_{I_1,\ldots,I_n}$
$E \simeq H$
$Invariant(G, H, F)_{I_1,\ldots,I_n}$
$\#I_s$ does not occur in G

```
while (E) C :
   [!next.continues AND !next.breaks AND
   (next.executes =>
      !H[next/now][I1'/I1,...,In'/In]) AND
   (G[now/next][I1/I1',...,In/In'] =>
      EXISTS #Is : G[#Is/next] AND
         IF #Is.continues OR #Is.breaks
            THEN next.executes
            ELSE next == #Is)
   ]^{FALSE,FALSE,Fr;K1,...,Km}_{I1,...,In}
```

Soundness Proof Analogous to the proof of the basic rule without breaks combined with the proof of the invariant rule with breaks. □

5.5 Termination and Interruptions

We proceed with the adaptation of the termination calculus of Section 3.3 to the command language with interruptions. Figure 5.14 depicts the corresponding termination conditions; apart from the adding rules for the new commands, only the rules for command sequences and loops have changed. Also the extended version satisfies the condition stated below.

Theorem (Termination Condition) If the termination condition of a program respectively command is true on an executing prestate s, then the program respectively command relates s to some poststate s':

$$\begin{array}{l}\forall P \in \text{Program}, s \in \mathit{State}: \\ \quad [\![P]\!]_{\mathrm{T}}(s) \wedge \mathit{executes}(\mathit{control}(s)) \Rightarrow \exists s' \in \mathit{State}: [\![P]\!](s,s') \\ \forall C \in \text{Command}, s \in \mathit{State}: \\ \quad [\![C]\!]_{\mathrm{T}}(s) \wedge \mathit{executes}(\mathit{control}(s)) \Rightarrow \exists s' \in \mathit{State}: [\![C]\!](s,s')\end{array}$$

Proof The first part of the theorem is immediately clear by the definition of $[\![\_]\!]_{\mathrm{T}} : \text{Program} \to \mathit{StateCondition}$ and by the second part of the theorem. We are now going to show this second part i.e.

(a) $$\begin{array}{l}\forall C \in \text{Command}, s \in \mathit{State}: \\ \quad [\![C]\!]_{\mathrm{T}}(s) \wedge \mathit{executes}(\mathit{control}(s)) \Rightarrow \exists s' \in \mathit{State}: [\![C]\!](s,s')\end{array}$$

Take arbitrary $C_0 \in \text{Command}, s \in \mathit{State}$ and assume

(1a) $[\![C_0]\!]_{\mathrm{T}}(s)$

(1b) $\mathit{executes}(\mathit{control}(s))$

We show

(b) $\exists s' \in \mathit{State}: [\![C_0]\!](s,s')$

We proceed by induction on the structure of C_0 (we only deal with the rules for command sequences, exception handling, and loops; the proofs of the others are either trivial or proceed as in Section 3.3).

Command Language with Interruptions: Termination

$$
\begin{array}{l}
\mathit{infiniteExecution}: \\
\quad \mathbb{P}(\mathit{State}^{\infty} \times \mathit{State}^{\infty} \times \mathit{State} \times \mathit{StateFunction} \times \mathit{StateRelation}) \\
\mathit{infiniteExecution}(t,u,s,E,C) \Leftrightarrow \\
\quad t(0) = s \wedge u(0) = s \wedge \\
\quad \forall i \in \mathbb{N}: \\
\qquad \neg \mathit{breaks}(\mathit{control}(u(i))) \wedge \mathit{executes}(\mathit{control}(t(i))) \wedge \\
\qquad E(t(i)) = \text{TRUE} \wedge C(t(i),u(i+1)) \wedge \\
\qquad \text{IF } \mathit{continues}(\mathit{control}(u(i+1))) \vee \mathit{breaks}(\mathit{control}(u(i+1))) \\
\qquad\quad \text{THEN } t(i+1) = \mathit{execute}(u(i+1)) \\
\qquad\quad \text{ELSE } t(i+1) = u(i+1)
\end{array}
$$

$[\![\_]\!]_{\mathrm{T}}$: **Command** $\rightarrow$ ***StateCondition***

$$
\begin{array}{l}
[\![I = E]\!]_{\mathrm{T}}(s) \Leftrightarrow \text{ TRUE} \\
[\![\texttt{var } I\texttt{; } C]\!]_{\mathrm{T}}(s) \Leftrightarrow \forall v \in \mathit{Value}: [\![C]\!]_{\mathrm{T}}(\mathit{write}(s,I,v)) \\
[\![\texttt{var } I\texttt{=}E\texttt{; } C]\!]_{\mathrm{T}}(s) \Leftrightarrow [\![C]\!]_{\mathrm{T}}(\mathit{write}(s,I,[\![E]\!](s))) \\
[\![C_1\texttt{; } C_2]\!]_{\mathrm{T}}(s) \Leftrightarrow \\
\quad [\![C_1]\!]_{\mathrm{T}}(s) \wedge \\
\quad \forall s' \in \mathit{State}: [\![C_1]\!](s,s') \wedge \mathit{executes}(\mathit{control}(s')) \Rightarrow [\![C_2]\!]_{\mathrm{T}}(s') \\
[\![\texttt{if (}E\texttt{) } C]\!]_{\mathrm{T}}(s) \Leftrightarrow [\![E]\!](s) = \text{ TRUE } \Rightarrow [\![C]\!]_{\mathrm{T}}(s) \\
[\![\texttt{if (}E\texttt{) } C_1 \texttt{ else } C_2]\!]_{\mathrm{T}}(s) \Leftrightarrow \\
\quad \text{IF } [\![E]\!](s) = \text{ TRUE THEN } [\![C_1]\!]_{\mathrm{T}}(s) \text{ ELSE } [\![C_2]\!]_{\mathrm{T}}(s) \\
[\![\texttt{continue}]\!]_{\mathrm{T}}(s) \Leftrightarrow \text{TRUE} \\
[\![\texttt{break}]\!]_{\mathrm{T}}(s) \Leftrightarrow \text{TRUE} \\
[\![\texttt{return } E]\!]_{\mathrm{T}}(s) \Leftrightarrow \text{TRUE} \\
[\![\texttt{throw } I\ E]\!]_{\mathrm{T}}(s) \Leftrightarrow \text{TRUE} \\
[\![\texttt{try } C_1 \texttt{ catch (}I_k\ I_v\texttt{) } C_2]\!]_{\mathrm{T}}(s) \Leftrightarrow \\
\quad [\![C_1]\!]_{\mathrm{T}}(s) \wedge \\
\quad \forall s' \in \mathit{State}: \\
\qquad [\![C_1]\!](s,s') \wedge \mathit{throws}(\mathit{control}(s')) \wedge \mathit{key}(\mathit{control}(s')) = I_k \Rightarrow \\
\qquad [\![C_2]\!]_{\mathrm{T}}(\mathit{write}(\mathit{execute}(s'),I_v,\mathit{value}(\mathit{control}(s')))) \\
[\![\texttt{while (}E\texttt{) } C]\!]_{\mathrm{T}}(s) \Leftrightarrow \\
\quad \forall t,u \in \mathit{State}^{\infty}, k \in \mathbb{N}: \\
\qquad \neg \mathit{infiniteExecution}(t,u,s,[\![E]\!],[\![C]\!]) \wedge \\
\qquad (\mathit{finiteExecution}(k,t,u,s,[\![E]\!],[\![C]\!]) \wedge [\![E]\!](t(k)) = \text{TRUE} \wedge \\
\qquad\quad (\mathit{executes}(\mathit{control}(u(k))) \vee \mathit{continues}(\mathit{control}(u(k))))) \\
\qquad\qquad \Rightarrow [\![C]\!]_{\mathrm{T}}(t(k)))
\end{array}
$$

Figure 5.14: The Termination Conditions of the Command Language

C_1 ; C_2 From (1a) and the definition of $[\![\_]\!]_T$, we know

(2) $[\![C_1]\!]_T(s)$
(3) $\forall s' \in State : [\![C_1]\!](s,s') \wedge executes(control(s')) \Rightarrow [\![C_2]\!]_T(s')$

From the definition of $[\![\_]\!]$, it suffices to show

(c)
$$\begin{array}{l}\exists s_0, s' \in State : \\ \quad [\![C_1]\!](s,s_0) \wedge \\ \quad \text{IF } executes(control(s_0)) \text{ THEN } [\![C_2]\!](s_0,s') \text{ ELSE } s' = s_0\end{array}$$

From (1b), (2), and the induction hypothesis, we know for some $s_0 \in State$

(4) $[\![C_1]\!](s,s_0)$

We proceed by case distinction.

- Case $executes(control(s_0))$: from the case condition, (3), and (4), we know
  (5) $[\![C_2]\!]_T(s_0)$
  From (5), the case condition, and the induction hypothesis, we know for some $s' \in State$
  (6) $[\![C_2]\!](s_0,s')$
  From the case condition, (4) and (6), we know (c).
- Case $\neg executes(control(s_0))$: from the case condition and (4), we know (c) (for $s' := s_0$).

`try` C_1 `catch(`I_k I_v`)` C_2 From (1a) and the definition of $[\![\_]\!]_T$, we know

(2) $[\![C_1]\!]_T(s)$
(3)
$$\begin{array}{l}\forall s' \in State : \\ \quad [\![C_1]\!](s,s') \wedge throws(control(s')) \wedge key(control(s')) = I_k \Rightarrow \\ \quad [\![C_2]\!]_T(write(execute(s'), I_v, value(control(s'))))\end{array}$$

From the definition of $[\![\_]\!]$, it suffices to show

(c)
$$\begin{array}{l}\exists s_0, s_1, s_2, s' \in State : \\ \quad [\![C_1]\!](s,s_0) \wedge \\ \quad \text{IF } throws(control(s_0)) \wedge key(control(s_0)) = I_k \text{ THEN} \\ \qquad s_1 = write(execute(s_0), I_v, value(control(s_0))) \wedge \\ \qquad [\![C_2]\!](s_1,s_2) \wedge \\ \qquad s' = write(s_2, I_v, read(s_0, I_v)) \\ \quad \text{ELSE } s' = s_0\end{array}$$

From (1b), (2), and the induction hypothesis, we know for some $s_0 \in State$

(4) $[\![C_1]\!](s,s_0)$

We proceed by case distinction.

- Case $\mathit{throws}(\mathit{control}(s_0)) \wedge \mathit{key}(\mathit{control}(s_0)) = I_k$: we define
  (5) $s_1 := \mathit{write}(\mathit{execute}(s_0), I_v, \mathit{value}(\mathit{control}(s_0)))$
  From the case condition, (3), and (4), and (5), we know
  (6) $[\![C_2]\!]_{\mathrm{T}}(s_1)$
  From (5), (CD1), and (CW), we know
  (7) $\mathit{executes}(\mathit{control}(s_1))$
  From (6), (7), and the induction hypothesis, we also know for some $s_2 \in \mathit{State}$
  (8) $[\![C_2]\!](s_1, s_2)$
  From the case condition, (4), (5), and (8), we know (c) (for $s' := \mathit{write}(s_2, I_v, \mathit{read}(s_0, I_v))$).
- Case $\neg(\mathit{throws}(\mathit{control}(s_0)) \wedge \mathit{key}(\mathit{control}(s_0)) = I_k)$: from the case condition and (4), we know (c) (for arbitrary s_1, s_2 and $s' = s_0$).

`while (E) C` From (1a) and the definition of $[\![\_]\!]_{\mathrm{T}}$, we know

$$
\begin{array}{l}
\forall t, u \in \mathit{State}^\infty, k \in \mathbb{N} : \\
\quad \neg \mathit{infiniteExecution}(t, u, s, [\![E]\!], [\![C]\!]) \wedge \\
\quad (\mathit{finiteExecution}(k, t, u, s, [\![E]\!], [\![C]\!]) \wedge [\![E]\!](t(k)) = \mathrm{TRUE} \wedge \\
\qquad (\mathit{executes}(\mathit{control}(u(k))) \vee \mathit{continues}(\mathit{control}(u(k))))) \\
\qquad\quad \Rightarrow [\![C]\!]_{\mathrm{T}}(t(k)))
\end{array}
\tag{2}
$$

From the definition of $[\![\_]\!]$, it suffices to show

$$
\begin{array}{l}
\exists k \in \mathbb{N}, t, u \in \mathit{State}^\infty, s' \in \mathit{State} : \\
\quad \mathit{finiteExecution}(k, t, u, s, [\![E]\!], [\![C]\!]) \wedge \\
\quad ([\![E]\!](t(k)) \neq \mathrm{TRUE} \vee \\
\qquad \neg(\mathit{executes}(\mathit{control}(u(k))) \vee \\
\qquad\quad \mathit{continues}(\mathit{control}(u(k))))) \wedge \\
\quad t(k) = s'
\end{array}
\tag{c}
$$

which can be simplified to

$$
\begin{array}{l}
\exists k \in \mathbb{N}, t, u \in \mathit{State}^\infty : \\
\quad \mathit{finiteExecution}(k, t, u, s, [\![E]\!], [\![C]\!]) \wedge \\
\quad ([\![E]\!](t(k)) \neq \mathrm{TRUE} \vee \\
\qquad \neg(\mathit{executes}(\mathit{control}(u(k))) \vee \\
\qquad\quad \mathit{continues}(\mathit{control}(u(k)))))
\end{array}
\tag{d}
$$

We define $c : \mathit{State} \Rightarrow \mathit{State}$ as follows:

$$(3)\quad c(s) := \begin{array}{l}\text{IF } \exists s' \in State : [\![C]\!](s,s') \\ \quad \text{THEN SUCH } s' \text{ IN } State : [\![C]\!](s,s') \\ \quad \text{ELSE } s\end{array}$$

We define $u,t \in State^{\infty}$ inductively as follows:

$$(4)\quad u(0) := s$$
$$(5)\quad t(0) := s$$
$$(6)\quad u(i+1) := c(t(i))$$
$$(7)\quad t(i+1) := \begin{array}{l}\text{IF } continues(control(u(i+1))) \vee \\ \qquad breaks(control(u(i+1)) \\ \quad \text{THEN } execute(u(i+1)) \\ \quad \text{ELSE } u(i+1)\end{array}$$

We define

$$(8)\quad \begin{array}{l} loop(i) :\Leftrightarrow \\ \quad \neg breaks(control(u(i))) \wedge executes(control(t(i))) \wedge \\ \quad [\![E]\!](t(i)) = \text{TRUE} \wedge [\![C]\!](t(i),u(i+1)) \wedge \\ \quad \text{IF } continues(control(u(i+1))) \vee breaks(control(u(i+1))) \\ \qquad \text{THEN } t(i+1) = execute(u(i+1)) \\ \qquad \text{ELSE } t(i+1) = u(i+1)\end{array}$$

From (2), we know

$$(9)\quad \neg infiniteExecution(t,u,s,[\![E]\!],[\![C]\!])$$

From (8), (9), and the definition of *infiniteExecution*, we know

$$(10)\quad t(0) \neq s \ \vee u(0) \neq s \vee \exists i \in \mathbb{N} : \neg loop(i)$$

From (4), (5), and (10), we know

$$(11)\quad \exists i \in \mathbb{N} : \neg loop(i)$$

We define

$$(12)\quad k := \text{MIN } i \in \mathbb{N} : \neg loop(i)$$

From (11) and (12), we know

$$(13)\quad \neg loop(k)$$
$$(14)\quad \forall i \in \mathbb{N}_k : loop(i)$$

From (8), (14), and the definition of *finiteExecution*, we know

$$(15)\quad finiteExecution(k,t,u,s,[\![E]\!],[\![C]\!])$$

It now suffices to show

Termination Calculus: Judgements

$$C \downarrow F \Leftrightarrow \\ \forall s \in State : executes(control(s)) \wedge [\![F]\!](s) \Rightarrow [\![C]\!]_{\mathrm{T}}(s)$$

Figure 5.15: The Termination Calculus of the Command Language (Part 1 of 4)

(e) $[\![E]\!](t(k)) \neq \text{TRUE} \vee \\ \neg(executes(control(u(k))) \vee continues(control(u(k))))$

because from this and (15), we know (d).

To show (e), we assume

(16) $[\![E]\!](t(k)) = \text{TRUE}$
(17) $executes(control(u(k))) \vee continues(control(u(k)))$

and show a contradiction.

From (17) and Lemma "State Control Predicates", we know

(18) $\neg breaks(control(u(k)))$

From (1b), (5), (7), (17), and (CD1), we know

(19) $executes(control(t(k)))$

From (2), (15), (16), and (17), we know

(20) $[\![C]\!]_{\mathrm{T}}(t(k))$

From (20) and the induction assumption, we know

(21) $\exists s' \in State : [\![C]\!](t(k), s')$

From (3), (6), and (21), we know

(22) $[\![C]\!](t(k), u(k+1))$

But (7), (8), (16), (18), (19), and (22) contradict (13). □

Figures 5.15, 5.16, 5.17, and 5.18 adapt the calculus of Section 3.4 to reason about the termination of programs that may trigger control flow interruptions. The judgements of the calculus have the form $C \downarrow F$ where F is a formula that does not contain free variables and does not depend on the poststate. The interpretation of the judgement is given by the following soundness theorem.

Termination Calculus: Rules

$$I = E \downarrow F$$

$$\frac{\$I \text{ does not occur in } F \qquad C \downarrow \texttt{EXISTS } \$I : F[\$I/I]}{\texttt{var } I\texttt{; } C \downarrow F}$$

$$\frac{E \simeq T \qquad \$I \text{ does not occur in } T \text{ and } F \qquad C \downarrow \texttt{EXISTS } \$I : I\texttt{=}T[\$I/I] \texttt{ AND } F[\$I/I]}{\texttt{var } I\texttt{=}E\texttt{; } C \downarrow F}$$

$$\frac{C_1 \downarrow F \qquad C_2 \downarrow \texttt{now.executes}}{C_1\texttt{;}C_2 \downarrow F}$$

$$\frac{C_1 \downarrow F \qquad C_1 : [S]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \qquad \$I_1,\ldots,\$I_n, \#I_s \text{ do not occur in } F \text{ and } S \qquad C_2 \downarrow \texttt{now.executes AND}\ \texttt{EXISTS } \$I_1,\ldots,\$I_n\texttt{:}\ \texttt{EXSTATE } \#I_s\texttt{:}\ \#I_s\texttt{.executes AND}\ F[\#I_s/\texttt{now}][\$I_1/I_1,\ldots,\$I_n/I_n]\ \texttt{AND}\ S[\#I_s/\texttt{now}][\$I_1/I_1,\ldots,\$I_n/I_n]\ [\texttt{now/next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]}{C_1\texttt{;}C_2 \downarrow F}$$

$$\frac{E \simeq T \qquad C \downarrow F \texttt{ AND } T}{\texttt{if (}E\texttt{) } C \downarrow F}$$

$$\frac{E \simeq T \qquad C_1 \downarrow F \texttt{ AND } T \qquad C_2 \downarrow F \texttt{ AND !}T}{\texttt{if (}E\texttt{) } C_1 \texttt{ else } C_2 \downarrow F}$$

Figure 5.16: The Termination Calculus of the Command Language (Part 2 of 4)

Termination Calculus: Rules

$$\texttt{continue} \downarrow F$$

$$\texttt{break} \downarrow F$$

$$\texttt{return} \downarrow F$$

$$\texttt{throw}\ I\ E \downarrow F$$

$$\frac{\begin{array}{l} C_1 \downarrow F \\ C_1 : [S]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \\ C_2 \downarrow \texttt{now.executes AND} \\ \quad \texttt{EXISTS } \$I_1,\ldots,\$I_n,\$I_a,\$I_b\texttt{: EXSTATE } \#I_s,\#I_t\texttt{:} \\ \quad\quad \#I_s\texttt{.executes AND} \\ \quad\quad \#I_t\texttt{.throws } I_k \texttt{ AND } I_v = \#I_t\texttt{.value AND} \\ \quad\quad F[\#I_s/\texttt{now}][\$I_1/I_1,\ldots,\$I_n/I_n][\$I_a/I_v]\texttt{ AND} \\ \quad\quad S[\#I_s/\texttt{now},\#I_t/\texttt{next}][\$I_1/I_1,\ldots,\$I_n/I_n] \\ \quad\quad\quad [\$I_a/I_v,\$I_b/I_v{}'][I_1/I_1{}',\ldots,I_n/I_n{}'] \\ I_a \neq I_b \wedge \{I_a,I_b\} \cap \{I_1,\ldots,I_n\} = \emptyset \wedge I_s \neq I_t \\ \$I_1,\ldots,\$I_n,\$I_a,\$I_b,\#I_s,\#I_t \text{ do not occur in } F \text{ and } S \end{array}}{\texttt{try } C_1 \texttt{ catch(}I_k\ I_v\texttt{) } C_2 \downarrow F}$$

Figure 5.17: The Termination Calculus of the Command Language (Part 3 of 4)

Termination Calculus: Rules

$$\frac{\forall s \in \mathit{State} : [\![F \texttt{ => } \texttt{!}E]\!](s)}{\texttt{while (}E\texttt{) } C \downarrow F}$$

$$\frac{\begin{array}{l} E \simeq H \\ C : [S]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \\ \mathit{Invariant}(G,H,S)_{I_1,\ldots,I_n} \\ T \text{ has no free variables and no primed program variables} \\ \quad \text{and no occurrence of } \texttt{now} \text{ or } \texttt{next} \\ \forall s \in \mathit{State} : [\![F \texttt{ => } G[\texttt{now}/\texttt{next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]]\!](s) \end{array}}{\begin{array}{l} C \downarrow \texttt{ EXISTS } \$I_1,\ldots,\$I_n\texttt{ : EXSTATE } \#I_s,\#I_t\texttt{ :} \\ \quad F[\#I_s/\texttt{now}][\$I_1/I_1,\ldots,\$I_n/I_n]\texttt{ AND} \\ \quad G[\#I_s/\texttt{now},\#I_t/\texttt{next}] \\ \qquad [\$I_1/I_1,\ldots,\$I_n/I_n,I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]\texttt{ AND } H\texttt{ AND} \\ \quad (\#I_t\texttt{.executes OR }\#I_t\texttt{.continues}) \end{array}}$$

$$\frac{\begin{array}{l} \forall e \in \mathit{Environment}, s, s' \in \mathit{Store}, v_1,\ldots,v_n \in \mathit{Value}, c_s, c_t \in \mathit{Control} : \\ \quad \text{LET } e_0 = e[I_s \mapsto c_s, I_t \mapsto c_t]_c[I_1 \mapsto v_1,\ldots,I_n \mapsto v_n] \text{ IN} \\ \qquad [\![F[\#I_s/\texttt{now}][\$I_1/I_1,\ldots,\$I_n/I_n]\texttt{ AND} \\ \qquad\quad G[\#I_s/\texttt{now},\#I_t/\texttt{next}] \\ \qquad\qquad [\$I_1/I_1,\ldots,\$I_n/I_n,I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]\texttt{ AND } H\texttt{ AND} \\ \qquad\quad (\#I_t\texttt{.executes OR }\#I_t\texttt{.continues})\texttt{ AND} \\ \qquad\quad \texttt{now.executes AND } [S]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}\texttt{ AND} \\ \qquad\quad \texttt{(next.executes OR next.continues)} \\ \qquad]\!](e_0)(s,s') \Rightarrow \\ \qquad \text{LET} \\ \qquad\quad m = [\![T]\!](e_0)(s,s'), \\ \qquad\quad m' = [\![T[I_1\texttt{'}/I_1,\ldots,I_n\texttt{'}/I_n]]\!](e_0)(s,s') \\ \qquad \text{IN } m \in \mathbb{N} \wedge m > m' \end{array}}{\texttt{while (}E\texttt{) } C \downarrow F}$$

Figure 5.18: The Termination Calculus of the Command Language (Part 4 of 4)

Theorem (Soundness of the Termination Calculus) Assume the condition denoted by *DifferentVariables*. If a judgement $C \downarrow F$ can be derived from the rules of the termination calculus of the command language, then it is true that

$$\begin{array}{l} F \text{ has no free (mathematical or state) variables} \wedge \\ F \text{ does not depend on the poststate } \Rightarrow \\ \quad \forall s \in \mathit{State} : \mathit{executes}(\mathit{control}(s)) \wedge [\![F]\!](s) \Rightarrow [\![C]\!]_{\mathrm{T}}(s) \end{array}$$

Corollary (Existence of Poststate) If $C \downarrow F$ can be derived, then for every prestate s with $[\![F]\!](s)$, there exists some poststate s' with $[\![C]\!](s, s')$.

Proof (Soundness Theorem and Corollary) Assume

(1a) *DifferentVariables*

Take C and F such that $C \downarrow F$ can be derived and assume

(1b) F has no free (mathematical or state) variables

(1c) $\forall s, s', s'' \in \mathit{State}, e \in \mathit{Environment} :$
$[\![F]\!](e)(s, s') \Leftrightarrow [\![F]\!](e)(s, s'')$

Take arbitrary $s \in \mathit{State}$. We prove

$$\mathit{executes}(\mathit{control}(s)) \wedge [\![F]\!](s) \Rightarrow [\![C]\!]_{\mathrm{T}}(s)$$

by induction on the derivation of $C \downarrow F$. The following subsections cover all cases for the last step of such a derivation.

In these proofs, we assume that the induction hypothesis immediately implies $\forall s \in \mathit{State} : \mathit{executes}(\mathit{control}(s)) \wedge [\![F']\!](s) \Rightarrow [\![C']\!]_{\mathrm{T}}(s))$ i.e. we assume that F' does not have any free variables and that it does not depend on the poststate. In addition to the caveats already mentioned in Section 3.4, we have to make by corresponding substitutions sure that references to `next` in specification formulas do not cause harm.

The corollary follows from the soundness theorem and Theorem "Termination Condition". □

In the following, we contend ourselves with proofs for the three most important rules; the others are either simple or similar to the proofs given in Section 3.4.

5.5.1 Command Sequence

$$\frac{\begin{array}{l} C_1 \downarrow F \\ C_1 : [S]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \\ \$I_1,\ldots,\$I_n, \#I_s \text{ do not occur in } F \text{ and } S \\ C_2 \downarrow \texttt{now.executes AND} \\ \quad \texttt{EXISTS } \$I_1,\ldots,\$I_n\texttt{: EXSTATE } \#I_s\texttt{:} \\ \quad\quad \#I_s\texttt{.executes AND} \\ \quad\quad F[\#I_s/\texttt{now}][\$I_1/I_1,\ldots,\$I_n/I_n]\ \texttt{AND} \\ \quad\quad S[\#I_s/\texttt{now}][\$I_1/I_1,\ldots,\$I_n/I_n] \\ \quad\quad\quad [\texttt{now/next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}] \end{array}}{C_1\,\texttt{;}\,C_2 \downarrow F}$$

Soundness Proof We have to show

(a) $\mathit{executes}(\mathit{control}(s)) \wedge [\![F]\!](s) \Rightarrow [\![C_1\,\texttt{;}\; C_2]\!]_{\mathrm{T}}(s)$

We assume

(2) $\mathit{executes}(\mathit{control}(s))$

(3) $[\![F]\!](s)$

By the definition of $[\![\_]\!]_{\mathrm{T}}$, it suffices to show

(b.1) $[\![C_1]\!]_{\mathrm{T}}(s)$

(b.2) $\forall s' \in \mathit{State} : [\![C_1]\!](s,s') \wedge \mathit{executes}(\mathit{control}(s')) \Rightarrow [\![C_2]\!]_{\mathrm{T}}(s')$

From the premises, the induction hypothesis, and the definition of $[\![\_]\!]$, we know

(4) $\forall s \in \mathit{State} : \mathit{executes}(\mathit{control}(s)) \wedge [\![F]\!](s) \Rightarrow [\![C_1]\!]_{\mathrm{T}}(s)$

(5) $\forall s,s' \in \mathit{State}, e \in \mathit{Environment} :$
$\mathit{executes}(\mathit{control}(s)) \wedge [\![C_1]\!](s,s') \Rightarrow [\![[S]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}]\!](e)(s,s')$

(5a) $\$I_1,\ldots,\$I_n, \#I_s$ do not occur in F and S

(6)
$\forall s \in \mathit{State} :$
$\mathit{executes}(\mathit{control}(s)) \wedge$
$[\![\texttt{now.executes AND}$
$\quad \texttt{EXISTS } \$I_1,\ldots,\$I_n\texttt{: EXSTATE } \#I_s\texttt{:}$
$\quad\quad \#I_s\texttt{.executes AND}$
$\quad\quad F[\#I_s/\texttt{now}][\$I_1/I_1,\ldots,\$I_n/I_n]\ \texttt{AND}$
$\quad\quad S[\#I_s/\texttt{now}][\$I_1/I_1,\ldots,\$I_n/I_n]$
$\quad\quad\quad [\texttt{now/next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]]\!](s)$
$\Rightarrow [\![C_2]\!]_{\mathrm{T}}(s)$

From (2), (3), and (4), we know (b.1).

To show (b.2), we take arbitrary $s' \in \mathit{State}$ and assume

(7) $[\![C_1]\!](s,s')$

(8) $\mathit{executes}(\mathit{control}(s'))$

We have to show

(b.2.a) $[\![C_2]\!]_{\mathrm{T}}(s')$

To prove this, by (6) and the definition of $[\![\,\_\,]\!]$, it suffices to prove for arbitrary but fixed $e \in \mathit{Environment}$

(b.2.a.1) $\mathit{executes}(\mathit{control}(s'))$

(b.2.a.2) $\mathit{executes}(\mathit{control}(s'))$

(b.2.a.3)
$$\begin{array}{l}
\exists v_1,\ldots,v_n \in \mathit{Value}, c \in \mathit{Control}: \\
\quad \text{LET} \\
\quad\quad e_0 := e[I_s \mapsto c]_c[I_1 \mapsto v_1,\ldots,I_n \mapsto v_n] \\
\quad \text{IN} \\
\quad\quad \mathit{executes}(c) \wedge \\
\quad\quad [\![F[\#I_s/\mathtt{now}][\$I_1/I_1,\ldots,\$I_n/I_n]]\!](e_0)(s',s') \wedge \\
\quad\quad [\![S[\#I_s/\mathtt{now}][\$I_1/I_1,\ldots,\$I_n/I_n] \\
\quad\quad\quad [\mathtt{now}/\mathtt{next}][I_1/I_1{}',\ldots,I_n/I_n{}']]\!](e_0)(s',s')
\end{array}$$

From (8), we know (b.2.a.1) and (b.2.a.2).

We define

(9)
$$\begin{array}{l}
e_0 := e[I_s \mapsto \mathit{control}(s)] \\
e_1 := e_0[I_1 \mapsto \mathit{read}(s,I_1),\ldots,I_n \mapsto \mathit{read}(s,I_n)]
\end{array}$$

To show (b.2.a.3), it suffices to show

(b.2.a.3.a.1) $\mathit{executes}(\mathit{control}(s))$

(b.2.a.3.a.2) $[\![F[\#I_s/\mathtt{now}][\$I_1/I_1,\ldots,\$I_n/I_n]]\!](e_1)(s',s')$

(b.2.a.3.a.3)
$$\begin{array}{l}
[\![S[\#I_s/\mathtt{now}][\$I_1/I_1,\ldots,\$I_n/I_n] \\
\quad [\mathtt{now}/\mathtt{next}][I_1/I_1{}',\ldots,I_n/I_n{}']]\!](e_1)(s',s')
\end{array}$$

From (2), we know (b.2.a.3.a.1).

From (2), (5), (7), and the definition of $[\![\,\_\,]\!]$ and $[\,\_\,]_{\_}$, we know

(10) $[\![S]\!](e)(s,s')$

(11) $s = s'$ EXCEPT $I_1,\ldots,I_n$

From (3) and the definition of $[\![\_]\!]$, we know

(12) $[\![F]\!](e)(s,s)$

From (5a), (9), (12), and (CNOF2), we know

(13) $[\![F[\#I_s/\texttt{now}]]\!](e_0)(s,s)$.

From (5a), (9), (11), (13) (CNOF0), and (PMVF1'), we know

(14) $[\![F[\#I_s/\texttt{now}][\$I_1/I_1,\ldots,\$I_n/I_n]]\!](e_1)(s',s)$

From (1c) and (14), we know (b.2.a.3.a.2).

From (5a), (9), (10), and (CNOF2), we know

(15) $[\![S[\#I_s/\texttt{now}]]\!](e_0)(s,s')$

From (5a), (9), (11), (15), (CNOF0), and (PMVF1'), we know

(16) $[\![S[\#I_s/\texttt{now}][\$I_1/I_1,\ldots,\$I_n/I_n]]\!](e_1)(s',s')$

From (16), (CD0), and (PNNF2),

(17) $[\![S[\#I_s/\texttt{now}][\$I_1/I_1,\ldots,\$I_n/I_n][\texttt{now}/\texttt{next}]]\!](e_1)(s',s')$

From (IDE), we know

(18) $s' = \mathit{writes}(s',I_1,\mathit{read}(s',I_1),\ldots,I_n,\mathit{read}(s',I_n))$

From (17), (18), and (PPVF2'), we know (b.2.a.3.a.3). □

5.5.2 Catch Exception

$$\frac{\begin{array}{l}C_1 \downarrow F\\ C_1 : [S]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}\\ C_2 \downarrow \texttt{now.executes AND}\\ \quad\texttt{EXISTS } \$I_1,\ldots,\$I_n,\$I_a,\$I_b\texttt{: EXSTATE } \#I_s,\#I_t\texttt{:}\\ \quad\quad \#I_s\texttt{.executes AND}\\ \quad\quad \#I_t\texttt{.throws } I_k \texttt{ AND } I_v = \#I_t\texttt{.value AND}\\ \quad\quad F[\#I_s/\texttt{now}][\$I_1/I_1,\ldots,\$I_n/I_n][\$I_a/I_v] \texttt{ AND}\\ \quad\quad S[\#I_s/\texttt{now},\#I_t/\texttt{next}][\$I_1/I_1,\ldots,\$I_n/I_n]\\ \quad\quad\quad [\$I_a/I_v,\$I_b/I_v{}'][I_1/I_1{}',\ldots,I_n/I_n{}']\\ I_a \neq I_b \wedge \{I_a,I_b\} \cap \{I_1,\ldots,I_n\} = \emptyset \wedge I_s \neq I_t\\ \$I_1,\ldots,\$I_n,\$I_a,\$I_b,\#I_s,\#I_t \text{ do not occur in } F \text{ and } S\end{array}}{\texttt{try } C_1 \texttt{ catch (}I_k\ I_v\texttt{) } C_2 \downarrow F}$$

Soundness Proof We have to show

(a) $\mathit{executes}(\mathit{control}(s)) \wedge [\![F]\!](s) \Rightarrow [\![\texttt{try}\ C_1\ \texttt{catch}\ (I_k\ I_v)\ C_2]\!]_{\mathrm{T}}(s)$

We assume

(2) $\mathit{executes}(\mathit{control}(s))$

(3) $[\![F]\!](s)$

By the definition of $[\![\_]\!]_{\mathrm{T}}$, it suffices to show

(b.1) $[\![C_1]\!]_{\mathrm{T}}(s)$

(b.2) $\forall s' \in \mathit{State}:$
$[\![C_1]\!](s,s') \wedge \mathit{throws}(\mathit{control}(s')) \wedge \mathit{key}(\mathit{control}(s')) = I_k \Rightarrow$
$[\![C_2]\!]_{\mathrm{T}}(\mathit{write}(\mathit{execute}(s'), I_v, \mathit{value}(\mathit{control}(s'))))$

From the premises, the induction hypothesis, and the definition of $[\![\_]\!]$, we know

(4) $\forall s \in \mathit{State} : \mathit{executes}(\mathit{control}(s)) \wedge [\![F]\!](s) \Rightarrow [\![C_1]\!]_{\mathrm{T}}(s)$

(5) $\forall s, s' \in \mathit{State}, e \in \mathit{Environment}:$
$\mathit{executes}(\mathit{control}(s)) \wedge [\![C_1]\!](s,s') \Rightarrow [\![[S]_{I_1,\ldots,I_n}^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}]\!](e)(s,s')$

(6)
$\forall s \in \mathit{State}:$
$\mathit{executes}(\mathit{control}(s)) \wedge$
$[\![\texttt{now.executes AND}$
$\quad \texttt{EXISTS \$}I_1,\ldots,\texttt{\$}I_n,\texttt{\$}I_a,\texttt{\$}I_b\texttt{ : EXSTATE \#}I_s,\texttt{\#}I_t\texttt{ :}$
$\quad\quad \texttt{\#}I_s\texttt{.executes AND}$
$\quad\quad \texttt{\#}I_t\texttt{.throws }I_k\texttt{ AND }I_v = \texttt{\#}I_t\texttt{.value AND}$
$\quad\quad F[\texttt{\#}I_s/\texttt{now}][\texttt{\$}I_1/I_1,\ldots,\texttt{\$}I_n/I_n][\texttt{\$}I_a/I_v]\texttt{ AND}$
$\quad\quad S[\texttt{\#}I_s/\texttt{now},\texttt{\#}I_t/\texttt{next}][\texttt{\$}I_1/I_1,\ldots,\texttt{\$}I_n/I_n]$
$\quad\quad\quad [\texttt{\$}I_a/I_v,\texttt{\$}I_b/I_v{}']['I_1/I_1{}',\ldots,I_n/I_n{}']\,]\!](s)$
$\Rightarrow [\![C_2]\!]_{\mathrm{T}}(s)$

(7) $I_a \neq I_b \wedge \{I_a, I_b\} \cap \{I_1,\ldots,I_n\} = \emptyset \wedge I_s \neq I_t$

(7a) $\texttt{\$}I_1,\ldots,\texttt{\$}I_n,\texttt{\$}I_a,\texttt{\$}I_b,\texttt{\#}I_s,\texttt{\#}I_t$ do not occur in F and S

From (2), (3), and (4), we know (b.1).

To show (b.2), we take arbitrary $s' \in \mathit{State}$ and assume

(8) $[\![C_1]\!](s,s')$

(9) $\mathit{throws}(\mathit{control}(s'))$

(10) $key(control(s')) = I_k$

We have to show

(b.2.a) $[\![C_2]\!]_{\mathrm{T}}(write(execute(s'), I_v, value(control(s'))))$

We define

(11) $s'' := write(execute(s'), I_v, value(control(s')))$

From (11), (CD2), (RWE), (TRE), and (WSE), we know

(12) $s'' = s'$ EXCEPT I_v

(13) $read(s'', I_v) = value(control(s'))$

To prove (b.2.a), by (6), (11), and the definition of $[\![\_]\!]$, it suffices to prove for arbitrary but fixed $e \in Environment$

(b.2.a.1) $executes(control(s''))$

(b.2.a.2) $executes(control(s''))$

$\exists v_1, \ldots, v_n, a, b \in Value, c_s, c_t \in Control :$
LET
$e_0 := e[I_s \mapsto c_s, I_t \mapsto c_t]_c[I_1 \mapsto v_1, \ldots, I_n \mapsto v_n][I_a \mapsto a, I_b \mapsto b]$
IN

(b.2.a.3)
$executes(c_s) \wedge$
$throws(c_t) \wedge\ key(c_t) = I_k \wedge read(s'', I_v) = value(c_t) \wedge$
$[\![F[\#I_s/\mathtt{now}][\$I_1/I_1, \ldots, \$I_n/I_n][\$I_a/I_v]]\!](e_0)(s'', s'') \wedge$
$[\![S[\#I_s/\mathtt{now}, \#I_t/\mathtt{next}][\$I_1/I_1, \ldots, \$I_n/I_n]$
$[\$I_a/I_v, \$I_b/I_v{}'][I_1/I_1{}', \ldots, I_n/I_n{}']]\!](e_0)(s'', s'')$

From (11), (CD1), and (CW), we know (b.2.a.1) and (b.2.a.2).

We define

(14)
$e_0 := e[I_s \mapsto control(s), I_t \mapsto control(s')]$
$e_1 := e_0[I_1 \mapsto read(s, I_1), \ldots, I_n \mapsto read(s, I_n)]$
$e_2 := e_1[I_a \mapsto read(s, I_v), I_b \mapsto read(s', I_v)]$

To show (b.2.a.3), it suffices to show

(b.2.a.3.a.1) $executes(control(s))$

(b.2.a.3.a.2) $throws(control(s'))$

(b.2.a.3.a.3) $key(control(s')) = I_k$

(b.2.a.3.a.4) $read(s'', I_v) = value(control(s'))$

(b.2.a.3.a.5) $[\![F[\#I_s/\texttt{now}][\$I_1/I_1, \ldots, \$I_n/I_n][\$I_a/I_v]]\!](e_2)(s'', s'')$

(b.2.a.3.a.6) $[\![S[\#I_s/\texttt{now}, \#I_t/\texttt{next}][\$I_1/I_1, \ldots, \$I_n/I_n]$
$[\$I_a/I_v, \$I_b/I_v{}'][I_1/I_1{}', \ldots, I_n/I_n{}']]\!](e_2)(s'', s'')$

From (2), we know (b.2.a.3.a.1).

From (9), we know (b.2.a.3.a.2).

From (10), we know (b.2.a.3.a.3).

From (13), we know (b.2.a.3.a.4).

From (2), (5), (8), and the definition of $[\![\_]\!]$ and $[\_][\_]$, we know

(15) $[\![S]\!](e)(s, s')$

(16) $s = s'$ EXCEPT $I_1, \ldots, I_n$

From (3) and the definition of $[\![\_]\!]$, we know

(17) $[\![F]\!](e)(s, s)$

From (7a), (14), (17), and (CNOF2), we know

(18) $[\![F[\#I_s/\texttt{now}]]\!](e_0)(s, s)$

From (7a), (14), (15), (CNOF2), and (CNEF2), we know

(19) $[\![S[\#I_s/\texttt{now}, \#I_t/\texttt{next}]]\!](e_0)(s, s')$

We proceed by case distinction:

- Case $I_v \in \{I_1, \ldots, I_n\}$: from the case condition, (12), and (AVE) we know

  (20) $s'' = s'$ EXCEPT $I_1, \ldots, I_n$

  From (16), (20), and (TRE), we know

  (21) $s = s''$ EXCEPT $I_1, \ldots, I_n$

  From (7a), (14), (18), (21) (CNOF0), and (PMVF1'), we know

  (22) $[\![F[\#I_s/\texttt{now}][\$I_1/I_1, \ldots, \$I_n/I_n]]\!](e_1)(s'', s)$

  From the case condition, (22), and (MPVF0'), we know

(23) $[\![F[\#I_s/\texttt{now}][\$I_1/I_1,\ldots,\$I_n/I_n][\$I_a/I_v]]\!](e_1)(s'',s)$

From (7), (7a), (14), (23), and (MPVF0'), we know

(24) $[\![F[\#I_s/\texttt{now}][\$I_1/I_1,\ldots,\$I_n/I_n][\$I_a/I_v]]\!](e_2)(s'',s)$

From (1c) and (24), we know (b.2.a.3.a.5).

From (7a), (14), (19), (21), (CNOF0), and (PMVF1'), we know

(25) $[\![S[\#I_s/\texttt{now},\#I_t/\texttt{next}][\$I_1/I_1,\ldots,\$I_n/I_n]]\!](e_1)(s'',s')$

From the case condition, (25), and (MPVF0'), we know

(26) $[\![S[\#I_s/\texttt{now},\#I_t/\texttt{next}][\$I_1/I_1,\ldots,\$I_n/I_n][\$I_a/I_v]]\!](e_1)(s'',s')$

From (7a), (14), (20), (26), (CNEF0), (PMVF2'), we know

(27) $[\![S[\#I_s/\texttt{now},\#I_t/\texttt{next}][\$I_1/I_1,\ldots,\$I_n/I_n][\$I_a/I_v,\$I_b/I_v{}']]\!](e_2)(s'',s'')$

From (IDE), we know

(28) $s'' = writes(s'',I_1,read(s'',I_1),\ldots,I_n,read(s'',I_n))$

From (27), (28), and (PPVF2'), we know (b.2.a.3.a.6).

- Case $I_v \notin \{I_1,\ldots,I_n\}$: from the case condition, (16), and (RSE), we know

(29) $read(s,I_v) = read(s',I_v)$

From (7a), (14), (16), (18), (CNOF0), and (PMVF1'), we know

(30) $[\![F[\#I_s/\texttt{now}][\$I_1/I_1,\ldots,\$I_n/I_n]]\!](e_1)(s',s)$

From (7), (7a), (12), (14), (29), (30), (CNOF0), and (PMVF1'), we know

(31) $[\![F[\#I_s/\texttt{now}][\$I_1/I_1,\ldots,\$I_n/I_n][\$I_a/I_v]]\!](e_2)(s'',s)$

From (1c) and (31), we know (b.2.a.3.a.5).

From (7a), (14), (16), (19), (CNOF0), and (PMVF1'), we know

(32) $[\![S[\#I_s/\texttt{now},\#I_t/\texttt{next}][\$I_1/I_1,\ldots,\$I_n/I_n]]\!](e_1)(s',s')$

From (7), (7a), (12), (14), (32), (CNOF0), and (PMVF1'), we know

(33) $[\![S[\#I_s/\texttt{now},\#I_t/\texttt{next}][\$I_1/I_1,\ldots,\$I_n/I_n][\$I_a/I_v]]\!](e_2)(s'',s')$

From (14), we know

(34) $e_2 = e_2[I_b \mapsto read(s',I_v)]$

From (7), (7a), (12), (14), (33), (34), (CNEF0), and (PMVF2'), we know

(35) $[\![S[\#I_s/\texttt{now}, \#I_t/\texttt{next}][\$I_1/I_1, \ldots, \$I_n/I_n] [\$I_a/I_v, \$I_b/I_v{}']]\!](e_2)(s'', s'')$

From (IDE), we know

(36) $s'' = writes(s'', I_1, read(s'', I_1), \ldots, I_n, read(s'', I_n))$

From (35), (36), and (PPVF2'), we know (b.2.a.3.a.6). □

5.5.3 While Loop (Without Invariant)

$$\frac{\forall s \in State : [\![F \texttt{ => } \texttt{!}E]\!](s)}{\texttt{while (}E\texttt{) } C \downarrow F}$$

If no invariant is given, the only way to make sure that a loop terminates is to make sure that it is never entered.

Soundness Proof We have to show

(a) $executes(control(s)) \wedge [\![F]\!](s) \Rightarrow [\![\texttt{while (}E\texttt{) } C]\!]_{\mathrm{T}}(s)$

We assume

(2) $executes(control(s))$

(3) $[\![F]\!](s)$

By the definition of $[\![\_]\!]_{\mathrm{T}}$, it suffices to show for arbitrary $t, u \in State^{\infty}, k \in \mathbb{N}$

(a.1) $\neg infiniteExecution(t, u, s, [\![E]\!], [\![C]\!])$

(a.2) $finiteExecution(k, t, u, s, [\![E]\!], [\![C]\!]) \wedge [\![E]\!](t(k)) = \mathrm{TRUE} \wedge (executes(control(u(k))) \vee continues(control(u(k)))) \Rightarrow [\![C]\!]_{\mathrm{T}}(t(k)))$

From the premise and the definition of $[\![\_]\!]$, we know

(4) $\forall s \in State : [\![F]\!](s) \Rightarrow \neg [\![E]\!](s)$

From (3) and (4), we know

(5) $\neg [\![E]\!](s)$

From (5) and the definition of *infiniteExecution*, we know (a.1).

To show (a.2), we assume

(6) $\mathit{finiteExecution}(k,t,u,s,[\![E]\!],[\![C]\!])$

(7) $[\![E]\!](t(k)) = \text{TRUE}$

(8) $\mathit{executes}(\mathit{control}(u(k))) \vee \mathit{continues}(\mathit{control}(u(k)))$

(9) $\neg[\![C]\!]_{\mathrm{T}}(t(k))$

and show a contradiction.

From (5), (6), and the definition of *finiteExecution*, we have a contradiction. □

5.5.4 While Loop (With Invariant)

$E \simeq H$
$C : [S]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}$
$\mathit{Invariant}(G,H,S)_{I_1,\ldots,I_n}$
T has no free variables and no primed program variables
and no occurrence of `now` or `next`

$\forall s \in \mathit{State} : [\![F \Rightarrow G[\texttt{now}/\texttt{next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]]\!](s)$

$C \downarrow$ `EXISTS` $\$I_1,\ldots,\$I_n$ `:` `EXSTATE` $\#I_s,\#I_t$ `:`
$F[\#I_s/\texttt{now}][\$I_1/I_1,\ldots,\$I_n/I_n]$ `AND`
$G[\#I_s/\texttt{now},\#I_t/\texttt{next}]$
$[\$I_1/I_1,\ldots,\$I_n/I_n,I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]$ `AND` H `AND`
$(\#I_t\texttt{.executes}$ `OR` $\#I_t\texttt{.continues})$

$\forall e \in \mathit{Environment}, s,s' \in \mathit{Store}, v_1,\ldots,v_n \in \mathit{Value}, c_s,c_t \in \mathit{Control}$:
LET $e_0 = e[I_s \mapsto c_s, I_t \mapsto c_t]_c[I_1 \mapsto v_1,\ldots,I_n \mapsto v_n]$ IN
$[\![F[\#I_s/\texttt{now}][\$I_1/I_1,\ldots,\$I_n/I_n]$ `AND`
$G[\#I_s/\texttt{now},\#I_t/\texttt{next}]$
$[\$I_1/I_1,\ldots,\$I_n/I_n,I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]$ `AND` H `AND`
$(\#I_t\texttt{.executes}$ `OR` $\#I_t\texttt{.continues})$ `AND`
`now.executes` `AND` $[S]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}$ `AND`
`(next.executes OR next.continues)`
$]\!](e_0)(s,s') \Rightarrow$
LET
$m = [\![T]\!](e_0)(s,s')$,
$m' = [\![T[I_1\texttt{'}/I_1,\ldots,I_n\texttt{'}/I_n]]\!](e_0)(s,s')$
IN $m \in \mathbb{N} \wedge m > m'$

`while (`E`)` $C \downarrow F$

Soundness Proof We have to show

(a) $\mathit{executes}(\mathit{control}(s)) \wedge [\![F]\!](s) \Rightarrow [\![\texttt{while (}E\texttt{) } C]\!]_{\mathrm{T}}(s)$

We assume

(2) $\mathit{executes}(\mathit{control}(s))$

(3) $[\![F]\!](s)$

By the definition of $[\![\_]\!]_{\mathrm{T}}$, it suffices to show for arbitrary $t, u \in \mathit{State}^{\infty}, k \in \mathbb{N}$

(a.1) $\neg \mathit{infiniteExecution}(t, u, s, [\![E]\!], [\![C]\!])$

(a.2) $\mathit{finiteExecution}(k, t, u, s, [\![E]\!], [\![C]\!]) \wedge [\![E]\!](t(k)) = \mathrm{TRUE} \wedge (\mathit{executes}(\mathit{control}(u(k))) \vee \mathit{continues}(\mathit{control}(u(k)))) \Rightarrow [\![C]\!]_{\mathrm{T}}(t(k)))$

From the premises, the soundness of the verification calculus, and the induction hypothesis, we know

(4) $E \simeq H$

(5) $\forall s, s' \in \mathit{State}, e \in \mathit{Environment}:$
$\mathit{executes}(\mathit{control}(s)) \wedge [\![C]\!](s, s') \Rightarrow [\![[S]^{F_c, F_b, F_r, \{K_1, \ldots, K_m\}}_{I_1, \ldots, I_n}]\!](e)(s, s')$

(6) $\mathit{Invariant}(G, H, S)_{I_1, \ldots, I_n}$

(7) T has no free variables and no primed program variables and no occurrence of `now` or `next`

(8) $\forall s \in \mathit{State} : [\![F \texttt{ => } G[\texttt{now/next}][I_1/I_1\texttt{'}, \ldots, I_n/I_n\texttt{'}]]\!](s)$

(9) $\forall s \in \mathit{State}:$
$[\![\texttt{EXISTS \$}I_1, \ldots, \texttt{\$}I_n\texttt{ : EXSTATE \#}I_s, \texttt{\#}I_t\texttt{ :}$
$F[\texttt{\#}I_s\texttt{/now}][\texttt{\$}I_1/I_1, \ldots, \texttt{\$}I_n/I_n]\ \texttt{AND}$
$G[\texttt{\#}I_s\texttt{/now}, \texttt{\#}I_t\texttt{/next}]$
$[\texttt{\$}I_1/I_1, \ldots, \texttt{\$}I_n/I_n, I_1/I_1\texttt{'}, \ldots, I_n/I_n\texttt{'}]\ \texttt{AND}\ H\ \texttt{AND}$
$(\texttt{\#}I_t\texttt{ . executes OR \#}I_t\texttt{ . continues})]\!](s)$
$\Rightarrow [\![C]\!]_{\mathrm{T}}(s)$

$$
\begin{array}{l}
\forall e \in \mathit{Environment}, s, s' \in \mathit{Store}, v_1, \ldots, v_n \in \mathit{Value}, c_s, c_t \in \mathit{Control}: \\
\quad \text{LET } e_0 = e[I_s \mapsto c_s, I_t \mapsto c_t]_c[I_1 \mapsto v_1, \ldots, I_n \mapsto v_n] \text{ IN} \\
\quad\quad [\![F[\#I_s/\texttt{now}][\$I_1/I_1, \ldots, \$I_n/I_n] \texttt{ AND} \\
\quad\quad\quad G[\#I_s/\texttt{now}, \#I_t/\texttt{next}] \\
\quad\quad\quad\quad [\$I_1/I_1, \ldots, \$I_n/I_n, I_1/I_1\texttt{'}, \ldots, I_n/I_n\texttt{'}] \texttt{ AND } H \texttt{ AND} \\
\quad\quad\quad (\#I_t\texttt{.executes OR } \#I_t\texttt{.continues}) \texttt{ AND} \\
\quad\quad\quad \texttt{now.executes AND } [S]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \texttt{ AND} \\
\quad\quad\quad \texttt{(next.executes OR next.continues)} \\
\quad\quad [\!] (e_0)(s,s') \Rightarrow \\
\quad\quad \text{LET} \\
\quad\quad\quad m = [\![T]\!](e_0)(s,s'), \\
\quad\quad\quad m' = [\![T[I_1\texttt{'}/I_1, \ldots, I_n\texttt{'}/I_n]]\!](e_0)(s,s') \\
\quad\quad \text{IN } m \in \mathbb{N} \wedge m > m'
\end{array}
\tag{10}
$$

From (6) and the definition of *Invariant*, we know

(10a) $\$I_1, \ldots, \$I_n, \#I_s, \#I_t$ do not occur in $G, H,$ and F

To show (a.2), we assume

(11) $\mathit{finiteExecution}(k, t, u, s, [\![E]\!], [\![C]\!])$

(12) $[\![E]\!](t(k)) = \text{TRUE}$

(13) $\mathit{executes}(\mathit{control}(u(k))) \vee \mathit{continues}(\mathit{control}(u(k)))$

and show

(a.2.a) $[\![C]\!]_{\text{T}}(t(k))$

From (9), it suffices to show

$$
\begin{array}{l}
[\![\texttt{EXISTS } \$I_1, \ldots, \$I_n\texttt{: EXSTATE } \#I_s, \#I_t\texttt{:} \\
\quad F[\#I_s/\texttt{now}][\$I_1/I_1, \ldots, \$I_n/I_n] \texttt{ AND} \\
\quad G[\#I_s/\texttt{now}, \#I_t/\texttt{next}] \\
\quad\quad [\$I_1/I_1, \ldots, \$I_n/I_n, I_1/I_1\texttt{'}, \ldots, I_n/I_n\texttt{'}] \texttt{ AND } H \texttt{ AND} \\
\quad (\#I_t\texttt{.executes OR } \#I_t\texttt{.continues})]\!](t(k))
\end{array}
\tag{a.2.b}
$$

From the definition of $[\![\_]\!]$, it suffices to show for arbitrary $e \in \mathit{Environment}$

$$
\begin{array}{l}
\exists v_1, \ldots, v_n \in \mathit{Value}, c_s, c_t \in \mathit{Control}: \\
\quad \text{LET } e_0 = e[I_s \mapsto c_s, I_t \mapsto c_t]_c[I_1 \mapsto v_1, \ldots, I_n \mapsto v_n] \text{ IN} \\
\quad\quad [\![F[\#I_s/\texttt{now}][\$I_1/I_1, \ldots, \$I_n/I_n]]\!](e_0)(t(k), t(k)) \wedge \\
\quad\quad [\![G[\#I_s/\texttt{now}, \#I_t/\texttt{next}] \\
\quad\quad\quad [\$I_1/I_1, \ldots, \$I_n/I_n, I_1/I_1\texttt{'}, \ldots, I_n/I_n\texttt{'}]]\!](e_0)(t(k), t(k)) \wedge \\
\quad\quad [\![H]\!](e_0)(t(k), t(k)) \wedge \\
\quad\quad (\mathit{executes}(c_t) \vee \mathit{continues}(c_t))
\end{array}
\tag{a.2.c}
$$

We define

$$(14)\quad \begin{array}{l} e_0 := e[I_s \mapsto \mathit{control}(s), I_t \mapsto \mathit{control}(u(k))]_c \\ e_1 := e_0[I_1 \mapsto \mathit{read}(s, I_1), \ldots, I_n \mapsto \mathit{read}(s, I_n)] \end{array}$$

and show

$$(\text{a.2.c.1})\quad [\![F[\#I_s/\texttt{now}][\$I_1/I_1, \ldots, \$I_n/I_n]]\!](e_1)(t(k), t(k))$$

$$(\text{a.2.c.2})\quad \begin{array}{l} [\![G[\#I_s/\texttt{now}, \#I_t/\texttt{next}] \\ \quad [\$I_1/I_1, \ldots, \$I_n/I_n, I_1/I_1\,', \ldots, I_n/I_n\,']]\!](e_1)(t(k), t(k)) \end{array}$$

$$(\text{a.2.c.3})\quad [\![H]\!](e_1)(t(k), t(k))$$

$$(\text{a.2.c.4})\quad \mathit{executes}(\mathit{control}(u(k))) \vee \mathit{continues}(\mathit{control}(u(k)))$$

From (11) and the definition of *finiteExecution*, we know

$$(15)\quad t(0) = s$$

$$(16)\quad u(0) = s$$

$$(17)\quad \begin{array}{l} \forall i \in \mathbb{N}_k : \\ \quad \neg \mathit{breaks}(\mathit{control}(u(i))) \wedge \mathit{executes}(\mathit{control}(t(i))) \wedge \\ \quad [\![E]\!](t(i)) = \text{TRUE} \wedge [\![C]\!](t(i), u(i+1)) \wedge \\ \quad \text{IF } \mathit{continues}(\mathit{control}(u(i+1))) \vee \mathit{breaks}(\mathit{control}(u(i+1))) \\ \qquad \text{THEN } t(i+1) = \mathit{execute}(u(i+1)) \\ \qquad \text{ELSE } t(i+1) = u(i+1) \end{array}$$

From (5), (17), and the definition of $[\,\_\,]_\_$, we know

$$(18)\quad \begin{array}{l} \forall i \in \mathbb{N}_k : \\ \quad [\![S]\!](e)(t(i), u(i+1)) \wedge \\ \quad t(i) = u(i+1) \text{ EXCEPT } I_1, \ldots, I_n \wedge \\ \quad (t(i+1) = u(i+1) \ \vee \ t(i+1) = \mathit{execute}(u(i+1))) \end{array}$$

From (15), (18), (CD2), (NEQ), (AVE), and (TRE), we know

$$(19)\quad s = t(k) \text{ EXCEPT } I_1, \ldots, I_n$$

From (1c), (3), and the definition of $[\![\,\_\,]\!]$, we know

$$(20)\quad [\![F]\!](e)(s, t(k))$$

From (10a), (14), (20), and (CNOF2), we know

$$(21)\quad [\![F[\#I_s/\texttt{now}]]\!](e_1)(s, t(k))$$

From (10a), (14), (19), (21), (CNOF0), and (PMVF1'), we know (a.2.c.1).

From (3), (8), and the definition of $[\![\_]\!]$, we know

(22) $[\![G[\texttt{now}/\texttt{next}][I_1/I_1\texttt{'}, \ldots, I_n/I_n\texttt{'}]]\!](e)(s,s)$

From (22), (CD0), and (PNNF2), we know

(23) $[\![G[I_1/I_1\texttt{'}, \ldots, I_n/I_n\texttt{'}]]\!](e)(s,s)$

From (23), (PPVF1'), and (IDE), we know

(24) $[\![G]\!](e)(s,s)$

From (4), (5), (6), and (24), we can show (as demonstrated in the soundness proof of the Invariant Rule in Section 5.4.3)

(25) $[\![G]\!](e)(s,u(k))$

From (10a), (14), (25), (CNOF2) and (CNEF2), we know

(27) $[\![G[\#I_s/\texttt{now}, \#I_t/\texttt{next}]]\!](e_0)(s,u(k))$

From (10a), (19), (27), (CNOF0), and (PMVF1'), we know

(28) $[\![G[\#I_s/\texttt{now}, \#I_t/\texttt{next}][\$I_1/I_1, \ldots, \$I_n/I_n]]\!](e_1)(t(k),u(k))$

From (15), (16), (18), and (CD2), we know

(29) $t(k)$ EQUALS $u(k)$

From (29), (NEQ), (WSE), and (IDE), we know

(30) $t(k) = \mathit{writes}(t(k), I_1, \mathit{read}(u(k), I_1), \ldots, I_n, \mathit{read}(u(k), I_n))$

From (28), (30), and (PPVF1'), we know

(31) $[\![G[\#I_s/\texttt{now}, \#I_t/\texttt{next}]$
$[\$I_1/I_1, \ldots, \$I_n/I_n, I_1/I_1\texttt{'}, \ldots, I_n/I_n\texttt{'}]]\!](e_1)(t(k),u(k))$

From (29), (CD0) and (CD4), we know

(32) $t(k) = (\mathit{store}(u(k)), \mathit{control}(t(k)))$

From (31), (32), (CNEF0) and (PVFNE), we know (a.2.c.2).

From (4), (12), and the definition of $\simeq$, we know (a.2.c.3).

From (13), we have (a.2.c.4).

To show (a.1), we assume

(33) $\mathit{infiniteExecution}(t,u,s,[\![E]\!],[\![C]\!])$

and show a contradiction. We take an arbitrary $e \in \mathit{Environment}$ and define

(34) $m : \mathit{Value}^{\infty}, m(i) = [\![T]\!](e)(t(i),u(i+1))$

Since $\langle \mathbb{N}, > \rangle$ is a well-founded ordering, it suffices to show

(a.1.b) $\forall i \in \mathbb{N} : m(i) \in \mathbb{N} \wedge m(i) > m(i+1)$

Take arbitrary $i \in \mathbb{N}$. We show

(a.1.b.1) $m(i) \in \mathbb{N}$

(a.1.b.2) $m(i) > m(i+1)$

We define

(35) $$\begin{array}{l} e_0 := e[I_s \mapsto \mathit{control}(s), I_t \mapsto \mathit{control}(u(i))]_c \\ e_1 := e[I_1 \mapsto \mathit{read}(s,I_1), \ldots, I_n \mapsto \mathit{read}(s,I_n)] \end{array}$$

From (10), (35), and the definition of $[\![\_]\!]$, we know

(36) $$\begin{array}{l} ([\![F[\#I_s/\mathtt{now}][\$I_1/I_1,\ldots,\$I_n/I_n]]\!](e_1)(t(i),u(i+1)) \wedge \\ \quad [\![G[\#I_s/\mathtt{now},\#I_t/\mathtt{next}][\$I_1/I_1,\ldots,\$I_n/I_n,I_1/I_1{}',\ldots,I_n/I_n{}']]\!] \\ \qquad (e_1)(t(i),u(i+1)) \wedge \\ \quad [\![H]\!](e_1)(t(i),u(i+1)) \wedge \\ \quad (\mathit{executes}(\mathit{control}(u(i))) \vee \mathit{continues}(\mathit{control}(u(i)))) \wedge \\ \quad \mathit{executes}(\mathit{control}(t(i))) \wedge \\ \quad [\![[S]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}]\!](e_1)(t(i),u(i+1)) \wedge \\ \quad (\mathit{executes}(\mathit{control}(u(i+1))) \vee \mathit{continues}(\mathit{control}(u(i+1)))) \Rightarrow \\ \mathrm{LET} \\ \quad m = [\![T]\!](e_1)(t(i),u(i+1)), \\ \quad m' = [\![T[\mathtt{next}/\mathtt{now}][I_1{}'/I_1,\ldots,I_n{}'/I_n]]\!](e_1)(t(i),u(i+1)) \\ \mathrm{IN}\ m \in \mathbb{N} \wedge m > m' \end{array}$$

From (3) and the definition of $[\![\_]\!]$, we know

(37) $[\![F]\!](e)(s,s)$

From (33), and the definition of *infiniteExecution*, we know

(38) $t(0) = s$

(39) $u(0) = s$

(40)
$$\begin{array}{l}\forall i \in \mathbb{N}: \\ \quad \neg \mathit{breaks}(\mathit{control}(u(i))) \wedge \mathit{executes}(\mathit{control}(t(i))) \wedge \\ \quad [\![E]\!](t(i)) = \text{TRUE} \wedge [\![C]\!](t(i), u(i+1)) \wedge \\ \quad \text{IF } \mathit{continues}(\mathit{control}(u(i+1))) \vee \mathit{breaks}(\mathit{control}(u(i+1))) \\ \quad\quad \text{THEN } t(i+1) = \mathit{execute}(u(i+1)) \\ \quad\quad \text{ELSE } t(i+1) = u(i+1)\end{array}$$

From (5), (40), and the definition of $[\_]_{\_}$, we know

(41) $\forall i \in \mathbb{N}: [\![[S]_{I_1,\ldots,I_n}^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}]\!](e_1)(t(i), u(i+1))$

(42) $\forall i \in \mathbb{N}: t(i) = u(i+1)$ EXCEPT $I_1, \ldots, I_n$

(43) $\forall i \in \mathbb{N}: t(i+1) = u(i+1) \vee t(i+1) = \mathit{execute}(u(i+1))$

From (42), (43), (CD2), (AVE), and (TRE), we know

(44) $s = t(i)$ EXCEPT $I_1, \ldots, I_n$

(45) $s = t(i+1)$ EXCEPT $I_1, \ldots, I_n$

From (1c) and (37), we know

(46) $[\![F]\!](e)(s, u(i+1))$

From (10a), (35), (46), and (CNOF2), we know

(47) $[\![F[\#I_s/\texttt{now}]]\!](e_0)(s, u(i+1))$

From (10a), (35), (44), (47), (CNOF0), and (PMVF1'), we know

(48) $[\![F[\#I_s/\texttt{now}][\$I_1/I_1, \ldots, \$I_n/I_n]]\!](e_1)(t(i), u(i+1))$

From (4), (5), (6), and (24), we can show (as presented in the soundness proof the Invariant Rule in Section 5.4.3)

(49) $[\![G]\!](e)(s, u(i))$

From (6), (10a), (35), (49), (CNOF2), and (CNEF2), we know

(50) $[\![G[\#I_s/\texttt{now}, \#I_t/\texttt{next}]]\!](e_0)(s, u(i))$

From (6), (10a), (35), (44), (50), (CNOF0), and (PMVF1'), we know

(51) $[\![G[\#I_s/\texttt{now}, \#I_t/\texttt{next}][\$I_1/I_1, \ldots, \$I_n/I_n]]\!](e_1)(t(i), u(i))$

From (15), (16), (18), and (CD2), we know

(52) $t(i)$ EQUALS $u(i)$

From (52), (NEQ), (WSE), and (IDE), we know

(53) $u(i) = writes(t(i), I_1, read(t(i), I_1), \ldots, I_n, read(t(i), I_n))$

From (51), (53), and (PPVF2'), we know

(54) $[\![G[\#I_s/\texttt{now}, \#I_t/\texttt{next}][\$I_1/I_1, \ldots, \$I_n/I_n, I_1/I_1\texttt{'}, \ldots, I_n/I_n\texttt{'}]]\!]$
$(e_1)(t(i), t(i))$

From (54), (CNEF0), and (PVFNE), we know

(55) $[\![G[\#I_s/\texttt{now}, \texttt{now}/\texttt{next}][\$I_1/I_1, \ldots, \$I_n/I_n, I_1/I_1\texttt{'}, \ldots, I_n/I_n\texttt{'}]]\!]$
$(e_1)(t(i), (store(t(i)), control(u(i+1))))$

From (CD3), (NEQ), and (AVE), we know

(56) $t(i) = (store(t(i)), control(u(i+1)))$ EXCEPT $I_1, \ldots, I_n$

From (42), (56), and (TRE), we know

(57) $u(i+1) = (store(t(i)), control(u(i+1))$ EXCEPT $I_1, \ldots, I_n$

From (55), (57), (PPVF0'), and (PVF4'), we know

(58) $[\![G[\#I_s/\texttt{now}, \#I_t/\texttt{next}][\$I_1/I_1, \ldots, \$I_n/I_n, I_1/I_1\texttt{'}, \ldots, I_n/I_n\texttt{'}]]\!]$
$(e_1)(t(i), u(i+1))$

From (4), (40), and the definition of $\simeq$, we know

(59) $[\![H]\!](e_1)(t(i), u(i+1))$

From (2), (16), (40), we can show

(60) $\forall i \in \mathbb{N} : executes(control(u(i))) \vee continues(control(u(i)))$

We define

(61) $m_0 = [\![T]\!](e_1)(t(i), u(i+1))$

(62) $m_1 = [\![T[\texttt{next/now}][I_1\,'/I_1, \ldots, I_n\,'/I_n]]\!](e_1)(t(i), u(i+1))$

From (36), (40), (41), (48), (58), (59), (60), (61), (62), we know

(63) $m_0 \in \mathbb{N}$

(64) $m_0 > m_1$

From (7), (61), and (MVT'), we know

(65) $m_0 = [\![T]\!](e)(t(i), u(i+1))$

From (34), (63), and (65), we know (a.1.b.1).

We define

(66) $s_0 := \mathit{writes}(t(i), I_1, \mathit{read}(u(i+1), I_1), \ldots, I_n, \mathit{read}(u(i+1), I_n))$

From (62), (66), and (PPVT1'), we know

(67) $m_1 = [\![T[\texttt{next/now}]]\!](e_1)(s_0, u(i+1))$

From (67) and (PNNT1), we know

(68) $m_1 = [\![T]\!](e_1)((\mathit{store}(s_0), \mathit{control}(u(i+1))), u(i+1))$

From (1a), (66), and (WSE), we know

(69) $s_0 = t(i)$ EXCEPT $I_1, \ldots, I_n$

From (42), (69), and (TRE), we know

(70) $s_0 = u(i+1)$ EXCEPT $I_1, \ldots, I_n$

From (66) and (RWE), we know

(71) $\mathit{read}(s_0, I_1) = \mathit{read}(u(i+1), I_1) \wedge \ldots \wedge$
$\mathit{read}(s_0, I_n) = \mathit{read}(u(i+1), I_n)$

From (70), (71), (RVE), and (NEQ), we know

(72) s_0 EQUALS $u(i+1)$

From (CD3), we know

(73) s_0 EQUALS $(store(s_0), control(u(i+1)))$

From (72), (73), (NEQ), and (TRE), we know

(74) $u(i+1)$ EQUALS $(store(s_0), control(u(i+1)))$

From (68), (74), and (EST'), we know

(75) $m_1 = [\![T]\!](e_1)(u(i+1), u(i+1))$

From (7), (75), and (PVTNE), we know

(76) $m_1 = [\![T]\!](e_1)(u(i+1), (store(u(i+1)), control(u(i+2))))$

From (7), (76), (MVT'), and (PVT2'), we know

(77) $m_1 = [\![T]\!](e)(u(i+1), u(i+2))$

From (7), (77), and (PVTNO), we know

(78) $m_1 = [\![T]\!](e)((store(u(i+1)), control(t(i+1))), u(i+2))$

From (43), (REE), (NEQ), and (CD2), we know

(79) $t(i+1)$ EQUALS $u(i+1)$

From (78), (79), and (EST'), we know

(80) $m_1 = [\![T]\!](e)(t(i+1), u(i+2))$

From (34), (64), (65), and (80), we know (a.1.b.2). □

5.6 Computing Command Preconditions

The computation of preconditions in the presence of (possible) control flow interruptions proceeds analogously to the computation presented in Section 4.1. Pre- and postconditions are not allowed refer to `next` (in addition to the primed program variables). The rules are presented in Figures 5.19 to 5.24. The "generic" rule is a simple extension of the previously presented version; as for the specialized rules, especially the rules for loops have become more complicated.

Precondition Calculus with Interruptions: Judgements

$\text{PRE}(C,Q) = P \Leftrightarrow$
Q has no primed program variables and no occurr. of `next` $\Rightarrow$
P has no primed prog. variables and no occurr. of `next` $\wedge$
$\forall e \in \mathit{Environment}, s, s' \in \mathit{State}$:
$[\![\texttt{now.executes}]\!](e)(s,s) \Rightarrow$
$([\![P]\!](e)(s,s) \wedge [\![C]\!](s,s') \Rightarrow [\![Q]\!](e)(s',s'))$

Precondition Calculus with Interruptions: Generic Rule

$C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}$
$J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$
$\$J_1,\ldots,\$J_n$ do not occur in F and Q
$\#I_s$ does not occur in F and Q

$\text{PRE}(C,Q) =$
`FORALL` $\$J_1,\ldots,\$J_n$ `: ALLSTATE` $\#I_s$ `:`
$F[\#I_s/\texttt{next}][\$J_1/I_1\texttt{'},\ldots,\$J_n/I_n\texttt{'}]$ =>
$Q[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]$

Figure 5.19: The Precondition Calculus of the Command Language (Part 1/6)

Precondition Calculus with Interruptions: Non-Loops

$$\frac{E \simeq T \quad \$J \text{ does not occur in } Q}{\textsc{pre}(I\texttt{=}E, Q) = \texttt{LET } \$J\texttt{=}T \texttt{ IN } Q[\$J/I]}$$

$$\frac{\$J \text{ does not occur in } P \quad \$K \text{ does not occur in } Q \quad J \neq K \quad \textsc{pre}(C, Q[\$K/I]) = P}{\textsc{pre}(\texttt{var } I\texttt{;}\, C, Q) = \texttt{FORALL } \$J : P[\$J/I][I/\$K]}$$

$$\frac{E \simeq T \quad \$J \text{ does not occur in } P \quad \$K \text{ does not occur in } Q \quad J \neq K \quad \textsc{pre}(C, Q[\$K/I]) = P}{\textsc{pre}(\texttt{var } I\texttt{=}E\texttt{;}\, C, Q) = \texttt{LET } \$J\texttt{=}T \texttt{ IN } P[\$J/I][I/\$K]}$$

$$\frac{\textsc{pre}(C_1, \texttt{IF now.executes THEN } P \texttt{ ELSE } Q) = O \quad \textsc{pre}(C_2, Q) = P}{\textsc{pre}(C_1\texttt{;}\, C_2, Q) = O}$$

$$\frac{E \simeq F \quad \textsc{pre}(C, Q) = P}{\textsc{pre}(\texttt{if (}E\texttt{) } C, Q) = \texttt{IF } F \texttt{ THEN } P \texttt{ ELSE } Q}$$

$$\frac{E \simeq F \quad \textsc{pre}(C_1, Q) = P_1 \quad \textsc{pre}(C_2, Q) = P_2}{\textsc{pre}(\texttt{if (}E\texttt{) } C_1 \texttt{ else } C_2, Q) = \texttt{IF } F \texttt{ THEN } P_1 \texttt{ ELSE } P_2}$$

Figure 5.20: The Precondition Calculus of the Command Language (Part 2/6)

Precondition Calculus with Interruptions: Non-Loops

$\#I$ does not occur in Q

$$\frac{}{\text{PRE}(\texttt{continue}, Q) = \texttt{ALLSTATE\#}I : \texttt{\#}I\texttt{.continues => }Q[\texttt{\#}I/\texttt{now}]}$$

$\#I$ does not occur in Q

$$\frac{}{\text{PRE}(\texttt{break}, Q) = \texttt{ALLSTATE\#}I : \texttt{\#}I\texttt{.breaks => }Q[\texttt{\#}I/\texttt{now}]}$$

$\#I$ does not occur in Q

$$\frac{}{\text{PRE}(\texttt{return } E, Q) = \texttt{ALLSTATE\#}I : \texttt{\#}I\texttt{.returns AND \#}I\texttt{.value = }E\texttt{ =>}\ Q[\texttt{\#}I/\texttt{now}]}$$

$\#I_s$ does not occur in Q

$$\frac{}{\text{PRE}(\texttt{throw } I_k\ E, Q) = \texttt{ALLSTATE\#}I_s : \texttt{\#}I_s\texttt{.throws }I_k\texttt{ AND \#}I_s\texttt{.value = }E\texttt{ =>}\ Q[\texttt{\#}I_s/\texttt{now}]}$$

$\#I$ does not occur in P
$\$J$ does not occur in P
$\$K$ does not occur in Q
$J \neq K$

$$\frac{\begin{array}{l}\text{PRE}(C_1,\\ \quad \texttt{IF now.throws }I_k\texttt{ THEN}\\ \quad\quad \texttt{ALLSTATE \#}I\texttt{: \#}I\texttt{.executes =>}\\ \quad\quad\quad \texttt{LET \$}J = \texttt{now.value IN}\\ \quad\quad\quad P[\texttt{\#}I/\texttt{now}][\texttt{\$}J/I_v][I_v/\texttt{\$}K]\\ \quad \texttt{ELSE}\\ \quad\quad Q) = O\\ \text{PRE}(C_2, Q[\texttt{\$}K/I_v]) = P\end{array}}{\text{PRE}(\texttt{try } C_1\ \texttt{catch (}I_k\ I_v\texttt{)}\ C_2, Q) = O}$$

Figure 5.21: The Precondition Calculus of the Command Language (Part 3/6)

Precondition Calculus with Interruptions: Loops

$E \simeq H$
$C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}$
$J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$
$\$J_1,\ldots,\$J_n$ does not occur in Q
$\#I_s$ does not occur in Q

$\text{PRE}(\texttt{while (}E\texttt{) }C, Q) =$
  `FORALL` $\$J_1,\ldots,\$J_n$ `: ALLSTATE` $\#I_s$ `:`
    `!` $\#I_s$`.continues AND !` $\#I_s$`.breaks AND`
    `(` $\#I_s$`.returns =>` F_r`) AND`
    `(` $\#I_s$`.throws =>`
      `(` $\#I_s$`.throws` K_1 `OR ... OR` $\#I_s$`.throws` K_m`)) =>`
    $Q[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]$

$E \simeq H$
$C : [F]^{F_c,\text{FALSE},F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}$
$J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$
$\$J_1,\ldots,\$J_n$ do not occur in Q
$\#I_s$ does not occur in Q

$\text{PRE}(\texttt{while (}E\texttt{) }C, Q) =$
  `FORALL` $\$J_1,\ldots,\$J_n$ `: ALLSTATE` $\#I_s$ `:`
    `!` $\#I_s$`.continues AND !` $\#I_s$`.breaks AND`
    `(` $\#I_s$`.returns =>` F_r`) AND`
    `(` $\#I_s$`.throws =>`
      `(` $\#I_s$`.throws` K_1 `OR ... OR` $\#I_s$`.throws` K_m`)) AND`
    `(` $\#I_s$`.executes =>` $!H[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]$`)`
    `=>` $Q[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]$

Figure 5.22: The Precondition Calculus of the Command Language (Part 4/6)

Precondition Calculus with Interruptions: Loops

$E \simeq H$
$C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}$
$\mathit{Invariant}(G,H,F)_{I_1,\ldots,I_n}$
$J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$
$\$J_1,\ldots,\$J_n$ do not occur in Q
$I_s \neq I_t$
$\#I_s$ does not occur in Q

```
PRE(while (E) C,Q) =
   FORALL $J1,...,$Jn: ALLSTATE #Is:
      !#Is.continues AND !#Is.breaks AND
      (#Is.returns => Fr) AND
      (#Is.throws =>
         (#Is.throws K1 OR ... OR #Is.throws Km)) AND
      (G[now/next][I1/I1',...,In/In'] =>
         EXISTS #It: G[#It/next][$J1/I1',...,$Jn/In'] AND
            IF #It.continues OR #It.breaks
               THEN #Is.executes
               ELSE #Is == #It) =>
      Q[#Is/now][$J1/I1,...,$Jn/In]
```

Figure 5.23: The Precondition Calculus of the Command Language (Part 5/6)

Precondition Calculus with Interruptions: Loops

$E \simeq H$
$C : [F]^{F_c,\texttt{FALSE},F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}$
$\mathit{Invariant}(G,H,F)_{I_1,\ldots,I_n}$
$J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$
$\$J_1,\ldots,\$J_n$ does not occur in Q
$I_s \neq I_t$
$\#I_s$ does not occur in Q

$\texttt{PRE}(\texttt{while (}E\texttt{) }C, Q) =$
  FORALL $\$J_1,\ldots,\$J_n$: ALLSTATE $\#I_s$:
    ! $\#I_s$. continues AND ! $\#I_s$. breaks AND
    ($\#I_s$. returns => F_r) AND
    ($\#I_s$. throws =>
      ($\#I_s$. throws K_1 OR ... OR $\#I_s$. throws K_m)) AND
    (G[now/next][I_1/I_1' ,..., I_n/I_n'] =>
      EXISTS $\#I_t : G[\#I_t/\texttt{next}][\$J_1/I_1', \ldots, \$J_n/I_n']$ AND
        IF $\#I_t$. continues OR $\#I_t$. breaks
          THEN $\#I_s$. executes
          ELSE $\#I_s == \#I_t$) AND
    ($\#I_s$. executes => $!H[\#I_s/\texttt{now}][\$J_1/I_1, \ldots, \$J_n/I_n]$)
    => $Q[\#I_s/\texttt{now}][\$J_1/I_1, \ldots, \$J_n/I_n]$

Figure 5.24: The Precondition Calculus of the Command Language (Part 6/6)

Theorem (Soundness of the Precondition Calculus with Interruptions) Assume the condition denoted by *DifferentVariables*. If $\text{PRE}(C,Q) = P$ can be derived from the rules of the precondition calculus of the command language, then it is true that

$$
\begin{array}{l}
Q \text{ has no primed program variables and no occurr. of } \texttt{next} \Rightarrow \\
\quad P \text{ has no primed program variables and no occurr. of } \texttt{next} \wedge \\
\quad \forall e \in \mathit{Environment}, s, s' \in \mathit{State} : \\
\qquad [\![\texttt{now.executes}]\!](e)(s,s) \Rightarrow \\
\qquad\quad ([\![P]\!](e)(s,s) \wedge [\![C]\!](s,s') \Rightarrow [\![Q]\!](e)(s',s'))
\end{array}
$$

Proof Assume

(1a) *DifferentVariables*

Take C, Q, and P such that $\text{PRE}(C,Q) = P$ can be derived and assume

(1b) Q has no primed program variables and no occurrence of `next`

From (1b) and the rules, it is easy to show by induction on the derivation of $\text{PRE}(C,Q) = P$ that P has no primed program variables and also no occurrence of `next`.

Now take arbitrary $e \in \mathit{Environment}$ and $s, s' \in \mathit{State}$. We prove

$$
\begin{array}{l}
[\![\texttt{now.executes}]\!](e)(s,s) \Rightarrow \\
\quad ([\![P]\!](e)(s,s) \wedge [\![C]\!](s,s') \Rightarrow [\![Q]\!](e)(s',s'))
\end{array}
$$

by induction on the derivation of $\text{PRE}(C,Q) = P$. The following subsections cover all cases for the last step of such a derivation.

From (1b) and the rules, we can immediately deduce that in every derivation $\text{PRE}(C',Q') = P'$ matching the premise of a rule with conclusion $\text{PRE}(C,Q) = P$, the formula Q' has no primed variables; we thus assume in the proofs that the induction hypothesis immediately implies the core claim $\forall e \in \mathit{Environment}, s, s' \in \mathit{State} : [\![P']\!](e)(s,s) \wedge [\![C']\!](s,s') \Rightarrow [\![Q']\!](e)(s',s')$. □

Actually, we only show proofs for those cases where interruptions play a role; the other proofs are analogous to those for programs without interruptions.

5.6.1 Generic Rule

$$\frac{\begin{array}{l} C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \\ J_1,\ldots,J_n \text{ is a renaming of } I_1,\ldots,I_n \\ \$J_1,\ldots,\$J_n \text{ do not occur in } F \text{ and } Q \\ \#I_s \text{ does not occur in } F \text{ and } Q \end{array}}{\begin{array}{l} \text{PRE}(C,Q) = \\ \quad \texttt{FORALL } \$J_1,\ldots,\$J_n \texttt{: ALLSTATE } \#I_s \texttt{:} \\ \quad\quad F[\#I_s/\texttt{next}][\$J_1/I_1\texttt{'},\ldots,\$J_n/I_n\texttt{'}] \texttt{ =>} \\ \quad\quad Q[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n] \end{array}}$$

Soundness Proof We have to show

$$\begin{array}{ll} & [\![\texttt{now.executes}]\!](e)(s,s) \wedge \\ & [\![\texttt{FORALL } \$J_1,\ldots,\$J_n \texttt{: ALLSTATE } \#I_s \texttt{:} \\ & \quad F[\#I_s/\texttt{next}][\$J_1/I_1\texttt{'},\ldots,\$J_n/I_n\texttt{'}] \texttt{ =>} \\ \text{(a)} & \quad Q[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e)(s,s) \wedge \\ & [\![C]\!](s,s') \\ & \Rightarrow \\ & [\![Q]\!](e)(s',s') \end{array}$$

We assume

$$\begin{array}{ll} \text{(2)} & [\![\texttt{now.executes}]\!](e)(s,s) \\ & [\![\texttt{FORALL } \$J_1,\ldots,\$J_n \texttt{: ALLSTATE } \#I_s \texttt{:} \\ \text{(3)} & \quad F[\#I_s/\texttt{next}][\$J_1/I_1\texttt{'},\ldots,\$J_n/I_n\texttt{'}] \texttt{ =>} \\ & \quad Q[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e)(s,s) \\ \text{(4)} & [\![C]\!](s,s') \end{array}$$

and show

$$\text{(b)} \quad [\![Q]\!](e)(s',s')$$

From the hypotheses, we know

$$\begin{array}{ll} \text{(5)} & C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \\ \text{(6)} & J_1,\ldots,J_n \text{ is a renaming of } I_1,\ldots,I_n \\ \text{(7)} & \$J_1,\ldots,\$J_n \text{ do not occur in } F \text{ and } Q \\ \text{(8)} & \#I_s \text{ does not occur in } F \text{ and } Q \end{array}$$

From (1a), (c), (4), (5), and the soundness of the verification calculus, we know

(9) $[\![[F]_{I_1,\ldots,I_n}^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}]\!](e)(s,s')$

From (9) and the definition of $[\![\_]\!]$, we know

(10) $[\![F]\!](e)(s,s')$

(11) $s = s'$ EXCEPT $I_1,\ldots,I_n$

We define

(12) $e_0 := e[I_s \mapsto \mathit{control}(s')]_c$
$e_1 := e_0[J_1 \mapsto \mathit{read}(s',I_1),\ldots,J_n \mapsto \mathit{read}(s',I_n)]$

From (3), (12), and the definition of $[\![\_]\!]$, we know

(13) $[\![F[\#I_s/\texttt{next}][\$J_1/I_1\texttt{'},\ldots,\$J_n/I_n\texttt{'}]]\!](e_1)(s,s) \Rightarrow$
$[\![Q[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e_1)(s,s)$

From (8), (10), (12), and (CNEF2), we know

(14) $[\![F[\#I_s/\texttt{next}]]\!](e_0)(s,s')$

From (6), (7), (11), (12), (14), (CNEF0), and (PMVF2'), we know

(15) $[\![F[\#I_s/\texttt{next}][\$J_1/I_1\texttt{'},\ldots,\$J_n/I_n\texttt{'}]]\!](e_1)(s,s)$

From (13) and (15), we know

(16) $[\![Q[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e_1)(s,s)$

From (6), (7), (11), (12), (16), (CNOF0), and (PMVF1'), we know

(17) $[\![Q[\#I_s/\texttt{now}]]\!](e_0)(s',s)$

From (12), (17), and (CNOF2), we know

(18) $[\![Q]\!](e)(s',s)$

From (1b), (18), (PVFNE), and (PVF2'), we know (b). □

5.6.2 Command Sequence

$$\frac{\text{PRE}(C_1, \texttt{IF now.executes THEN } P \texttt{ ELSE } Q) = O \qquad \text{PRE}(C_2, Q) = P}{\text{PRE}(C_1 \texttt{;} C_2, Q) = O}$$

Soundness Proof We have to show

(a) $[\![\texttt{now.executes}]\!](e)(s,s) \Rightarrow ([\![O]\!](e)(s,s) \wedge [\![C_1 \texttt{;} C_2]\!](s,s') \Rightarrow [\![Q]\!](e)(s',s'))$

We assume

(2) $[\![\texttt{now.executes}]\!](e)(s,s)$

(3) $[\![O]\!](e)(s,s)$

(4) $[\![C_1 \texttt{;} C_2]\!](s,s')$

and show

(b) $[\![Q]\!](e)(s',s')$

From the premises, we know by the induction hypothesis

(5) $\forall e \in \mathit{Environment}, s, s' \in \mathit{State}:$
$[\![\texttt{now.executes}]\!](e)(s,s) \Rightarrow ([\![O]\!](e)(s,s) \wedge [\![C_1]\!](s,s') \Rightarrow [\![\texttt{IF now.executes THEN } P \texttt{ ELSE } Q]\!](e)(s',s'))$

(6) $\forall e \in \mathit{Environment}, s, s' \in \mathit{State}:$
$[\![\texttt{now.executes}]\!](e)(s,s) \Rightarrow ([\![P]\!](e)(s,s) \wedge [\![C_2]\!](s,s') \Rightarrow [\![Q]\!](e)(s',s'))$

From (4) and the definition of $[\![\_]\!]$, we know for some $s_0 \in \mathit{State}$

(7) $[\![C_1]\!](s,s_0)$

(8) IF $\mathit{executes}(\mathit{control}(s_0))$ THEN $[\![C_2]\!](s_0,s')$ ELSE $s' = s_0$

From (2), (3), (5), and (7), we know

(9) $[\![\texttt{IF now.executes THEN } P \texttt{ ELSE } Q]\!](e)(s_0,s_0)$

From (9), we know by the definition of $[\![\_]\!]$

(10) IF *executes*(*control*(s_0)) THEN $[\![P]\!](e)(s_0,s_0)$ ELSE $[\![Q]\!](e)(s_0,s_0)$

We proceed by case distinction.

- Case *executes*(*control*(s_0)): from this condition, (8), and (10), we know

  (11) $[\![C_2]\!](s_0,s')$
  (12) $[\![P]\!](e)(s_0,s_0)$

  From the case condition and the definition of $[\![\_]\!]$, we know

  (13) $[\![\texttt{now.executes}]\!](e)(s_0,s_0)$

  From (6), (11), (12), and (13), we know (b).

- Case $\neg$*executes*(*control*(s_0)): from this condition, (8), and (10), we know

  (14) $s' = s_0$
  (15) $[\![Q]\!](e)(s_0,s_0)$

  From (14) and (15), we know (b). □

5.6.3 Continue Loop

$$\frac{\#I \text{ does not occur in } Q}{\begin{array}{l}\text{PRE}(\texttt{continue},Q) = \\ \quad \texttt{ALLSTATE\#}I : \#I\texttt{.continues => }Q[\#I/\texttt{now}]\end{array}}$$

Soundness Proof We have to show

(a)
$$\begin{array}{l}[\![\texttt{now.executes}]\!](e)(s,s) \Rightarrow \\ \quad ([\![\texttt{ALLSTATE\#}I : \#I\texttt{.continues => }Q[\#I/\texttt{now}]]\!](e)(s,s) \wedge \\ \quad [\![\texttt{continue}]\!](s,s') \Rightarrow \\ \quad\quad [\![Q]\!](e)(s',s'))\end{array}$$

We assume

(2) $[\![\texttt{now.executes}]\!](e)(s,s)$
(3) $[\![\texttt{ALLSTATE\#}I : \#I\texttt{.continues => }Q[\#I/\texttt{now}]]\!](e)(s,s)$
(4) $[\![\texttt{continue}]\!](s,s')$

and show

(b) $[\![Q]\!](e)(s',s')$

From the assumption, we know

(5) $\#I$ does not occur in Q

From (3), (4), and the definition of $[\![\_]\!]$, we know

(6) $\forall c \in \mathit{Control} : \mathit{continues}(c) \Rightarrow [\![Q[\#I/\texttt{now}]]\!](e[I \mapsto c]_c)(s,s)$

(7) $s' = \mathit{continue}(s)$

From (7) and (CD1), we know

(8) $\mathit{continues}(\mathit{control}(s'))$

From (6) and (8), we know

(9) $[\![Q[\#I/\texttt{now}]]\!](e[I \mapsto \mathit{control}(s')]_c)(s,s)$

From (1b), (9), (PVF2'), and (PVFNE), we know

(10) $[\![Q[\#I/\texttt{now}]]\!](e[I \mapsto \mathit{control}(s')]_c)(s,s')$

From (10), (CNOF0), and (PVFNO), we know

(11) $[\![Q[\#I/\texttt{now}]]\!](e[I \mapsto \mathit{control}(s')]_c)((\mathit{store}(s),\mathit{control}(s')),s')$

From (7) and (CD2), we know

(12) s' EQUALS s

From (12), (CD3), (NEQ), (AVE), and (TRE), we know

(13) s' EQUALS $(\mathit{store}(s),\mathit{control}(s'))$

From (11), (13), and (ESF'), we know

(14) $[\![Q[\#I/\texttt{now}]]\!](e[I \mapsto \mathit{control}(s')]_c)(s',s')$

From (5), (14), and (CNOF2), we know (b). □

5.6.4 Break Loop

$$\frac{\#I \text{ does not occur in } Q}{\begin{array}{l}\text{PRE}(\texttt{break}, Q) = \\ \quad \texttt{ALLSTATE\#}I : \texttt{\#}I\texttt{.breaks => } Q[\#I/\texttt{now}]\end{array}}$$

Soundness Proof Analogous to the one given in Section 5.6.3.

5.6.5 Return Result

$$\frac{\#I \text{ does not occur in } Q}{\begin{array}{l}\textsc{pre}(\texttt{return}\ E, Q) = \\ \quad \texttt{ALLSTATE\#}I : \texttt{\#}I\texttt{.returns AND \#}I\texttt{.value} = E \texttt{ =>} \\ \quad\quad Q[\texttt{\#}I/\texttt{now}]\end{array}}$$

Soundness Proof Analogous to the one given in Section 5.6.3.

5.6.6 Throw Exception

$$\frac{\#I \text{ does not occur in } Q}{\begin{array}{l}\textsc{pre}(\texttt{throw}\ I_k\ E, Q) = \\ \quad \texttt{ALLSTATE \#}I_s : \texttt{\#}I_s\texttt{.throws}\ I_k\ \texttt{AND \#}I_s\texttt{.value} = E \texttt{ =>} \\ \quad\quad Q[\texttt{\#}I_s/\texttt{now}]\end{array}}$$

Soundness Proof Analogous to the one given in Section 5.6.3.

5.6.7 Catch Exception

$$\frac{\begin{array}{l}\#I \text{ does not occur in } P \\ \$J \text{ does not occur in } P \\ \$K \text{ does not occur in } Q \\ J \neq K \\ \textsc{pre}(C_1, \\ \quad \texttt{IF now.throws}\ I_k\ \texttt{THEN} \\ \quad\quad \texttt{ALLSTATE \#}I\texttt{ : \#}I\texttt{.executes =>} \\ \quad\quad\quad \texttt{LET \$}J = \texttt{now.value IN} \\ \quad\quad\quad P[\texttt{\#}I/\texttt{now}][\$J/I_v][I_v/\$K] \\ \quad \texttt{ELSE} \\ \quad\quad Q) = O \\ \textsc{pre}(C_2, Q[\$K/I_v]) = P\end{array}}{\textsc{pre}(\texttt{try}\ C_1\ \texttt{catch}\ (I_k\ I_v)\ C_2, Q) = O}$$

Soundness Proof We have to show

(a) $[\![\texttt{now.executes}]\!](e)(s,s) \Rightarrow$
$([\![O]\!](e)(s,s) \wedge [\![\texttt{try}\ C_1\ \texttt{catch}\ (I_k\ I_v)\ C_2]\!](s,s') \Rightarrow [\![Q]\!](e)(s',s'))$

We assume

(2) $[\![\texttt{now.executes}]\!](e)(s,s)$

(3) $[\![O]\!](e)(s,s)$

(4) $[\![\texttt{try}\ C_1\ \texttt{catch}\ (I_k\ I_v)\ C_2]\!](s,s')$

and show

(b) $[\![Q]\!](e)(s',s')$

From the premises, we know by the induction hypothesis

(5) $\#I$ does not occur in P

(6) $\$J$ does not occur in P

(7) $\$K$ does not occur in Q

(8) $J \neq K$

(9)
$\forall e \in \mathit{Environment}, s, s' \in \mathit{State}:$
$[\![\texttt{now.executes}]\!](e)(s,s) \Rightarrow$
$([\![O]\!](e)(s,s) \wedge [\![C_1]\!](s,s') \Rightarrow$
$[\![\texttt{IF now.throws}\ I_k\ \texttt{THEN}$
$\texttt{ALLSTATE}\ \#I\texttt{:}\ \#I\texttt{.executes =>}$
$\texttt{LET}\ \$J = \texttt{now.value}\ \texttt{IN}$
$P[\#I/\texttt{now}][\$J/I_v][I_v/\$K]$
$\texttt{ELSE}$
$Q]\!](e)(s',s'))$

(10)
$\forall e \in \mathit{Environment}, s, s' \in \mathit{State}:$
$[\![\texttt{now.executes}]\!](e)(s,s) \Rightarrow$
$([\![P]\!](e)(s,s) \wedge [\![C_2]\!](s,s') \Rightarrow [\![Q[\$K/I_v]]\!](e)(s',s'))$

From (4) and the definition of $[\![\_]\!]$, we know for some $s_0, s_1, s_2 \in \mathit{State}$

(11) $[\![C_1]\!](s,s_0)$

(12)
IF $\mathit{throws}(\mathit{control}(s_0)) \wedge \mathit{key}(\mathit{control}(s_0)) = I_k$ THEN
$s_1 = \mathit{write}(\mathit{execute}(s_0), I_v, \mathit{value}(\mathit{control}(s_0))) \wedge$
$[\![C_2]\!](s_1,s_2) \wedge$
$s' = \mathit{write}(s_2, I_v, \mathit{read}(s_0, I_v))$
ELSE $s' = s_0$

From (2), (3), (9), and (11), we know

$$
\begin{array}{l}
[\![\,\texttt{IF now.throws } I_k \texttt{ THEN} \\
\quad \texttt{ALLSTATE \#}I\texttt{: \#}I\texttt{.executes =>} \\
\qquad \texttt{LET \$}J = \texttt{now.value IN} \\
\qquad P[\#I/\texttt{now}][\$J/I_v][I_v/\$K] \\
\texttt{ELSE} \\
\quad Q[\![(e)(s_0,s_0)
\end{array}
\tag{13}
$$

From (13) and the definition of $[\![\,\_\,]\!]$, we know

$$
\begin{array}{l}
\text{IF } \mathit{throws}(\mathit{control}(s_0)) \wedge \mathit{key}(\mathit{control}(s_0)) = I_k \text{ THEN} \\
\quad \forall c \in \mathit{Control} : \mathit{executes}(c) \Rightarrow \\
\qquad [\![P[\#I/\texttt{now}][\$J/I_v][I_v/\$K]]\!] \\
\qquad\quad (e[I \mapsto c]_c[J \mapsto \mathit{value}(\mathit{control}(s_0))])(s_0,s_0) \\
\text{ELSE} \\
\quad [\![Q]\!](e)(s_0,s_0)
\end{array}
\tag{14}
$$

We proceed by case distinction.

If $\neg(\mathit{throws}(\mathit{control}(s_0)) \wedge \mathit{key}(\mathit{control}(s_0)) = I_k)$, we know from (12) and (14)

(15) $s' = s_0$

(16) $[\![Q]\!](e)(s_0,s_0)$

From (15) and (16), we know (b).

As for the other case, we assume

(17) $\mathit{throws}(\mathit{control}(s_0))$

(18) $\mathit{key}(\mathit{control}(s_0)) = I_k$

From (12), (17), and (18), we know

(19) $s_1 = \mathit{write}(\mathit{execute}(s_0), I_v, \mathit{value}(\mathit{control}(s_0)))$

(20) $[\![C_2]\!](s_1,s_2)$

(21) $s' = \mathit{write}(s_2, I_v, \mathit{read}(s_0, I_v))$

From (19) and (RW1), we know

(22) $\mathit{read}(s_1, I_v) = \mathit{value}(\mathit{control}(s_0))$

From (19), (CD1), and (CWE), we know

(23) $executes(control(s_1))$

From (10), we know

$$\begin{array}{ll} & [\![\texttt{now.executes}]\!](e[K \mapsto read(s', I_v)])(s_1, s_1) \Rightarrow \\ (24) & \quad ([\![P]\!](e[K \mapsto read(s', I_v)])(s_1, s_1) \wedge [\![C_2]\!](s_1, s_2) \Rightarrow \\ & \quad\quad [\![Q[\$K/I_v]]\!](e[K \mapsto read(s', I_v)])(s_2, s_2)) \end{array}$$

Assume that we can show

(c) $[\![P]\!](e[K \mapsto read(s', I_v)](s_1, s_1)$

From (20), (23), (24), (c) and the definition of $[\![\_]\!]$, we know

(25) $[\![Q[\$K/I_v]]\!](e[K \mapsto read(s', I_v)])(s_2, s_2)$

From (21) and (WS), we know

(26) $s_2 = s' \text{ EXCEPT } I_v$

From (21) and (CW), we know

(27) $control(s_2) = control(s')$

From (7), (25), (26), (27), and (PMVF1"), we know

(28) $[\![Q]\!](e)(s', s_2)$

From (1b), (27), (28), and (PVF2'), we know (b).
It remains to show (c). We define

(29) $P' := P[\#I/\texttt{now}][\$J/I_v]$

From (14), (17), (18), (23), and (29), we know

(30) $[\![P'[I_v/\$K]]\!](e[I \mapsto control(s_1)]_c[J \mapsto value(control(s_0))])(s_0, s_0)$

From (22) and (30), we know

(31) $[\![P'[I_v/\$K]]\!](e[I \mapsto control(s_1)]_c[J \mapsto read(s_1, I_v)])(s_0, s_0)$

From (PMVF0'), we know

(32) $\$K$ does not occur in $P'[I_v/\$K]$

From (29) and (CNOF0), we know

(33) `now` does not occur in $P'[I_v/\$K]$

From (19), (CD2), (NEQ), and (WS), we know

(34) $s_0 = s_1$ EXCEPT I_v

From (31), (32), (33), (RE), and (PMVF1'), we know

(35) $[\![P'[I_v/\$K][\$K/I_v]]\!]$
$(e[I \mapsto \mathit{control}(s_1)]_c[J \mapsto \mathit{read}(s_1, I_v)][K \mapsto \mathit{read}(s_0, I_v)])$
(s_0, s_0)

From (29) and (MPF0'), we know

(36) I_v does not occur in P'

From (36) and (MPVF2'), we know

(37) $P'[I_v/\$K][\$K/I_v] = P'$

From (29), (35), and (37), we know

(38) $[\![P[\#I/\texttt{now}][\$J/I_v]]\!]$
$(e[I \mapsto \mathit{control}(s_1)]_c[J \mapsto \mathit{read}(s_1, I_v)][K \mapsto \mathit{read}(s_0, I_v)])$
(s_0, s_0)

From (8) and (38), we know

(39) $[\![P[\#I/\texttt{now}][\$J/I_v]]\!]$
$(e[I \mapsto \mathit{control}(s_1)]_c[K \mapsto \mathit{read}(s_0, I_v)][J \mapsto \mathit{read}(s_1, I_v)])$
(s_0, s_0)

From (6), (34), (39), (CNOF0), (PMVF1'), we know

(40) $[\![P[\#I/\texttt{now}]]\!](e[I \mapsto \mathit{control}(s_1)]_c[K \mapsto \mathit{read}(s_0, I_v)])(s_1, s_0)$

From (5), (40), and (CNOF2), we know

(41) $[\![P]\!](e[K \mapsto \mathit{read}(s_0, I_v)])(s_1, s_0)$

From (21) and (RW1), we know

(42) $\mathit{read}(s', I_v) = \mathit{read}(s_0, I_v)$

From (41) and (42), we know

(43) $[\![P]\!](e[K \mapsto \mathit{read}(s', I_v)])(s_1, s_0)$

From the last premise and the induction hypothesis, we know

(44) P has no primed variables and no occurrence of `next`

From (43), (44), (PVF2'), and (PVFNE), we have (c). □

5.6.8 While Loop (Without Invariant)

$$\frac{\begin{array}{l} E \simeq H \\ C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \\ J_1,\ldots,J_n \text{ is a renaming of } I_1,\ldots,I_n \\ \$J_1,\ldots,\$J_n \text{ do not occur in } Q \\ \#I_s \text{ does not occur in Q} \end{array}}{\begin{array}{l} \texttt{PRE}(\texttt{while (}E\texttt{) } C, Q) = \\ \quad \texttt{FORALL } \$J_1,\ldots,\$J_n\texttt{: ALLSTATE } \#I_s\texttt{:} \\ \quad\quad !\#I_s\texttt{.continues AND } !\#I_s\texttt{.breaks AND} \\ \quad\quad \texttt{(}\#I_s\texttt{.returns => } F_r\texttt{) AND} \\ \quad\quad \texttt{(}\#I_s\texttt{.throws =>} \\ \quad\quad\quad \texttt{(}\#I_s\texttt{.throws } K_1 \texttt{ OR ... OR } \#I_s\texttt{.throws } K_m\texttt{)) =>} \\ \quad\quad Q[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n] \end{array}}$$

If the loop is not provided with an invariant and the loop may be terminated by a `break`, we do not know which values the variables potentially changed by the loop have in the state in which the loop terminates. However, we know that the terminating state is not breaking or continuing and that it may only be returning or throwing if the loop body allows this. To ensure a certain postcondition, the precondition must thus state that the postcondition holds in every possible termination state for all possible values of the potentially changed variables.

Soundness Proof We have to show

$$\begin{array}{ll} & [\![\texttt{now.executes}]\!](e)(s,s) \Rightarrow \\ & \quad [\![\texttt{FORALL } \$J_1,\ldots,\$J_n\texttt{: ALLSTATE } \#I_s\texttt{:} \\ & \quad\quad !\#I_s\texttt{.continues AND } !\#I_s\texttt{.breaks AND} \\ & \quad\quad \texttt{(}\#I_s\texttt{.returns => } F_r\texttt{) AND} \\ \text{(a)} & \quad\quad \texttt{(}\#I_s\texttt{.throws =>} \\ & \quad\quad\quad \texttt{(}\#I_s\texttt{.throws } K_1 \texttt{ OR ... OR } \#I_s\texttt{.throws } K_m\texttt{)) =>} \\ & \quad\quad Q[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e)(s,s) \wedge \\ & \quad [\![\texttt{while (}E\texttt{) } C]\!](s,s') \Rightarrow \\ & \quad\quad [\![Q]\!](e)(s',s') \end{array}$$

We assume

$$\text{(2)} \quad [\![\texttt{now.executes}]\!](e)(s,s)$$

(3)
$$\begin{array}{l}[\![\texttt{FORALL \$}J_1,\ldots,\texttt{\$}J_n\texttt{: ALLSTATE \#}I_s\texttt{:}\\ \quad\texttt{!\#}I_s\texttt{.continues AND !\#}I_s\texttt{.breaks AND}\\ \quad\texttt{(\#}I_s\texttt{.returns => }F_r\texttt{) AND}\\ \quad\texttt{(\#}I_s\texttt{.throws =>}\\ \quad\quad\texttt{(\#}I_s\texttt{.throws }K_1\texttt{ OR ... OR \#}I_s\texttt{.throws }K_m\texttt{)) =>}\\ \quad Q[\texttt{\#}I_s/\texttt{now}][\texttt{\$}J_1/I_1,\ldots,\texttt{\$}J_n/I_n]][\![e]\!](s,s)\end{array}$$

(4) $[\![\texttt{while(}E\texttt{) }C]\!](s,s')$

and show

(b) $[\![Q]\!](e)(s',s')$

From the premises, we know

(5) $E \simeq H$

(6) $C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}$

(7) $J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$

(8) $\texttt{\$}J_1,\ldots,\texttt{\$}J_n$ do not occur in Q

(9) $\texttt{\#}I_s$ does not occur in Q

From (4) and the definition of $[\![\_]\!]$, we know for some $k \in \mathbb{N}, t, u \in State^\infty$:

(10) $finiteExecution(k,t,u,s,[\![E]\!],[\![C]\!])$

(11)
$$\begin{array}{l}[\![E]\!](t(k)) \neq \text{TRUE} \vee\\ \neg(executes(control(u(k))) \vee continues(control(u(k))))\end{array}$$

(12) $t(k) = s'$

We define

(13)
$$\begin{array}{l}e_0 := e[I_s \mapsto control(s')]_c\\ e_1 := e_0[J_1 \mapsto read(s',I_1),\ldots,J_n \mapsto read(s',I_n)]\end{array}$$

From (3), (13), and the definition of $[\![\_]\!]$, we know

(14)
$$\begin{array}{l}\neg continues(control(s')) \wedge \neg breaks(control(s')) \wedge\\ (returns(control(s')) \Rightarrow [\![F_c]\!](e_1)(s,s)) \wedge\\ (throws(control(s')) \Rightarrow\\ \quad (key(control(s')) = K_1 \vee \ldots \vee key(control(s')) = K_m)) \Rightarrow\\ [\![Q[\texttt{\#}I_s/\texttt{now}][\texttt{\$}J_1/I_1,\ldots,\texttt{\$}J_n/I_n]]\!](e_1)(s,s)\end{array}$$

From (10), (12), the definition of *finiteExecution*, and Lemma "State Control Predicates", we can show

(15) $executes(control(s')) \vee returns(control(s')) \vee throws(control(s'))$

From (15) and Lemma "State Control Predicates", we know

(16) $\neg continues(control(s')) \wedge \neg breaks(control(s'))$

From (1a), (2), (6), and the soundness of the verification calculus, we can show

(17)
$$\begin{array}{l} \forall s, s' \in State : [\![C]\!](s,s') \Rightarrow \\ \quad (returns(control(s')) \Rightarrow [\![F_c]\!](e_1)(s',s'))) \wedge \\ \quad (throws(control(s')) \Rightarrow \\ \qquad (key(control(s')) = K_1 \vee \ldots \vee key(control(s')) = K_m)) \end{array}$$

From (10), (12), (17) and the definition of *finiteExecution*, we can show

(18) $returns(control(s')) \Rightarrow [\![F_c]\!](e_1)(s,s)$

(19)
$$\begin{array}{l} throws(control(s')) \Rightarrow \\ \quad (key(control(s')) = K_1 \vee \ldots \vee key(control(s')) = K_m) \end{array}$$

From (14), (16), (18), and (19), we know

(20) $[\![Q[\#I_s/\mathtt{now}][\$J_1/I_1, \ldots, \$J_n/I_n]]\!](e_1)(s,s)$

From (1a), (2), (4), (5), (6) , and the soundness of the verification calculus, we can prove (as demonstrated in the proof of the soundness of the basic rule for loops with breaks on page 199)

(21) $s = s'$ EXCEPT $I_1, \ldots, I_n$

From (7), (8), (13), (20), (21), (CNOF0), and (PMVF1'), we know

(22) $[\![Q[\#I_s/\mathtt{now}]]\!](e_0)(s',s)$

From (9), (13), (22), and (CNOF2), we know

(23) $[\![Q]\!](e)(s',s)$

From (1b), (23), (PVF2'), and (PVFNE), we know (b). □

5.6.9 While Loop (Without Invariant, No Break)

$E \simeq H$
$C : [F]^{F_c,\texttt{FALSE},F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}$
$J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$
$\$J_1,\ldots,\$J_n$ do not occur in Q
$\#I_s$ does not occur in Q

$\texttt{PRE}(\texttt{while(}E\texttt{) }C, Q) =$
  FORALL $\$J_1,\ldots,\$J_n$: ALLSTATE $\#I_s$:
    ! $\#I_s$. continues AND ! $\#I_s$. breaks AND
    ($\#I_s$. returns => F_r) AND
    ($\#I_s$. throws =>
      ($\#I_s$. throws K_1 OR ... OR $\#I_s$. throws K_m)) AND
    ($\#I_s$. executes => $!H[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]$)
    => $Q[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]$

Soundness Proof We have to show

(a)
$[\![\texttt{now.executes}]\!](e)(s,s) \Rightarrow$
  $[\![$FORALL $\$J_1,\ldots,\$J_n$: ALLSTATE $\#I_s$:
    ! $\#I_s$. continues AND ! $\#I_s$. breaks AND
    ($\#I_s$. returns => F_r) AND
    ($\#I_s$. throws =>
      ($\#I_s$. throws K_1 OR ... OR $\#I_s$. throws K_m)) AND
    ($\#I_s$. executes => $!H[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]$) =>
    $Q[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e)(s,s) \wedge$
  $[\![\texttt{while(}E\texttt{) }C]\!](s,s') \Rightarrow$
    $[\![Q]\!](e)(s',s')$

We assume

(2) $[\![\texttt{now.executes}]\!](e)(s,s)$

(3)
$[\![$FORALL $\$J_1,\ldots,\$J_n$: ALLSTATE $\#I_s$:
  ! $\#I_s$. continues AND ! $\#I_s$. breaks AND
  ($\#I_s$. returns => F_r) AND
  ($\#I_s$. throws =>
    ($\#I_s$. throws K_1 OR ... OR $\#I_s$. throws K_m)) AND
  ($\#I_s$. executes => $!H[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]$) =>
  $Q[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e)(s,s)$

(4) $[\![\texttt{while(}E\texttt{) }C]\!](s,s')$

and show

(b) $[\![Q]\!](e)(s',s')$

From the premises, we know

(5) $E \simeq H$

(6) $C : [\, F \,]^{F_c,\mathtt{FALSE},F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}$

(7) $J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$

(8) $\$J_1,\ldots,\$J_n$ do not occur in Q

(9) $\#I_s$ does not occur in Q

From (4) and the definition of $[\![\_]\!]$, we know for some $k \in \mathbb{N}, t, u \in \mathit{State}^\infty$:

(10) $\mathit{finiteExecution}(k,t,u,s,[\![E]\!],[\![C]\!])$

(11) $[\![E]\!](t(k)) \neq \mathrm{TRUE} \vee$
$\neg(\mathit{executes}(\mathit{control}(u(k))) \vee \mathit{continues}(\mathit{control}(u(k))))$

(12) $t(k) = s'$

We define

(13) $e_0 := e[I_s \mapsto \mathit{control}(s')]_c$
$e_1 := e_0[J_1 \mapsto \mathit{read}(s',I_1),\ldots,J_n \mapsto \mathit{read}(s',I_n)]$

From (3), (13), and the definition of $[\![\_]\!]$, we know

(14)
$$\begin{array}{l}
\neg\mathit{continues}(\mathit{control}(s')) \wedge \neg\mathit{breaks}(\mathit{control}(s')) \wedge \\
(\mathit{returns}(\mathit{control}(s')) \Rightarrow [\![F_c]\!](e_1)(s,s))) \wedge \\
(\mathit{throws}(\mathit{control}(s')) \Rightarrow \\
\quad (\mathit{key}(\mathit{control}(s')) = K_1 \vee \ldots \vee \mathit{key}(\mathit{control}(s')) = K_m)) \wedge \\
(\mathit{executes}(\mathit{control}(s')) \Rightarrow \\
\quad \neg[\![H[\#I_s/\mathtt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e_1)(s,s)) \Rightarrow \\
[\![Q[\#I_s/\mathtt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e_1)(s,s)
\end{array}$$

From (10), (12), the definition of *finiteExecution*, and Lemma "State Control Predicates", we can show

(15) $\mathit{executes}(\mathit{control}(s')) \vee \mathit{returns}(\mathit{control}(s')) \vee \mathit{throws}(\mathit{control}(s'))$

From (15) and Lemma "State Control Predicates", we know

(16) $\neg continues(control(s')) \wedge \neg breaks(control(s'))$

From (1a), (2), (6), and the soundness of the verification calculus, we can show

$$\begin{array}{ll} & \forall s, s' \in State : [\![C]\!](s,s') \Rightarrow \\ & \quad \neg breaks(control(s')) \wedge \\ (17) & \quad (returns(control(s')) \Rightarrow [\![F_c]\!](e_1)(s',s')) \wedge \\ & \quad (throws(control(s')) \ \Rightarrow \\ & \qquad (key(control(s')) = K_1 \vee \ldots \vee key(control(s')) = K_m)) \end{array}$$

From (10), (12), (17) and the definition of *finiteExecution*, we can show

(18) $returns(control(s')) \Rightarrow [\![F_c]\!](e_1)(s,s)$

$$\begin{array}{ll} (19) & throws(control(s')) \ \Rightarrow \\ & \quad (key(control(s')) = K_1 \vee \ldots \vee key(control(s')) = K_m) \end{array}$$

Assume we can show

$$\begin{array}{ll} (c) & executes(control(s')) \Rightarrow \\ & \quad \neg [\![H[\#I_s/\mathtt{now}][\$J_1/I_1, \ldots, \$J_n/I_n]]\!](e_1)(s,s) \end{array}$$

From (14), (16), (18), (19), and (c), we know

(20) $[\![Q[\#I_s/\mathtt{now}][\$J_1/I_1, \ldots, \$J_n/I_n]]\!](e_1)(s,s)$

From (1a), (2), (4), (5), (6) , and the soundness of the verification calculus, we can prove (as demonstrated in the proof of the soundness of the basic rule for loops with breaks on page 199)

(21) $s = s'$ EXCEPT $I_1, \ldots, I_n$

From (7), (8), (13), (20), (21), (CNOF0), and (PMVF1'), we know

(22) $[\![Q[\#I_s/\mathtt{now}]]\!](e_0)(s',s)$

From (9), (13), (22), and (CNOF2), we know

(23) $[\![Q]\!](e)(s',s)$

From (1b), (23), (PVF2'), and (PVFNE), we know (b).

It remains to show (c). We assume

(24) $executes(control(s'))$

and show

(d) $\neg[\![H[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e_1)(s,s)$

From (12) and (24), we know

(25) $\mathit{executes}(\mathit{control}(t(k)))$

From (2), (10), (17), and the definitions of $[\![\_]\!]$, and *finiteExecution*, we know

(26) $\neg\mathit{breaks}(\mathit{control}(u(k)))$

From (10), (25), (26), and the definition of *finiteExecution*, we know

(27) $\mathit{executes}(\mathit{control}(u(k))) \vee \mathit{continues}(\mathit{control}(u(k)))$

From (11), (12), and (27), we know

(28) $[\![E]\!](s') \neq \text{TRUE}$

From (5), (28), and the definition of $\simeq$, we know

(29) $\neg[\![H]\!](e)(s',s')$

From (5) and the definition of $\simeq$, we know

(30) H has no free variables

(31) H has no primed program variables

(32) H has no occurrence of `next`

From (13), (29), (30), and (CNOF2), we know

(33) $\neg[\![H[\#I_s/\texttt{now}]]\!](e_0)(s',s')$

From (7), (13), (21), (30), (33), (CNOF0), and (PMVF1'), we know

(34) $\neg[\![H[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e_1)(s,s')$

From (31), (32), (34), (PVF2'), and (PVFNE), we know (d). □

5.6.10 While Loop (With Invariant)

$E \simeq H$
$C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}$
$Invariant(G,H,F)_{I_1,\ldots,I_n}$
$J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$
$\$J_1,\ldots,\$J_n$ do not free in Q
$I_s \neq I_t$
$\#I_s$ does not occur in Q

$\text{PRE}(\texttt{while (}E\texttt{) }C, Q) =$
  FORALL $\$J_1,\ldots,\$J_n$: ALLSTATE $\#I_s$:
    ! $\#I_s$ .continues AND ! $\#I_s$ .breaks AND
    ($\#I_s$ .returns => F_r) AND
    ($\#I_s$ .throws =>
      ($\#I_s$ .throws K_1 OR ... OR $\#I_s$ .throws K_m)) AND
    (G[now/next][I_1/I_1' ,..., I_n/I_n'] =>
      EXISTS $\#I_t : G$[$\#I_t$/next][$\$J_1/I_1'$,..., $\$J_n/I_n'$] AND
        IF $\#I_t$ .continues OR $\#I_t$ .breaks
          THEN $\#I_s$ .executes
          ELSE $\#I_s$ == $\#I_t$) =>
    Q[$\#I_s$/now][$\$J_1/I_1,\ldots,\$J_n/I_n$]

Soundness Proof We have to show

(a)
$[\![$now.executes$]\!](e)(s,s) \Rightarrow$
  $[\![$FORALL $\$J_1,\ldots,\$J_n$: ALLSTATE $\#I_s$:
    ! $\#I_s$ .continues AND ! $\#I_s$ .breaks AND
    ($\#I_s$ .returns => F_r) AND
    ($\#I_s$ .throws =>
      ($\#I_s$ .throws K_1 OR ... OR $\#I_s$ .throws K_m)) AND
    (G[now/next][I_1/I_1' ,..., I_n/I_n'] =>
      EXISTS $\#I_t : G$[$\#I_t$/next][$\$J_1/I_1'$,..., $\$J_n/I_n'$] AND
        IF $\#I_t$ .continues OR $\#I_t$ .breaks
          THEN $\#I_s$ .executes
          ELSE $\#I_s$ == $\#I_t$) =>
    Q[$\#I_s$/now][$\$J_1/I_1,\ldots,\$J_n/I_n$]$]\!](e)(s,s) \wedge$
  $[\![$while (E) C$]\!](s,s') \Rightarrow$
    $[\![Q]\!](e)(s',s')$

We assume

(2) $[\![$now.executes$]\!](e)(s,s)$

$$
\begin{array}{ll}
 & [\![\texttt{FORALL } \$J_1,\ldots,\$J_n\texttt{: ALLSTATE } \#I_s\texttt{:} \\
 & \quad !\,\#I_s\texttt{.continues AND } !\,\#I_s\texttt{.breaks AND} \\
 & \quad (\#I_s\texttt{.returns => } F_r\texttt{) AND} \\
 & \quad (\#I_s\texttt{.throws =>} \\
 & \qquad (\#I_s\texttt{.throws } K_1\texttt{ OR ... OR } \#I_s\texttt{.throws } K_m\texttt{)) AND} \\
(3) & \quad (G[\texttt{now/next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]\texttt{ =>} \\
 & \qquad \texttt{EXISTS } \#I_t : G[\#I_t/\texttt{next}][\$J_1/I_1\texttt{'},\ldots,\$J_n/I_n\texttt{'}]\texttt{ AND} \\
 & \qquad\quad \texttt{IF } \#I_t\texttt{.continues OR } \#I_t\texttt{.breaks} \\
 & \qquad\qquad \texttt{THEN } \#I_s\texttt{.executes} \\
 & \qquad\qquad \texttt{ELSE } \#I_s == \#I_t) \texttt{ =>} \\
 & \quad Q[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e)(s,s)
\end{array}
$$

(4) $[\![\texttt{while(}E\texttt{) } C]\!](s,s')$

and show

(b) $[\![Q]\!](e)(s',s')$

From the premises, we know

(5) $E \simeq H$

(6) $C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}$

(7) $\mathit{Invariant}(G,H,F)_{I_1,\ldots,I_n}$

(8) $J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$

(9) $\$J_1,\ldots,\$J_n$ do not occur in Q

(9a) $I_s \neq I_t$

(10) $\#I_s$ does not occur in Q

From (4) and the definition of $[\![\_]\!]$, we know for some $k \in \mathbb{N}, t, u \in \mathit{State}^\infty$:

(11) $\mathit{finiteExecution}(k,t,u,s,[\![E]\!],[\![C]\!])$

(12) $[\![E]\!](t(k)) \neq \text{TRUE} \vee$
$\neg(\mathit{executes}(\mathit{control}(u(k))) \vee \mathit{continues}(\mathit{control}(u(k))))$

(13) $t(k) = s'$

We define

(14) $e_0 := e[I_s \mapsto \mathit{control}(s')]_c$
$e_1 := e_0[J_1 \mapsto \mathit{read}(s',I_1),\ldots,J_n \mapsto \mathit{read}(s',I_n)]$

From (3), (14), and the definition of $[\![\_]\!]$, we know

$$
\begin{array}{ll}
 & \neg \mathit{continues}(\mathit{control}(s')) \wedge \neg \mathit{breaks}(\mathit{control}(s')) \wedge \\
 & (\mathit{returns}(\mathit{control}(s')) \Rightarrow [\![F_c]\!](e_1)(s,s)) \wedge \\
 & (\mathit{throws}(\mathit{control}(s')) \ \Rightarrow \\
 & \quad (\mathit{key}(\mathit{control}(s')) = K_1 \vee \ldots \vee \mathit{key}(\mathit{control}(s')) = K_m)) \wedge \\
 & ([\![G[\texttt{now/next}][I_1/I_1\texttt{'}, \ldots, I_n/I_n\texttt{'}]]\!](e_1)(s,s) \Rightarrow \\
 & \quad [\![\texttt{EXISTS \#}I_t : G[\texttt{\#}I_t/\texttt{next}][\texttt{\$}J_1/I_1\texttt{'}, \ldots, \texttt{\$}J_n/I_n\texttt{'}] \texttt{ AND} \\
 & \qquad \texttt{IF \#}I_t\texttt{.continues OR \#}I_t\texttt{.breaks} \\
 & \qquad\quad \texttt{THEN \#}I_s\texttt{.executes} \\
 & \qquad\quad \texttt{ELSE \#}I_s == \texttt{\#}I_t]\!](e_1)(s,s)) \ \Rightarrow \\
 & [\![Q[\texttt{\#}I_s/\texttt{now}][\texttt{\$}J_1/I_1, \ldots, \texttt{\$}J_n/I_n]]\!](e_1)(s,s)
\end{array} \tag{15}
$$

As demonstrated in Section 5.6.8 in the soundness proof for the loop without invariant, we can show

$$\neg \mathit{continues}(\mathit{control}(s')) \wedge \neg \mathit{breaks}(\mathit{control}(s')) \tag{16}$$

$$\mathit{returns}(\mathit{control}(s')) \Rightarrow [\![F_c]\!](e_1)(s,s) \tag{17}$$

$$
\begin{array}{ll}
 & \mathit{throws}(\mathit{control}(s')) \ \Rightarrow \\
 & \quad (\mathit{key}(\mathit{control}(s')) = K_1 \vee \ldots \vee \mathit{key}(\mathit{control}(s')) = K_m)
\end{array} \tag{18}
$$

We now show

$$
\begin{array}{ll}
 & [\![G[\texttt{now/next}][I_1/I_1\texttt{'}, \ldots, I_n/I_n\texttt{'}]]\!](e_1)(s,s) \Rightarrow \\
 & \quad [\![\texttt{EXISTS \#}I_t : G[\texttt{\#}I_t/\texttt{next}][\texttt{\$}J_1/I_1\texttt{'}, \ldots, \texttt{\$}J_n/I_n\texttt{'}] \texttt{ AND} \\
 & \qquad \texttt{IF \#}I_t\texttt{.continues OR \#}I_t\texttt{.breaks} \\
 & \qquad\quad \texttt{THEN \#}I_s\texttt{.executes} \\
 & \qquad\quad \texttt{ELSE \#}I_s == \texttt{\#}I_t]\!](e_1)(s,s)
\end{array} \tag{b.1}
$$

We assume

$$[\![G[\texttt{now/next}][I_1/I_1\texttt{'}, \ldots, I_n/I_n\texttt{'}]]\!](e_1)(s,s) \tag{19}$$

and show

$$
\begin{array}{l}
[\![\texttt{EXISTS \#}I_t : G[\texttt{\#}I_t/\texttt{next}][\texttt{\$}J_1/I_1\texttt{'}, \ldots, \texttt{\$}J_n/I_n\texttt{'}] \texttt{ AND} \\
\quad \texttt{IF \#}I_t\texttt{.continues OR \#}I_t\texttt{.breaks} \\
\qquad \texttt{THEN \#}I_s\texttt{.executes} \\
\qquad \texttt{ELSE \#}I_s == \texttt{\#}I_t]\!](e_1)(s,s)
\end{array} \tag{b.1.a}
$$

We define

$$e_2 := e_1[I_t \mapsto \mathit{control}(u(k))]_c \tag{20}$$

From (9a), (20) and the definition of $[\![\_]\!]$, to show (b.1.a), it suffices to show

(b.1.b.1) $[\![G[\#I_t/\texttt{next}][\$J_1/I_1\texttt{'}, \ldots, \$J_n/I_n\texttt{'}]]\!](e_2)(s,s)$
IF $\mathit{continues}(\mathit{control}(u(k))) \vee \mathit{breaks}(\mathit{control}(u(k)))$
(b.1.b.2) THEN $\mathit{executes}(\mathit{control}(s'))$
ELSE $\mathit{control}(s') = \mathit{control}(u(k))$

From (19), (CD0), and (PNNF2), we know

(21) $[\![G[I_1/I_1\texttt{'}, \ldots, I_n/I_n\texttt{'}]]\!](e_1)(s,s)$

From (21), (IDE), and (PPVF2'), we know

(22) $[\![G]\!](e_1)(s,s)$

From (4), (5), (6), and (22), we can show (as demonstrated in the soundness proof of the Invariant Rule in Section 5.4.3)

(23) $[\![G]\!](e_1)(s,u(k))$

From (7) and the definition of *Invariant*, we know

(24) $\$I_1, \ldots, \$I_n, \#I_s, \#I_t$ do not occur in $G, H,$ and F

From (20), (23), (24), and (CNEF2), we know

(25) $[\![G[\#I_t/\texttt{next}]]\!](e_2)(s,u(k))$

From (2), (11), and the definitions of $[\![\_]\!]$ and *finiteExecution*, we know

(26) $\mathit{executes}(\mathit{control}(t(0)))$

(27) $t(0) = u(0)$

(28) $\forall i \in \mathbb{N}_k$:
IF $\mathit{continues}(\mathit{control}(u(i+1))) \vee \mathit{breaks}(\mathit{control}(u(i+1)))$
THEN $t(i+1) = \mathit{execute}(u(i+1))$
ELSE $t(i+1) = u(i+1)$

From (27) and (28), we can conclude

(29) $t(k) = u(k) \vee t(k) = \mathit{execute}(u(k))$

From (29), (NEQ), (REE), and (CD2), we know

(30) $t(k)$ EQUALS $u(k)$

From (13), (30), (NEQ), and (AVE), we know

(31) $read(s', I_1) = read(u(k), I_1) \wedge \ldots \wedge read(s', I_n) = read(u(k), I_n)$

From (11) and the definition of *finiteExecution*, we can show

(32) $s = u(k)$ EXCEPT $I_1, \ldots, I_n$

From (8), (14), (20), (24), (25), (31), (32), (CNEF0), and finally (PMVF2'), we know (b.1.b.1).

From Lemma "State Control Predicates", (13), (26), (27), and (28), we can conclude (b.1.b.2).

From (15), (16), (17), (18), and (b.1), we know

(33) $[\![Q[\#I_s/\texttt{now}][\$J_1/I_1, \ldots, \$J_n/I_n]]\!](e_1)(s, s)$

From (13), (30), (32), (NEQ), (AVE), and (TRE), we know

(34) $s = s'$ EXCEPT $I_1, \ldots, I_n$

From (8), (9), (14), (33), (34), (CNOF0), and (PMVF1'), we know

(35) $[\![Q[\#I_s/\texttt{now}]]\!](e_0)(s', s)$

From (10), (14), (35), and (CNOF2), we know

(36) $[\![Q]\!](e)(s', s)$

From (1b), (36), (PVF2'), and (PVFNE), we know (b). □

5.6.11 While Loop (With Invariant, No Break)

$E \simeq H$
$C : [F]^{F_c,\texttt{FALSE},F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}$
$\mathit{Invariant}(G,H,F)_{I_1,\ldots,I_n}$
$J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$
$\$J_1,\ldots,\$J_n$ do not occur in Q
$I_s \neq I_t$
$\#I_s$ does not occur in Q

$\textsc{pre}(\texttt{while(}E\texttt{) } C, Q) =$
  FORALL $\$J_1,\ldots,\$J_n$: ALLSTATE $\#I_s$:
    ! $\#I_s$. continues AND ! $\#I_s$. breaks AND
    ($\#I_s$. returns => F_r) AND
    ($\#I_s$. throws =>
      ($\#I_s$. throws K_1 OR ... OR $\#I_s$. throws K_m)) AND
    (G[now/next][I_1/I_1', ..., I_n/I_n'] =>
      EXISTS $\#I_t$: G[$\#I_t$/next][$\$J_1$/$I_1$', ..., $\$J_n$/$I_n$'] AND
        IF $\#I_t$. continues OR $\#I_t$. breaks
          THEN $\#I_s$. executes
          ELSE $\#I_s$ == $\#I_t$) AND
    ($\#I_s$. executes => !H[$\#I_s$/now][$\$J_1$/$I_1$, ..., $\$J_n$/$I_n$])
    => Q[$\#I_s$/now][$\$J_1$/$I_1$, ..., $\$J_n$/$I_n$]

Soundness Proof Analogous to the proof of the rule without invariant or breaks combined with the proof of the rule with invariant. □

5.7 Computing Command Postconditions

As for preconditions, the computation of postconditions in the presence of (possible) control flow interruptions proceeds analogously to the computation presented in Section 4.2. Pre- and postconditions are not allowed refer to `next` (in addition to the primed program variables). The rules are presented in Figures 5.25 to 5.29. The "generic" rule is a simple extension of the previously presented version; as for the specialized rules, especially the rules for loops have become more complicated.

Theorem (Soundness of the Postcondition Calculus with Interruptions) Assume the condition denoted by *DifferentVariables*. If $\textsc{post}(C,P) = Q$ can be de-

Postcondition Calculus with Interruptions: Judgements

$\text{POST}(C,P) = Q \Leftrightarrow$
P has no primed program variables and no occurr. of `next` $\Rightarrow$
Q has no primed prog. variables and no occurr. of `next` $\wedge$
$\forall e \in \mathit{Environment}, s, s' \in \mathit{State}$:
$[\![\texttt{now.executes}]\!](e)(s,s) \Rightarrow$
$([\![P]\!](e)(s,s) \wedge [\![C]\!](s,s') \Rightarrow [\![Q]\!](e)(s',s'))$

Postcondition Calculus with Interruptions: Generic Rule

$C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}$
$J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$
$\$J_1,\ldots,\$J_n$ do not occur in F and P
$\#I_s$ does not occur in F and P

$\text{POST}(C,P) =$
`EXISTS` $\$J_1,\ldots,\$J_n$ **:** `EXSTATE` $\#I_s$ **:**
$P[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]$ `AND`
$F[\#I_s/\texttt{now},\texttt{now/next}]$
$[\$J_1/I_1,\ldots,\$J_n/I_n, I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]$

Figure 5.25: The Postcondition Calculus of the Command Language (Part 1/5)

Postcondition Calculus with Interruptions: Non-Loops

$$\frac{E \simeq T \quad \$J \text{ does not occur in } P}{\texttt{POST}(I\texttt{=}E, P) = \texttt{EXISTS } \$J : P[\$J/I] \texttt{ AND } I\texttt{=}T[\$J/I]}$$

$$\frac{\$J \text{ does not occur in } P \quad \$K \text{ does not occur in } Q \quad J \neq K \quad \texttt{POST}(C, P[\$J/I]) = Q}{\texttt{POST}(\texttt{var } I\texttt{;}\, C, P) = \texttt{EXISTS } \$K\texttt{:}\, Q[\$K/I][I/\$J]}$$

$$\frac{E \simeq T \quad \$J \text{ does not occur in } P \text{ and in } T \quad \$K \text{ does not occur in } Q \quad J \neq K \quad \texttt{POST}(C, P[\$J/I] \texttt{ AND } I = T[\$J/I]) = Q}{\texttt{POST}(\texttt{var } I\texttt{=}E\texttt{;}\, C, P) = \texttt{EXISTS } \$K\texttt{:}\, Q[\$K/I][I/\$J]}$$

$$\frac{\texttt{POST}(C_1, P) = Q \quad \texttt{POST}(C_2, Q) = O}{\texttt{POST}(C_1\texttt{;}\, C_2, P) = O \texttt{ OR (}Q \texttt{ AND !now.executes)}}$$

$$\frac{C_1 : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \quad \texttt{POST}(C_1, P) = Q \quad \texttt{POST}(C_2, Q) = O}{\begin{array}{l}\texttt{POST}(C_1\texttt{;}\, C_2, P) = O \texttt{ OR (}Q \texttt{ AND} \\ \quad \texttt{((now.continues AND } F_c\texttt{) OR} \\ \quad\quad \texttt{(now.breaks AND } F_b\texttt{) OR} \\ \quad\quad \texttt{(now.returns AND } F_r\texttt{) OR} \\ \quad\quad \texttt{(now.throws } K_1 \texttt{ OR ... OR now.throws } K_m\texttt{)))}\end{array}}$$

$$\frac{E \simeq F \quad \texttt{POST}(C, P \texttt{ AND } F) = Q}{\texttt{POST}(\texttt{if (}E\texttt{) } C, P) = Q \texttt{ OR (}P \texttt{ AND !}F\texttt{)}}$$

$$\frac{E \simeq F \quad \texttt{POST}(C_1, P \texttt{ AND } F) = Q_1 \quad \texttt{POST}(C_2, P \texttt{ AND !}F) = Q_2}{\texttt{POST}(\texttt{if (}E\texttt{) } C_1 \texttt{ else } C_2, P) = Q_1 \texttt{ OR } Q_2}$$

Figure 5.26: The Postcondition Calculus of the Command Language (Part 2/5)

Postcondition Calculus with Interruptions: Non-Loops

$\#I$ does not occur in P

$\text{POST}(\texttt{continue}, P) =$
$\quad \texttt{now.continues AND EXSTATE } \#I\texttt{: } P[\#I/\texttt{now}]$

$\#I$ does not occur in P

$\text{POST}(\texttt{break}, P) =$
$\quad \texttt{now.breaks AND EXSTATE } \#I\texttt{:} P[\#I/\texttt{now}]$

$E \simeq T$
$\#I$ does not occur in P

$\text{POST}(\texttt{return } E, P) =$
$\quad \texttt{now.returns AND now.value} = T \texttt{ AND}$
$\quad \texttt{EXSTATE } \#I\texttt{:} P[\#I/\texttt{now}]$

$E \simeq T$
$\#I_s$ does not occur in P

$\text{POST}(\texttt{throw } I_k\ E, Q) =$
$\quad \texttt{now.throws } I_k \texttt{ AND now.value} = T \texttt{ AND}$
$\quad \texttt{EXSTATE } \#I_s\texttt{: } P[\#I_s/\texttt{now}]$

$\#I$ does not occur in Q
$\$J$ does not occur in Q
$\$K$ does not occur in O
$J \neq K$
$\text{POST}(C_1, P) = Q$
$\text{POST}(C_2,$
$\quad \texttt{EXSTATE } \#I\texttt{:}$
$\quad\quad \#I\texttt{.throws } I_k \texttt{ AND } \#I\texttt{.value} = I_v \texttt{ AND}$
$\quad\quad Q[\#I/\texttt{now}][\$J/I_v]) = O$

$\text{POST}(\texttt{try } C_1 \texttt{ catch } (I_k\ I_v)\ C_2, P) =$
$\quad (Q \texttt{ AND !now.throws } I_k) \texttt{ OR}$
$\quad \texttt{EXISTS } \$K\texttt{: } O[\$K/I_v][I_v/\$J]$

Figure 5.27: The Postcondition Calculus of the Command Language (Part 3/5)

Postcondition Calculus with Interruptions: Loops

$C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}$
$J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$
$\$J_1,\ldots,\$J_n$ do not occur in P
$\#I_s$ does not occur in P

$\text{POST}(\texttt{while(}E\texttt{) }C, P) =$
$\quad \texttt{EXISTS } \$J_1,\ldots,\$J_n\texttt{: EXSTATE } \#I_s\texttt{:}$
$\quad\quad P[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]$

$E \simeq H$
$C : [F]^{F_c,\texttt{FALSE},F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}$
$J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$
$\$J_1,\ldots,\$J_n$ do not occur in P
$\#I_s$ does not occur in P

$\text{POST}(\texttt{while(}E\texttt{) }C, P) =$
$\quad \texttt{(now.executes =>!}H\texttt{) AND}$
$\quad \texttt{EXISTS } \$J_1,\ldots,\$J_n\texttt{: EXSTATE } \#I_s\texttt{:}$
$\quad\quad P[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]$

Figure 5.28: The Postcondition Calculus of the Command Language (Part 4/5)

Postcondition Calculus with Interruptions: Loops

$C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}$
$J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$
$\$J_1,\ldots,\$J_n$ do not occur in P
$\#I_s,\#I_t$ do not occur in P
$\mathit{Invariant}(G,H,F)_{I_1,\ldots,I_n}$

$\text{POST}(\texttt{while (}E\texttt{) } C, P) =$
  EXISTS $\$J_1,\ldots,\$J_n$: EXSTATE $\#I_s,\#I_t$:
    $P[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]$ AND
    $(G[\texttt{now}/\texttt{next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]$
      $[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]$ =>
      $G[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]$
        $[\#I_t/\texttt{next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]$ AND
      IF $\#I_t$.continues OR $\#I_t$.breaks
        THEN next.executes
        ELSE next == $\#I_t$)

$E \simeq H$
$C : [F]^{F_c,\text{FALSE},F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}$
$J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$
$\$J_1,\ldots,\$J_n$ do not occur in P
$\#I_s,\#I_t$ do not occur in P
$\mathit{Invariant}(G,H,F)_{I_1,\ldots,I_n}$

$\text{POST}(\texttt{while (}E\texttt{) } C, P) =$
  (now.executes =>!H) AND
  EXISTS $\$J_1,\ldots,\$J_n$: EXSTATE $\#I_s,\#I_t$:
    $P[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]$ AND
    $(G[\texttt{now}/\texttt{next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]$
      $[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]$ =>
      $G[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]$
        $[\#I_t/\texttt{next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]$ AND
      IF $\#I_t$.continues OR $\#I_t$.breaks
        THEN next.executes
        ELSE next == $\#I_t$)

Figure 5.29: The Postcondition Calculus of the Command Language (Part 5/5)

rived from the rules of the precondition calculus of the command language, then it is true that

$$
\begin{array}{l}
P \text{ has no primed program variables and no occurr. of } \texttt{next} \Rightarrow \\
\quad Q \text{ has no primed program variables and no occurr. of } \texttt{next} \wedge \\
\quad \forall e \in \mathit{Environment}, s, s' \in \mathit{State} : \\
\qquad [\![\texttt{now.executes}]\!](e)(s,s) \Rightarrow \\
\qquad\quad ([\![P]\!](e)(s,s) \wedge [\![C]\!](s,s') \Rightarrow [\![Q]\!](e)(s',s'))
\end{array}
$$

Proof Assume

(1a) *DifferentVariables*

Take C, P, and Q such that $\mathrm{POST}(C,P) = Q$ can be derived and assume

(1b) P has no primed program variables and no occurrence of `next`

From (1b) and the rules, it is easy to show by induction on the derivation of $\mathrm{POST}(C,P) = Q$ that Q has no primed program variables and also no occurrence of `next`.

Now take arbitrary $e \in \mathit{Environment}$ and $s, s' \in \mathit{State}$. We prove

$$
\begin{array}{l}
[\![\texttt{now.executes}]\!](e)(s,s) \Rightarrow \\
\quad ([\![P]\!](e)(s,s) \wedge [\![C]\!](s,s') \Rightarrow [\![Q]\!](e)(s',s'))
\end{array}
$$

by induction on the derivation of $\mathrm{POST}(C,P) = Q$. The following subsections cover all cases for the last step of such a derivation.

From (1b) and the rules, we can immediately deduce that in every derivation $\mathrm{POST}(C',P') = Q'$ matching the premise of a rule with conclusion $\mathrm{POST}(C,P) = Q$, the formula P' has no primed variables; we thus assume in the proofs that the induction hypothesis immediately implies the core claim $\forall e \in \mathit{Environment}, s, s' \in \mathit{State} : [\![P']\!](e)(s,s) \wedge [\![C']\!](s,s') \Rightarrow [\![Q']\!](e)(s',s')$. $\square$

Actually, we only show proofs for those cases where interruptions play a role; the other proofs are analogous to those for programs without interruptions.

5.7.1 Generic Rule

$$\frac{\begin{array}{l} C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \\ J_1,\ldots,J_n \text{ is a renaming of } I_1,\ldots,I_n \\ \$J_1,\ldots,\$J_n \text{ do not occur in } F \text{ and } P \\ \#I_s \text{ does not occur in } F \text{ and } P \end{array}}{\begin{array}{l} \text{POST}(C,P) = \\ \quad \texttt{EXISTS } \$J_1,\ldots,\$J_n\texttt{: EXSTATE } \#I_s\texttt{:} \\ \quad\quad P[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]\ \texttt{AND} \\ \quad\quad F[\#I_s/\texttt{now},\texttt{now/next}] \\ \quad\quad\quad [\$J_1/I_1,\ldots,\$J_n/I_n,I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}] \end{array}}$$

Soundness Proof We have to show

$$\text{(a)}\quad \begin{array}{l} [\![\texttt{now.executes}]\!](e)(s,s) \Rightarrow \\ \quad [\![P]\!](e)(s,s) \wedge [\![C]\!](s,s') \Rightarrow \\ \quad\quad [\![\texttt{EXISTS } \$J_1,\ldots,\$J_n\texttt{: EXSTATE } \#I_s\texttt{:} \\ \quad\quad\quad P[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]\ \texttt{AND} \\ \quad\quad\quad F[\#I_s/\texttt{now},\texttt{now/next}] \\ \quad\quad\quad\quad [\$J_1/I_1,\ldots,\$J_n/I_n,I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]]\!](e)(s',s') \end{array}$$

We assume

$$\text{(2)}\quad [\![\texttt{now.executes}]\!](e)(s,s)$$
$$\text{(3)}\quad [\![P]\!](e)(s,s)$$
$$\text{(4)}\quad [\![C]\!](s,s')$$

and show

$$\text{(b)}\quad \begin{array}{l} [\![\texttt{EXISTS } \$J_1,\ldots,\$J_n\texttt{: EXSTATE } \#I_s\texttt{:} \\ \quad P[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]\ \texttt{AND} \\ \quad F[\#I_s/\texttt{now},\texttt{now/next}] \\ \quad\quad [\$J_1/I_1,\ldots,\$J_n/I_n,I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]]\!](e)(s',s') \end{array}$$

We define

$$\text{(5)}\quad \begin{array}{l} e_0 := e[I_s \mapsto \mathit{control}(s)]_c \\ e_1 := e_0[J_1 \mapsto \mathit{read}(s,I_1),\ldots,J_n \mapsto \mathit{read}(s,I_n)] \end{array}$$

From (5) and the definition of $[\![\_]\!]$, it suffices to show

$$\text{(c.1)}\quad [\![P[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e_1)(s',s')$$

(c.2) $[\![F[\#I_s/\mathtt{now}, \mathtt{now}/\mathtt{next}] \\ [\$J_1/I_1, \ldots, \$J_n/I_n, I_1/I_1\,', \ldots, I_n/I_n\,']]\!](e_1)(s', s')$

From the hypotheses, we know

(6) $C : [F]_{I_1,\ldots,I_n}$

(7) $J_1, \ldots, J_n$ is a renaming of $I_1, \ldots, I_n$

(8) $\$J_1, \ldots, \$J_n$ do not occur in F and P

(9) $\#I_s$ does not occur in F and P

From (1a), (2), (4), (6), the soundness of the verification calculus with interruptions, and the definition of $[\![\_]\!]$, we know

(10) $[\![F]\!](e)(s, s')$

(11) $s = s'$ EXCEPT $I_1, \ldots, I_n$

From (3), (5), (9), and (CNOF2), we know

(12) $[\![P[\#I_s/\mathtt{now}]]\!](e_0)(s, s)$

From (5), (7), (8), (11), (12), (CNOF0), and (PMVF1'), we know

(13) $[\![P[\#I_s/\mathtt{now}][\$J_1/I_1, \ldots, \$J_n/I_n]]\!](e_1)(s', s)$

From (1b), (11), (13), (PVF2'), and (PVNE), we know (c.1).

From (5), (9), (10), and (CNOF2), we know

(14) $[\![F[\#I_s/\mathtt{now}]]\!](e_0)(s, s')$

From (14) and (PNNF2), we know

(15) $[\![F[\#I_s/\mathtt{now}, \mathtt{now}/\mathtt{next}]]\!](e_0)(s, (\mathit{store}(s'), \mathit{control}(s)))$

From (7), (8), (11), (15), (CNOF0), and (PMVF1'), we know

(16) $[\![F[\#I_s/\mathtt{now}, \mathtt{now}/\mathtt{next}] \\ [\$J_1/I_1, \ldots, \$J_n/I_n]]\!](e_1)(s', (\mathit{store}(s'), \mathit{control}(s)))$

From (16), (CD0), (CNEF1), and (PVFNE), we know

(17) $[\![F[\#I_s/\mathtt{now}, \mathtt{now}/\mathtt{next}][\$J_1/I_1, \ldots, \$J_n/I_n]]\!](e_1)(s', s')$

From (17), (IDE), and (PPVF2'), we know (c.2). □

5.7.2 Command Sequence (Basic Version)

$$\frac{\text{POST}(C_1, P) = Q \qquad \text{POST}(C_2, Q) = O}{\text{POST}(C_1\,;C_2, P) = O \texttt{ OR (}Q \texttt{ AND !now.executes)}}$$

The postcondition of the corresponding rule for programs without interruptions is extended by the disjunct Q `AND !now.executes`; this reflects the possibility that the execution of C_1 has resulted in a non-executing state such that C_2 is not executed any more and the postcondition Q of C_1 is consequently still valid.

Soundness Proof We have to show

$$\text{(a)} \quad \begin{array}{l} [\![\texttt{now.executes}]\!](e)(s,s) \Rightarrow \\ \quad [\![P]\!](e)(s,s) \wedge [\![C_1\,;C_2]\!](s,s') \Rightarrow \\ \quad\quad [\![O \texttt{ OR (}Q \texttt{ AND !now.executes)}\,]\!](e)(s',s') \end{array}$$

We assume

(2) $[\![\texttt{now.executes}]\!](e)(s,s)$

(3) $[\![P]\!](e)(s,s)$

(4) $[\![C_1\,;C_2]\!](s,s')$

By the definition of $[\![\_]\!]$, it suffices to show

(b) $[\![O]\!](e)(s',s') \vee ([\![Q]\!](e)(s',s') \wedge \neg \mathit{executes}(\mathit{control}(s')))$

From the premises, we know by the induction hypothesis

$$\text{(5)} \quad \begin{array}{l} \forall s, s' \in \mathit{State} : [\![\texttt{now.executes}]\!](e)(s,s) \Rightarrow \\ \quad [\![P]\!](e)(s,s) \wedge [\![C_1]\!](s,s') \Rightarrow [\![Q]\!](e)(s',s') \end{array}$$

$$\text{(6)} \quad \begin{array}{l} \forall s, s' \in \mathit{State} : [\![\texttt{now.executes}]\!](e)(s,s) \Rightarrow \\ \quad [\![Q]\!](e)(s,s) \wedge [\![C_2]\!](s,s') \Rightarrow [\![O]\!](e)(s',s') \end{array}$$

From (4) and the definition of $[\![\_]\!]$, we know for some $s_0 \in \mathit{State}$

(7) $[\![C_1]\!](s,s_0)$

(8) IF $\mathit{executes}(\mathit{control}(s_0))$ THEN $[\![C_2]\!](s_0,s')$ ELSE $s' = s_0$

From (2), (3), (5), and (7), we know

(9) $[\![Q]\!](e)(s_0,s_0)$

We proceed by case distinction.

- Case $\mathit{executes}(\mathit{control}(s_0))$: from the definition of $[\![\_]\!]$ and the case condition, we know

  (10) $[\![\texttt{now.executes}]\!](e)(s_0, s_0)$

  From (8) and the case condition, we know

  (11) $[\![C_2]\!](s_0, s')$

  From (6), (9), (10), and (11), we know

  (12) $[\![O]\!](e)(s', s')$

  From (12), we know (b).

- Case $\neg\mathit{executes}(\mathit{control}(s_0))$: from (8) and the case condition, we know

  (13) $s' = s_0$

  From (9), (13), and the case condition, we know (b). □

5.7.3 Command Sequence (Extended Version)

$$\frac{\begin{array}{l} C_1 : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \\ \textsc{post}(C_1, P) = Q \\ \textsc{post}(C_2, Q) = O \end{array}}{\begin{array}{l} \textsc{post}(C_1\texttt{;}C_2, P) = O \texttt{ OR (}Q \texttt{ AND} \\ \quad \texttt{((now.continues AND }F_c\texttt{) OR} \\ \quad\quad \texttt{(now.breaks AND }F_b\texttt{) OR} \\ \quad\quad \texttt{(now.returns AND }F_r\texttt{) OR} \\ \quad\quad \texttt{(now.throws }K_1\texttt{ OR ... OR now.throws }K_m\texttt{)))} \end{array}}$$

Compared to the basic rule, the extended rule takes into account the information which control flow interruptions may be triggered by the execution of C_1: the original formula `!now.executes` in the second disjunct is replaced by a more accurate description about the control flow interruptions that prevent the execution of C_2 such that the postcondition Q of P_1 is still valid.

Soundness Proof We have to show

(a)
$$
\begin{array}{l}
[\![\texttt{now.executes}]\!](e)(s,s) \Rightarrow \\
\quad [\![P]\!](e)(s,s) \wedge [\![C_1\,;C_2]\!](s,s') \Rightarrow \\
\quad\quad [\![O\ \texttt{OR (}Q\ \texttt{AND} \\
\quad\quad\quad \texttt{((now.continues AND }F_c\texttt{) OR} \\
\quad\quad\quad \texttt{(now.breaks AND }F_b\texttt{) OR} \\
\quad\quad\quad \texttt{(now.returns AND }F_r\texttt{) OR} \\
\quad\quad\quad \texttt{(now.throws }K_1\ \texttt{OR ... OR now.throws }K_m\texttt{)))} \\
\quad\quad [\(s',s')
\end{array}
$$

We assume

(2) $[\![\texttt{now.executes}]\!](e)(s,s)$

(3) $[\![P]\!](e)(s,s)$

(4) $[\![C_1\,;C_2]\!](s,s')$

By the definition of $[\![\_]\!]$, it suffices to show

(b)
$$
\begin{array}{l}
[\![O]\!](e)(s',s') \vee \\
([\![Q]\!](e)(s',s') \wedge \\
\quad (\mathit{continues}(\mathit{control}(s')) \wedge [\![F_c]\!](e)(s',s')) \vee \\
\quad (\mathit{breaks}(\mathit{control}(s')) \wedge [\![F_b]\!](e)(s',s')) \vee \\
\quad (\mathit{returns}(\mathit{control}(s')) \wedge [\![F_r]\!](e)(s',s')) \vee \\
\quad (\mathit{throws}(\mathit{control}(s')) \wedge \\
\quad\quad (\mathit{key}(\mathit{control}(s')) = K_1 \vee \ldots \vee \mathit{key}(\mathit{control}(s')) = K_m)))
\end{array}
$$

From the last two premises, we know by the induction hypothesis

(5)
$$
\begin{array}{l}
\forall s,s' \in \mathit{State} : [\![\texttt{now.executes}]\!](e)(s,s) \Rightarrow \\
\quad [\![P]\!](e)(s,s) \wedge [\![C_1]\!](s,s') \Rightarrow [\![Q]\!](e)(s',s')
\end{array}
$$

(6)
$$
\begin{array}{l}
\forall s,s' \in \mathit{State} : [\![\texttt{now.executes}]\!](e)(s,s) \Rightarrow \\
\quad [\![Q]\!](e)(s,s) \wedge [\![C_2]\!](s,s') \Rightarrow [\![O]\!](e)(s',s')
\end{array}
$$

From (4) and the definition of $[\![\_]\!]$, we know for some $s_0 \in \mathit{State}$

(7) $[\![C_1]\!](s,s_0)$

(8) IF $\mathit{executes}(\mathit{control}(s_0))$ THEN $[\![C_2]\!](s_0,s')$ ELSE $s' = s_0$

From (2), (3), (5), and (7), we know

(9) $[\![Q]\!](e)(s_0,s_0)$

We proceed by case distinction.

- Case $\mathit{executes}(\mathit{control}(s_0))$: from the definition of $[\![\_]\!]$ and the case condition, we know

  (10) $[\![\texttt{now.executes}]\!](e)(s_0, s_0)$

  From (8) and the case condition, we know

  (11) $[\![C_2]\!](s_0, s')$

  From (6), (9), (10), and (11), we know

  (12) $[\![O]\!](e)(s', s')$

  From (12), we know (b).

- Case $\neg\mathit{executes}(\mathit{control}(s_0))$: from (8) and the case condition, we know

  (13) $s' = s_0$

  From (9), (13), and the case condition, we know

  (14) $[\![Q]\!](e)(s', s')$

  From (1a), (2), (7), the first premise and the soundness of the verification calculus with interruptions, we know

  (15) $[\![[F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}]\!](e)(s, s_0)$

  From (15) and the definitions of $[\![\_]\!]$ and $[\_]^{\_}_{\_}$, we know

  (16) $[\![F]\!](e)(s, s_0)$
  (17) $s = s_0$ EXCEPT $I_1, \ldots, I_n$
  (18) $\mathit{continues}(\mathit{control}(s_0)) \Rightarrow [\![F_c]\!](e)(s, s_0)$
  (19) $\mathit{breaks}(\mathit{control}(s_0)) \Rightarrow [\![F_b]\!](e)(s, s_0)$
  (20) $\mathit{returns}(\mathit{control}(s_0)) \Rightarrow [\![F_r]\!](e)(s, s_0)$
  (21) $\mathit{throws}(\mathit{control}(s_0)) \Rightarrow$
  $(\mathit{key}(\mathit{control}(s_0)) = K_1 \vee \ldots \vee \mathit{key}(\mathit{control}(s_0)) = K_m)$

  From (13), (14), (18), (19), (20), (21), and Lemma "Constant Formulas", we know (b). □

5.7.4 Continue Loop

$\#I$ does not occur in P

$\mathrm{POST}(\texttt{continue}, P) =$
$\quad \texttt{now.continues}$ AND EXSTATE $\#I$: $P[\#I/\texttt{now}]$

Soundness Proof We have to show

$$\begin{array}{ll} & [\![\texttt{now.executes}]\!](e)(s,s) \Rightarrow \\ \text{(a)} & \quad [\![P]\!](e)(s,s) \wedge [\![\texttt{continue}]\!](s,s') \Rightarrow \\ & \quad\quad [\![\texttt{now.continues AND EXSTATE \#}I\texttt{: }P[\#I/\texttt{now}]]\!](e)(s',s') \end{array}$$

We assume

(2) $[\![\texttt{now.executes}]\!](e)(s,s)$

(3) $[\![P]\!](e)(s,s)$

(4) $[\![\texttt{continue}]\!](s,s')$

By the definition of $[\![\_]\!]$, it suffices to show

(b.1) $\mathit{continues}(\mathit{control}(s'))$

(b.2) $\exists c \in \mathit{Control} : [\![P[\#I/\texttt{now}]]\!](e[I \mapsto c]_c)(s',s')$

From (4) and the definition of $[\![\_]\!]$, we know

(5) $s' = \mathit{continue}(s)$

From (5) and (CD1), we know (b.1).

We define

(6) $e_0 := e[I \mapsto \mathit{control}(s)]_c$

By (6), to show (b.2), it suffices to show

(b.2.a) $[\![P[\#I/\texttt{now}]]\!](e_0)(s',s')$

From the premise, (3), (6), and (CNOF2), we know

(7) $[\![P[\#I/\texttt{now}]]\!](e_0)(s,s)$

From (1b), (7), (PVF2'), and (PVFNE), we know

(8) $[\![P[\#I/\texttt{now}]]\!](e_0)(s,s')$

From (8), (CNOF0), and (PVFNO), we know

(9) $[\![P[\#I/\texttt{now}]]\!](e_0)((\mathit{store}(s),\mathit{control}(s')),s')$

From (5) and (CD2), we know

(10) s' EQUALS s

From (10), (NEQ), (TRE), and (CD3), we know

(11) s' EQUALS $(\mathit{store}(s),\mathit{control}(s'))$

From (9), (11), (REE), (NEQ), and (ESF), we know (b.2.a). □

5.7.5 Break Loop

$$\frac{\#I \text{ does not occur in } P}{\begin{array}{l}\text{POST}(\texttt{break}, P) = \\ \quad \texttt{now.breaks AND EXSTATE } \#I : P[\#I/\texttt{now}]\end{array}}$$

Soundness Proof We have to show

$$\begin{array}{ll} & [\![\texttt{now.executes}]\!](e)(s,s) \Rightarrow \\ \text{(a)} & \quad [\![P]\!](e)(s,s) \wedge [\![\texttt{break}]\!](s,s') \Rightarrow \\ & \quad\quad [\![\texttt{now.breaks AND EXSTATE } \#I\texttt{: } P[\#I/\texttt{now}]]\!](e)(s',s') \end{array}$$

We assume

(2) $[\![\texttt{now.executes}]\!](e)(s,s)$

(3) $[\![P]\!](e)(s,s)$

(4) $[\![\texttt{break}]\!](s,s')$

By the definition of $[\![\_]\!]$, it suffices to show

(b.1) $\mathit{breaks}(\mathit{control}(s'))$

(b.2) $\exists c \in \mathit{Control} : [\![P[\#I/\texttt{now}]]\!](e[I \mapsto c]_c)(s',s')$

From (4) and the definition of $[\![\_]\!]$, we know

(5) $s' = \mathit{break}(s)$

From (5) and (CD1), we know (b.1).

We define

(6) $e_0 := e[I \mapsto \mathit{control}(s)]_c$

By (6), to show (b.2), it suffices to show

(b.2.a) $[\![P[\#I/\texttt{now}]]\!](e_0)(s',s')$

From the premise, (3), (6), and (CNOF2), we know

(7) $[\![P[\#I/\texttt{now}]]\!](e_0)(s,s)$

From (1b), (7), (PVF2'), and (PVFNE), we know

(8) $[\![P[\#I/\texttt{now}]]\!](e_0)(s,s')$

From (8), (CNOF0), and (PVFNO), we know

(9) $[\![P[\#I/\texttt{now}]]\!](e_0)((\mathit{store}(s),\mathit{control}(s')),s')$

From (5) and (CD2), we know

(10) s' EQUALS s

From (10), (NEQ), (TRE), and (CD3), we know

(11) s' EQUALS $(\mathit{store}(s),\mathit{control}(s'))$

From (9), (11), (REE), (NEQ), and (ESF), we know (b.2.a). □

5.7.6 Return Result

$$\frac{\begin{array}{l}E \simeq T\\ \#I \text{ does not occur in } P\end{array}}{\begin{array}{l}\text{POST}(\texttt{return}\ E,P) =\\ \quad \texttt{now.returns AND now.value} = T\ \texttt{AND}\\ \quad \texttt{EXSTATE}\ \#I\texttt{:}P[\#I/\texttt{now}]\end{array}}$$

Soundness Proof We have to show

(a)
$$\begin{array}{l}[\![\texttt{now.executes}]\!](e)(s,s)\ \Rightarrow\\ \quad [\![P]\!](e)(s,s) \wedge [\![\texttt{return}\ E]\!](s,s') \Rightarrow\\ \quad\quad [\![\texttt{now.returns AND now.value} = T\ \texttt{AND}\\ \quad\quad\quad \texttt{EXSTATE}\ \#I\texttt{:}P[\#I/\texttt{now}]]\!](e)(s',s')\end{array}$$

We assume

(2) $[\![\texttt{now.executes}]\!](e)(s,s)$

(3) $[\![P]\!](e)(s,s)$

(4) $[\![\texttt{return}\ E]\!](s,s')$

By the definition of $[\![\ \_\ ]\!]$, it suffices to show

(b.1) $\mathit{returns}(\mathit{control}(s'))$

(b.2) $\mathit{value}(\mathit{control}(s')) = [\![T]\!](e)(s',s')$

(b.3) $\exists c \in \mathit{Control} : [\![P[\#I/\mathtt{now}]]\!](e[I \mapsto c]_c)(s', s')$

From (4) and the definition of $[\![\_]\!]$, we know

(5) $s' = \mathit{return}(s, [\![E]\!](s))$

From (5) and (CD1), we know (b.1).

From (5) and (CD1), we also know

(6) $\mathit{value}(\mathit{control}(s')) = [\![E]\!](s)$

From the first premise and the definition of $\simeq$, we know

(7) $[\![E]\!](s) = [\![T]\!](e)(s, s')$

From (6) and (7), we know (b.2).

We define

(8) $e_0 := e[I \mapsto \mathit{control}(s)]_c$

By (8), to show (b.3), it suffices to show

(b.3.a) $[\![P[\#I/\mathtt{now}]]\!](e_0)(s', s')$

From the second premise, (3), (8), and (CNOF2), we know

(9) $[\![P[\#I/\mathtt{now}]]\!](e_0)(s, s)$

From (1b), (9), (PVF2'), and (PVFNE), we know

(10) $[\![P[\#I/\mathtt{now}]]\!](e_0)(s, s')$

From (10), (CNOF0), and (PVFNO), we know

(11) $[\![P[\#I/\mathtt{now}]]\!](e_0)((\mathit{store}(s), \mathit{control}(s')), s')$

From (5) and (CD2), we know

(12) s' EQUALS s

From (12), (NEQ), (TRE), and (CD3), we know

(13) s' EQUALS $(\mathit{store}(s), \mathit{control}(s'))$

From (11), (13), (REE), (NEQ), and (ESF), we know (b.3.a). □

5.7.7 Throw Exception

$$\frac{E \simeq T \qquad \#I_s \text{ does not occur in } P}{\begin{array}{l}\text{POST}(\texttt{throw}\ I_k\ E, Q) = \\ \quad \texttt{now.throws}\ I_k\ \texttt{AND now.value} = T\ \texttt{AND} \\ \quad \texttt{EXSTATE}\ \#I_s\texttt{:}\ P[\#I_s/\texttt{now}]\end{array}}$$

Soundness Proof We have to show

$$\text{(a)} \quad \begin{array}{l}[\![\texttt{now.executes}]\!](e)(s,s)\ \Rightarrow \\ \quad [\![P]\!](e)(s,s) \wedge [\![\texttt{throw}\ I_k\ E]\!](s,s') \Rightarrow \\ \quad\quad [\![\texttt{now.throws}\ I_k\ \texttt{AND now.value} = T\ \texttt{AND} \\ \quad\quad\quad \texttt{EXSTATE}\ \#I\texttt{:}\ P[\#I/\texttt{now}]]\!](e)(s',s')\end{array}$$

We assume

(2) $[\![\texttt{now.executes}]\!](e)(s,s)$

(3) $[\![P]\!](e)(s,s)$

(4) $[\![\texttt{throw}\ I_k\ E]\!](s,s')$

By the definition of $[\![\_]\!]$, it suffices to show

(b.1) $\mathit{throws}(\mathit{control}(s'))$

(b.2) $\mathit{key}(\mathit{control}(s')) = I_k$

(b.3) $\mathit{value}(\mathit{control}(s')) = [\![T]\!](e)(s',s')$

(b.4) $\exists c \in \mathit{Control} : [\![P[\#I/\texttt{now}]]\!](e[I \mapsto c]_c)(s',s')$

From (4) and the definition of $[\![\_]\!]$, we know

(5) $s' = \mathit{throw}(s, I_k, [\![E]\!](s))$

From (5) and (CD1), we know (b.1) and (b.2).

From (5) and (CD1), we also know

(6) $\mathit{value}(\mathit{control}(s')) = [\![E]\!](s)$

From the first premise and the definition of $\simeq$, we know

(7) $[\![E]\!](s) = [\![T]\!](e)(s,s')$

From (6) and (7), we know (b.3).

We define

(8) $e_0 := e[I \mapsto \mathit{control}(s)]_c$

By (8), to show (b.4), it suffices to show

(b.4.a) $[\![P[\#I/\texttt{now}]]\!](e_0)(s',s')$

From the second premise, (3), (8), and (CNOF2), we know

(9) $[\![P[\#I/\texttt{now}]]\!](e_0)(s,s)$

From (1b), (9), (PVF2'), and (PVFNE), we know

(10) $[\![P[\#I/\texttt{now}]]\!](e_0)(s,s')$

From (10), (CNOF0), and (PVFNO), we know

(11) $[\![P[\#I/\texttt{now}]]\!](e_0)((\mathit{store}(s),\mathit{control}(s')),s')$

From (5) and (CD2), we know

(12) s' EQUALS s

From (12), (NEQ), (TRE), and (CD3), we know

(13) s' EQUALS $(\mathit{store}(s),\mathit{control}(s'))$

From (11), (13), (REE), (NEQ), and (ESF), we know (b.4.a). □

5.7.8 Catch Exception

$\#I$ does not occur in Q
$\$J$ does not occur in Q
$\$K$ does not occur in O
$J \neq K$
POST$(C_1,P) = Q$
POST$(C_2,$
  EXSTATE $\#I$:
    $\#I$`.throws` I_k AND $\#I$`.value` = I_v AND
    $Q[\#I/\texttt{now}][\$J/I_v]) = O$

POST$($`try` C_1 `catch` $(I_k\ I_v)$ $C_2,P) =$
  $(Q$ AND `!now.throws` $I_k)$ OR
  EXISTS $\$K$: $O[\$K/I_v][I_v/\$J]$

Soundness Proof We have to show

$$\text{(a)}\quad \begin{array}{l} [\![\texttt{now.executes}]\!](e)(s,s) \Rightarrow \\ \quad [\![P]\!](e)(s,s) \wedge [\![\texttt{try } C_1 \texttt{ catch (}I_k\ I_v\texttt{) } C_2]\!](s,s') \Rightarrow \\ \quad\quad [\![(Q \texttt{ AND !now.throws } I_k) \texttt{ OR} \\ \quad\quad\quad \texttt{EXISTS \$}K\texttt{: } O[\$K/I_v][I_v/\$J]]\!](e)(s',s') \end{array}$$

We assume

(2) $[\![\texttt{now.executes}]\!](e)(s,s)$

(3) $[\![P]\!](e)(s,s)$

(4) $[\![\texttt{try } C_1 \texttt{ catch (}I_k\ I_v\texttt{) } C_2]\!](s,s')$

By the definition of $[\![\_]\!]$, it suffices to show

$$\text{(b)}\quad \begin{array}{l} ([\![Q]\!](e)(s',s') \wedge \neg(\mathit{throws}(\mathit{control}(s')) \wedge \mathit{key}(\mathit{control}(s')) = I_k)) \vee \\ \exists v \in \mathit{Value} : [\![O[\$K/I_v][I_v/\$J]]\!](e[K \mapsto v])(s',s') \end{array}$$

From the premises, we know

(5) $\#I$ does not occur in Q

(6) $\$J$ does not occur in Q

(7) $\$K$ does not occur in O

(8) $J \neq K$

(9) $\textsc{post}(C_1, P) = Q$

$$\text{(10)}\quad \begin{array}{l} \textsc{post}(C_2, \\ \quad \texttt{EXSTATE \#}I\texttt{:} \\ \quad\quad \texttt{\#}I\texttt{.throws } I_k \texttt{ AND \#}I\texttt{.value} = I_v \texttt{ AND} \\ \quad\quad Q[\#I/\texttt{now}][\$J/I_v]) = O \end{array}$$

From (9), (10), and the induction hypothesis, we know

$$\text{(11)}\quad \begin{array}{l} \forall e \in \mathit{Environment}, s, s' \in \mathit{State} : \\ \quad [\![\texttt{now.executes}]\!](e)(s,s) \Rightarrow \\ \quad\quad [\![P]\!](e)(s,s) \wedge [\![C_1]\!](s,s') \Rightarrow [\![Q]\!](e)(s',s') \end{array}$$

$$\text{(12)}\quad \begin{array}{l} \forall e \in \mathit{Environment}, s, s' \in \mathit{State} : \\ \quad [\![\texttt{now.executes}]\!](e)(s,s) \Rightarrow \\ \quad\quad [\![\texttt{EXSTATE \#}I\texttt{:} \\ \quad\quad\quad \texttt{\#}I\texttt{.throws } I_k \texttt{ AND \#}I\texttt{.value} = I_v \texttt{ AND} \\ \quad\quad\quad Q[\#I/\texttt{now}][\$J/I_v]]\!](e)(s,s) \wedge [\![C_2]\!](s,s') \Rightarrow [\![O]\!](e)(s',s') \end{array}$$

From (4) and the definition of $[\![\_]\!]$, we know for some $s_0, s_1, s_2 \in \mathit{State}$

(13) $[\![C_1]\!](s, s_0)$

IF $\mathit{throws}(\mathit{control}(s_0)) \wedge \mathit{key}(\mathit{control}(s_0)) = I_k$ THEN
$s_1 = \mathit{write}(\mathit{execute}(s_0), I_v, \mathit{value}(\mathit{control}(s_0))) \wedge$
(14) $[\![C_2]\!](s_1, s_2) \wedge$
$s' = \mathit{write}(s_2, I_v, \mathit{read}(s_0, I_v))$
ELSE $s' = s_0$

From (2), (3), (11), and (13), we know

(15) $[\![Q]\!](e)(s_0, s_0)$

First, we handle the case

(16) $\neg(\mathit{throws}(\mathit{control}(s_0)) \wedge \mathit{key}(\mathit{control}(s_0)) = I_k)$

From (14) and (16), we know

(17) $s' = s_0$

From (15), (16), and (17), we know (b).

The remainder of the proof may thus proceed under the assumption

(18) $\mathit{throws}(\mathit{control}(s_0))$

(19) $\mathit{key}(\mathit{control}(s_0)) = I_k$

From (14), (18), and (19), we know

(20) $s_1 = \mathit{write}(\mathit{execute}(s_0), I_v, \mathit{value}(\mathit{control}(s_0)))$

(21) $[\![C_2]\!](s_1, s_2)$

(22) $s' = \mathit{write}(s_2, I_v, \mathit{read}(s_0, I_v))$

To show (b), it suffices to show

(c) $[\![O[\$K/I_v][I_v/\$J]]\!](e[K \mapsto \mathit{read}(s_2, I_v)])(s', s')$

From (20), (CD1), and (CWE), we know

(23) $\mathit{executes}(\mathit{control}(s_1))$

We define

(24) $e_0 := e[J \mapsto \mathit{read}(s_0, I_v)]$

Let us assume the following lemma (which will be shown below):

(d) $[\![\texttt{EXSTATE \#}I :$
$\quad \texttt{\#}I\texttt{.throws } I_k \texttt{ AND \#}I\texttt{.value} = I_v \texttt{ AND}$
$\quad Q[\texttt{\#}I/\texttt{now}][\$J/I_v]]\!](e_0)(s_1, s_1)$

From (12), (21), (23), (24), the definition of $[\![\_]\!]$, and (d), we know

(25) $[\![O]\!](e[J \mapsto \mathit{read}(s_0, I_v)])(s_2, s_2)$

From (22) and (CW), we know

(26) $\mathit{control}(s') = \mathit{control}(s_2)$

From (22) and (WS), we know

(27) $s' = s_2$ EXCEPT I_v

From (7), (25), (26), (27), and (PMVF1"), we know

(28) $[\![O[\$K/I_v]]\!](e[J \mapsto \mathit{read}(s_0, I_v), K \mapsto \mathit{read}(s_2, I_v)])(s', s_2)$

We define

(29) $O' := O[\$K/I_v]$

From (29) and (MPVF0'), we know

(30) I_v does not occur in O'

From (30) and (PMVF2'), we know

(31) $O'[I_v/\$J][\$J/I_v] = O'$

From (28), (29), and (31), we know

(32) $[\![O'[I_v/\$J][\$J/I_v]]\!](e[J \mapsto \mathit{read}(s_0, I_v), K \mapsto \mathit{read}(s_2, I_v)])(s', s_2)$

From (22) and (RW1), we know

(33) $\mathit{read}(s', I_v) = \mathit{read}(s_0, I_v)$

From (8), (29), (32), (33), (PMVF0'), (REE), and (PMVF1"), we know

(34) $[\![O[\$K/I_v][I_v/\$J]]\!](e[K \mapsto \mathit{read}(s_2, I_v)])(s', s_2)$

From (10), we know

(35) O has no primed program variables

From (26), (34), (35), and (PVF2'), we know (c).

It remains to show (d). We define

(36) $e_1 := e_0[I \mapsto \mathit{control}(s_0)]_c$

By (36), and the definition of $[\![\_]\!]$, it suffices to show

(e.1) $\mathit{throws}(\mathit{control}(s_0))$

(e.2) $\mathit{key}(\mathit{control}(s_0)) = I_k$

(e.3) $\mathit{value}(\mathit{control}(s_0)) = \mathit{read}(s_1, I_v)$

(e.4) $[\![Q[\#I/\mathtt{now}][\$J/I_v]]\!](e_1)(s_1, s_1)$

From (18), we know (e.1).

From (19), we know (e.2).

From (20), (WS), (CD2), (NEQ), and (RSE), we know (e.3).

From (6), (15), (24), and (MVF0'), we know

(37) $[\![Q]\!](e_0)(s_0, s_0)$

From (5), (36), (37), and (CNOF2), we know

(38) $[\![Q[\#I/\mathtt{now}]]\!](e_1)(s_0, s_0)$

From (20), (WS), (CD2), (NEQ), (AVE), and (TRE), we know

(39) $s_1 = s_0$ EXCEPT I_v

From (6), (24), (36), (38), (39), (CNOF0), and (PMVF1'), we know

(40) $[\![Q[\#I/\mathtt{now}][\$J/I_v]]\!](e_1)(s_1, s_0)$

From (9), we know

(41) Q has no primed variables and no occurrence of `next`

From (40), (41), (PVF2'), and (PVFNE), we know (e.4). □

5.7.9 While Loop (Without Invariant)

$$\frac{\begin{array}{l} C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \\ J_1,\ldots,J_n \text{ is a renaming of } I_1,\ldots,I_n \\ \$J_1,\ldots,\$J_n \text{ do not occur in } P \\ \#I_s \text{ does not occur in } P \end{array}}{\begin{array}{l} \text{POST}(\texttt{while (}E\texttt{) } C, P) = \\ \quad \texttt{EXISTS } \$J_1,\ldots,\$J_n\texttt{: EXSTATE } \#I_s\texttt{:} \\ \quad\quad P[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n] \end{array}}$$

Soundness Proof We have to show

$$\text{(a)} \quad \begin{array}{l} [\![\texttt{now.executes}]\!](e)(s,s) \Rightarrow \\ \quad [\![P]\!](e)(s,s) \wedge [\![\texttt{while (}E\texttt{) } C]\!](s,s') \Rightarrow \\ \quad\quad [\![\texttt{EXISTS } \$J_1,\ldots,\$J_n\texttt{: EXSTATE } \#I_s\texttt{:} \\ \quad\quad\quad P[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e)(s',s') \end{array}$$

We assume

(2) $[\![\texttt{now.executes}]\!](e)(s,s)$

(3) $[\![P]\!](e)(s,s)$

(4) $[\![\texttt{while (}E\texttt{) } C]\!](s,s')$

By the definition of $[\![\_]\!]$, it suffices to show

$$\text{(b)} \quad \begin{array}{l} \exists v_1,\ldots,v_n \in \mathit{Value}, c \in \mathit{State} : \\ \quad [\![P[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]]\!] \\ \quad\quad (e[I_s \mapsto c]_c[J_1 \mapsto v_1,\ldots,J_n \mapsto v_n])(s',s') \end{array}$$

From the premises, we know

(5) $C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}$

(6) $J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$

(7) $\$J_1,\ldots,\$J_n$ do not occur in P

(8) $\#I_s$ does not occur in P

From (2), (4), (5), and the soundness of the verification calculus, we can show (as demonstrated in Section 5.4.1)

(9) $s = s'$ EXCEPT $I_1,\ldots,I_n$

We define

$$(10)\quad \begin{array}{l} e_0 := e[I_s \mapsto \mathit{control}(s)\,]_c \\ e_1 := e_0[J_1 \mapsto \mathit{read}(s,I_1),\ldots,J_n \mapsto \mathit{read}(s,I_n)\,] \end{array}$$

By (10), to show (b), it suffices to show

$$(c)\quad [\![P[\#I_s/\mathtt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e_1)(s',s')$$

From (3), (8), (10), and (CNOF2), we know

$$(11)\quad [\![P[\#I_s/\mathtt{now}]]\!](e_0)(s,s)$$

From (6), (7), (9), (10), (11), (CNOF0), and (PMVF1'), we know

$$(12)\quad [\![P[\#I_s/\mathtt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e_1)(s',s)$$

From (1b), (9), (10), (12), (PVF4') and (PVFNE), we know (c). □

5.7.10 While Loop (Without Invariant, No Break)

$$\frac{\begin{array}{l} E \simeq H \\ C : [F]^{F_c,\mathtt{FALSE},F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \\ J_1,\ldots,J_n \text{ is a renaming of } I_1,\ldots,I_n \\ \$J_1,\ldots,\$J_n \text{ do not occur in } P \\ \#I_s \text{ does not occur in } P \end{array}}{\begin{array}{l} \mathrm{POST}(\mathtt{while\,(}E\mathtt{)}\ C,P) = \\ \quad \mathtt{(now.executes =>!}H\mathtt{)\ AND} \\ \quad \mathtt{EXISTS}\ \$J_1,\ldots,\$J_n\mathtt{:\ EXSTATE}\ \#I_s\mathtt{:} \\ \qquad P[\#I_s/\mathtt{now}][\$J_1/I_1,\ldots,\$J_n/I_n] \end{array}}$$

Soundness Proof We have to show

$$(a)\quad \begin{array}{l} [\![\mathtt{now.executes}]\!](e)(s,s)\ \Rightarrow \\ \quad [\![P]\!](e)(s,s) \wedge [\![\mathtt{while\,(}E\mathtt{)}\ C]\!](s,s') \Rightarrow \\ \qquad [\![\mathtt{(now.executes =>!}H\mathtt{)\ AND} \\ \qquad\quad \mathtt{EXISTS}\ \$J_1,\ldots,\$J_n\mathtt{:\ EXSTATE}\ \#I_s\mathtt{:} \\ \qquad\qquad P[\#I_s/\mathtt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e)(s',s') \end{array}$$

We assume

$$(2)\quad [\![\mathtt{now.executes}]\!](e)(s,s)$$

(3) $[\![P]\!](e)(s,s)$

(4) $[\![\texttt{while(}E\texttt{) }C]\!](s,s')$

By the definition of $[\![\,\_\,]\!]$, it suffices to show

(b.1) $executes(control(s')) \Rightarrow \neg[\![H]\!](e)(s',s')$

(b.2) $\exists v_1,\ldots,v_n \in Value, c \in State :$
$[\![P[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]]\!]$
$(e[I_s \mapsto c]_c[J_1 \mapsto v_1,\ldots,J_n \mapsto v_n])(s',s')$

As demonstrated in Section 5.7.9, we can show (b.2).

To show (b.1), we assume

(5) $executes(control(s'))$

and show

(b.1.a) $\neg[\![H]\!](e)(s',s')$

From the premises, we know

(6) $E \simeq H$

(7) $C : [F]^{F_c,\texttt{FALSE},F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}$

(8) $J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$

(9) $\$J_1,\ldots,\$J_n$ do not occur in P

(10) $\#I_s$ does not occur in P

From (6) and the definition of $\simeq$, to show (b.1.a), it suffices to show

(b.1.b) $[\![E]\!](s') \neq \text{TRUE}$

From (7), the soundness of the verification calculus and the definitions of $[\,\_\,]^{\_}_{\_}$ and $[\![\,\_\,]\!]$, we know

(11) $\forall s,s' \in State : [\![\texttt{now.executes}]\!](e)(s,s') \wedge [\![C]\!](s,s') \Rightarrow$
$\neg breaks(control(s'))$

From (4) and the definition of $[\![\,\_\,]\!]$, we know for some $k \in \mathbb{N}, t,u \in State^\infty$

(12) $finiteExecution(k,t,u,s,[\![E]\!],[\![C]\!])$

$$
(13)\quad
\begin{array}{l}
([\![E]\!](t(k)) \neq \text{TRUE} \vee \\
\quad \neg(\mathit{executes}(\mathit{control}(u(k))) \vee \\
\quad\quad \mathit{continues}(\mathit{control}(u(k)))))
\end{array}
$$

$$(14)\quad t(k) = s'$$

From (13), to show (b.1.b), it suffices to show

$$(\text{b.1.c})\quad \mathit{executes}(\mathit{control}(u(k))) \vee \mathit{continues}(\mathit{control}(u(k)))$$

From (12) and the definition of *finiteExecution*, we know

$$(15)\quad t(0) = s$$

$$(16)\quad u(0) = s$$

$$
(17)\quad
\begin{array}{l}
\forall i \in \mathbb{N}_k : \\
\quad \neg \mathit{breaks}(\mathit{control}(u(i))) \wedge \mathit{executes}(\mathit{control}(t(i))) \wedge \\
\quad [\![E]\!](t(i)) = \text{TRUE} \wedge [\![C]\!](t(i), u(i+1)) \wedge \\
\quad \text{IF } \mathit{continues}(\mathit{control}(u(i+1))) \vee \mathit{breaks}(\mathit{control}(u(i+1))) \\
\quad\quad \text{THEN } t(i+1) = \mathit{execute}(u(i+1)) \\
\quad\quad \text{ELSE } t(i+1) = u(i+1)
\end{array}
$$

From (2), (15), (16), and the definition of $[\![\,\_\,]\!]$, we know

$$(18)\quad \mathit{executes}(\mathit{control}(t(0)))$$

$$(19)\quad \mathit{executes}(\mathit{control}(u(0)))$$

If $k = 0$, by (19), we know (b.1.c).

We may thus assume

$$(20)\quad k > 0$$

From (17) and (20), we know

$$(21)\quad \mathit{executes}(\mathit{control}(t(k-1)))$$

$$(22)\quad [\![C]\!](t(k-1), u(k))$$

$$
(23)\quad
\begin{array}{l}
\text{IF } \mathit{continues}(\mathit{control}(u(k))) \vee \mathit{breaks}(\mathit{control}(u(k))) \\
\quad \text{THEN } t(k) = \mathit{execute}(u(k)) \\
\quad \text{ELSE } t(k) = u(k)
\end{array}
$$

From (11), (21), (22), and the definition of $[\![\,\_\,]\!]$, we know

$$(24)\quad \neg \mathit{breaks}(\mathit{control}(u(k)))$$

From (5), (14), (23) and (24), we know (b.1.c). □

5.7.11 While Loop (With Invariant)

$$\frac{\begin{array}{l} C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \\ J_1,\ldots,J_n \text{ is a renaming of } I_1,\ldots,I_n \\ \$J_1,\ldots,\$J_n \text{ do not occur in } P \\ \#I_s,\#I_t \text{ do not occur in } P \\ \mathit{Invariant}(G,H,F)_{I_1,\ldots,I_n} \end{array}}{\begin{array}{l} \texttt{POST}(\texttt{while (}E\texttt{) } C,P) = \\ \quad \texttt{EXISTS } \$J_1,\ldots,\$J_n\texttt{: EXSTATE } \#I_s,\#I_t\texttt{:} \\ \quad\quad P[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]\ \texttt{AND} \\ \quad\quad (G[\texttt{now/next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}] \\ \quad\quad\quad [\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]\ \texttt{=>} \\ \quad\quad\quad G[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n] \\ \quad\quad\quad\quad [\#I_t/\texttt{next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]\ \texttt{AND} \\ \quad\quad\quad \texttt{IF } \#I_t\texttt{.continues OR } \#I_t\texttt{.breaks} \\ \quad\quad\quad\quad \texttt{THEN next.executes} \\ \quad\quad\quad\quad \texttt{ELSE next == } \#I_t) \end{array}}$$

Soundness Proof We have to show

$$\begin{array}{l} [\![\texttt{now.executes}]\!](e)(s,s) \Rightarrow \\ \quad [\![P]\!](e)(s,s) \wedge [\![\texttt{while (}E\texttt{) } C]\!](s,s') \Rightarrow \\ \quad\quad [\![\texttt{EXISTS } \$J_1,\ldots,\$J_n\texttt{: EXSTATE } \#I_s,\#I_t\texttt{:} \\ \quad\quad\quad P[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]\ \texttt{AND} \\ \quad\quad\quad (G[\texttt{now/next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}] \\ \text{(a)} \quad\quad\quad\quad [\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]\ \texttt{=>} \\ \quad\quad\quad\quad G[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n] \\ \quad\quad\quad\quad\quad [\#I_t/\texttt{next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]\ \texttt{AND} \\ \quad\quad\quad\quad \texttt{IF } \#I_t\texttt{.continues OR } \#I_t\texttt{.breaks} \\ \quad\quad\quad\quad\quad \texttt{THEN next.executes} \\ \quad\quad\quad\quad\quad \texttt{ELSE next == } \#I_t)]\!](e)(s',s') \end{array}$$

We assume

(2) $[\![\texttt{now.executes}]\!](e)(s,s)$

(3) $[\![P]\!](e)(s,s)$

(4) $[\![\texttt{while (}E\texttt{) } C]\!](s,s')$

and show

(b)
$$
\begin{array}{l}
[\![\texttt{EXISTS } \$J_1,\ldots,\$J_n\texttt{: EXSTATE } \#I_s,\#I_t\texttt{:}\\
\quad P[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]\ \texttt{AND}\\
\quad (G[\texttt{now}/\texttt{next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]\\
\qquad [\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]\ \texttt{=>}\\
\qquad G[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]\\
\qquad\quad [\#I_t/\texttt{next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]\ \texttt{AND}\\
\qquad \texttt{IF } \#I_t\texttt{.continues OR } \#I_t\texttt{.breaks}\\
\qquad\quad \texttt{THEN next.executes}\\
\qquad\quad \texttt{ELSE next} == \#I_t)\,]\!](e)(s',s')
\end{array}
$$

From the definition of $[\![\,\lrcorner\,]\!]$, we have to show

(c)
$$
\begin{array}{l}
\exists v_1,\ldots,v_n \in \mathit{Value}, c_s, c_t \in \mathit{State}:\\
\quad [\![P[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]\ \texttt{AND}\\
\qquad (G[\texttt{now}/\texttt{next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]\\
\qquad\quad [\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]\ \texttt{=>}\\
\qquad\quad G[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]\\
\qquad\qquad [\#I_t/\texttt{next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]\ \texttt{AND}\\
\qquad\quad \texttt{IF } \#I_t\texttt{.continues OR } \#I_t\texttt{.breaks}\\
\qquad\qquad \texttt{THEN next.executes}\\
\qquad\qquad \texttt{ELSE next} == \#I_t)\,]\!]\\
\qquad (e[I_s \mapsto c_s, I_t \mapsto c_t]_c[J_1 \mapsto v_1,\ldots,J_n \mapsto v_n])(s',s')
\end{array}
$$

From the premises, we know

(5) $C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}$

(6) $J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$

(7) $\$J_1,\ldots,\$J_n$ do not occur in P

(8) $\#I_s,\#I_t$ do not occur in P

(9) $\mathit{Invariant}(G,H,F)_{I_1,\ldots,I_n}$

From (4) and the definition of $[\![\,\lrcorner\,]\!]$, we know for some $k \in \mathbb{N}, t, u \in \mathit{State}^\infty$

(10) $\mathit{finiteExecution}(k,t,u,s,[\![E]\!],[\![C]\!])$

(11)
$$
\begin{array}{l}
([\![E]\!](t(k)) \neq \textsc{true} \vee\\
\quad \neg(\mathit{executes}(\mathit{control}(u(k))) \vee\\
\qquad \mathit{continues}(\mathit{control}(u(k)))))
\end{array}
$$

(12) $t(k) = s'$

We define

$$
\begin{array}{ll}
 & e_0 := e[\,I_s \mapsto \mathit{control}(s)\,]_c \\
(13) & e_1 := e_0[\,I_t \mapsto \mathit{control}(u(k))\,]_c \\
 & e_2 := e_1[\,J_1 \mapsto \mathit{read}(s,I_1),\ldots,J_n \mapsto \mathit{read}(s,I_n)\,]
\end{array}
$$

By (13) and the definition of $[\![\,\_\,]\!]$, to show (c), it suffices to show

$$
\begin{array}{ll}
\text{(d.1)} & [\![P[\#I_s/\mathtt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e_2)(s',s') \\
\text{(d.2)} & [\![G[\mathtt{now}/\mathtt{next}][I_1/I_1\mathtt{'},\ldots,I_n/I_n\mathtt{'}]]\!](e_2)(s,s') \Rightarrow \\
 & \quad [\![G[\#I_s/\mathtt{now}][\$J_1/I_1,\ldots,\$J_n/I_n] \\
 & \qquad [\#I_t/\mathtt{next}][I_1/I_1\mathtt{'},\ldots,I_n/I_n\mathtt{'}]]\!](e_2)(s',s') \wedge \\
 & \quad \text{IF } \mathit{continues}(\mathit{control}(u(k))) \vee \mathit{breaks}(\mathit{control}(u(k))) \\
 & \qquad \text{THEN } \mathit{executes}(\mathit{control}(s')) \\
 & \qquad \text{ELSE } \mathit{control}(s') = \mathit{control}(u(k))
\end{array}
$$

From (3), (8), (13), and (CNOF2), we know

$$(14)\quad [\![P[\#I_s/\mathtt{now}]]\!](e_0)(s,s)$$

From (8), (13), (14), and (MVF1'), we know

$$(15)\quad [\![P[\#I_s/\mathtt{now}]]\!](e_1)(s,s)$$

From (2), (4), (5), and the soundness of the verification calculus, we can show (as demonstrated in Section 5.4.1)

$$(16)\quad s = s' \text{ EXCEPT } I_1,\ldots,I_n$$

From (6), (7), (13), (15), (16), (CNOF0), and (PMVF1'), we know

$$(17)\quad [\![P[\#I_s/\mathtt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e_2)(s',s)$$

From (1b), (13), (16), (17), (PVF4') and (PVFNE), we know (d.1).

To show (d.2), we assume

$$(18)\quad [\![G[\mathtt{now}/\mathtt{next}][I_1/I_1\mathtt{'},\ldots,I_n/I_n\mathtt{'}]]\!](e_2)(s,s')$$

and show

$$
\begin{array}{ll}
\text{(d.2.a.1)} & [\![G[\#I_s/\mathtt{now}][\$J_1/I_1,\ldots,\$J_n/I_n] \\
 & \quad [\#I_t/\mathtt{next}][I_1/I_1\mathtt{'},\ldots,I_n/I_n\mathtt{'}]]\!](e_2)(s',s') \\
 & \text{IF } \mathit{continues}(\mathit{control}(u(k))) \vee \mathit{breaks}(\mathit{control}(u(k))) \\
\text{(d.2.a.2)} & \quad \text{THEN } \mathit{executes}(\mathit{control}(s')) \\
 & \quad \text{ELSE } \mathit{control}(s') = \mathit{control}(u(k))
\end{array}
$$

From (10) and the definition of *finiteExecution*, we know

(19) $t(0) = s$

(20) $u(0) = s$

(21)
$$\begin{array}{l}\forall i \in \mathbb{N}_k : \\ \quad \neg \mathit{breaks}(\mathit{control}(u(i))) \wedge \mathit{executes}(\mathit{control}(t(i))) \wedge \\ \quad [\![E]\!](t(i)) = \text{TRUE} \wedge [\![C]\!](t(i), u(i+1)) \wedge \\ \quad \text{IF } \mathit{continues}(\mathit{control}(u(i+1))) \vee \mathit{breaks}(\mathit{control}(u(i+1))) \\ \qquad \text{THEN } t(i+1) = \mathit{execute}(u(i+1)) \\ \qquad \text{ELSE } t(i+1) = u(i+1)\end{array}$$

We show (d.2.a.2) by case distinction:

- Case $k = 0$: from this, (2), (19), and the definition of $[\![\_]\!]$, we know

  (22) $\mathit{executes}(\mathit{control}(u(k)))$

  From the case condition, (12), (19), and (20), we know

  (23) $s' = u(k)$

  From (22), (23), and Lemma "State Control Predicates", we know (d.2.a.2).

- Case $k > 0$: from this and (21), we know

  (24)
  $$\begin{array}{l}\text{IF } \mathit{continues}(\mathit{control}(u(k))) \vee \mathit{breaks}(\mathit{control}(u(k))) \\ \quad \text{THEN } t(k) = \mathit{execute}(u(k)) \\ \quad \text{ELSE } t(k) = u(k)\end{array}$$

  From (12), (24), and (CD1), we know (d.2.a.2).

It remains to show (d.2.a.1).

We define

(25) $s_0 := \mathit{writes}(s', I_1, \mathit{read}(s, I_1), \ldots, I_n, \mathit{read}(s, I_n))$

From (18), (25), and (PPVF2'), we know

(26) $[\![G[\texttt{now/next}]]\!](e_2)(s, s_0)$

From (26) and (PNNF2), we know

(27) $[\![G]\!](e_2)(s, (\mathit{store}(s_0), \mathit{control}(s)))$

From (25) and (RWE), we know

(28) $read(s_0, I_1) = read(s, I_1) \wedge \ldots \wedge read(s_0, I_n) = read(s, I_n)$

From (25) and (WSE), we know

(29) $s_0 = s'$ EXCEPT $I_1, \ldots, I_n$

From (16), (29), and (TRE), we know

(30) $s_0 = s$ EXCEPT $I_1, \ldots, I_n$

From (28), (30), (RVE), and (NEQ), we know

(31) s_0 EQUALS s

From (27), (31), (CD4) and (CD0), we know

(32) $[\![G]\!](e_2)(s, s)$

From (9) and the definition of *Invariant*, we know

(33) G has no free (mathematical or state) variables

(34) $\$I_1, \ldots, \$I_n, \#I_s, \#I_t$ do not occur in $G, H,$ and F

From (32), (33), and (MVF'), we know

(35) $[\![G]\!](e)(s, s)$

From (5), (9), and (35), we can derive (as demonstrated in the proof of the soundness of the invariant rule in Section 5.4.3)

(36) $[\![G]\!](e)(s, u(k))$

From (13), (34), (36) and (CNOF2), we know

(37) $[\![G[\#I_s/\texttt{now}]]\!](e_0)(s, u(k))$

From (6), (13), (16), (34), (37), (CNOF0), and (PMVF1'), we know

(38) $[\![G[\#I_s/\texttt{now}][\$J_1/I_1, \ldots, \$J_n/I_n]]\!](e_2)(s', u(k))$

From (13), (34), (38), and (CNEF2), we know

(39) $[\![G[\#I_s/\texttt{now}][\$J_1/I_1, \ldots, \$J_n/I_n][\#I_t/\texttt{next}]]\!](e_2)(s', u(k))$

We know from (12), (19), (20), and (21)

(40) $s' = u(k) \vee s' = execute(u(k))$

From (40) and (CD2), we know

(41) s' EQUALS $u(k)$

From (39), (42), (CD4), (PVFNE), and (CD0), we know

(43) $[\![G[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n][\#I_t/\texttt{next}]]\!](e_2)(s',s')$

From (43), (IDE), and (PPVF1'), we know (d.2.a.1). □

5.7.12 While Loop (With Invariant, No Break)

$E \simeq H$
$C : [F]^{F_c,\texttt{FALSE},F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}$
$J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$
$\$J_1,\ldots,\$J_n$ do not occur in P
$\#I_s,\#I_t$ do not occur in P
$Invariant(G,H,F)_{I_1,\ldots,I_n}$

POST(`while(`E`)` C,P) =
  `(now.executes =>!`H`) AND`
  `EXISTS` $\$J_1,\ldots,\$J_n$`: EXSTATE` $\#I_s,\#I_t$`:`
    $P[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]$ `AND`
    $(G[\texttt{now}/\texttt{next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]$
      $[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]$ =>
      $G[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]$
        $[\#I_t/\texttt{next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]$ `AND`
      `IF` $\#I_t$`.continues OR` $\#I_t$`.breaks`
        `THEN next.executes`
        `ELSE next ==` $\#I_t$)

Soundness Proof Analogous to the proof of the rule without invariant or breaks combined with the proof of the rule with invariants.

Assertion Calculus: Judgements

$\text{TRANS}(C,P) = C' \Leftrightarrow$
P has no primed program variables and no occurr. of `next` $\Rightarrow$
$\forall s,s' \in \mathit{State} : [\![\texttt{now.executes}]\!](s) \Rightarrow$
$([\![P]\!](s) \wedge [\![C]\!](s,s') \Leftrightarrow [\![C']\!](s,s'))$

Assertion Calculus: Definitions

$P^+ \equiv P$ `AND now.executes`

Assertion Calculus: Rules for Non-Loops

$\text{TRANS}(I\texttt{=}E,P) = \texttt{assert } P^+\texttt{; } I\texttt{=}E$

$\$J$ does not occur in P
$\text{TRANS}(C, \texttt{EXISTS } \$J : P[\$J/I]) = C'$
$\overline{\text{TRANS}(\texttt{var } I\texttt{;}C,P) = \texttt{assert } P^+\texttt{; var } I\texttt{;}C'}$

$E \simeq T$
$\$J$ does not occur in P and in T
$\text{TRANS}(C, \texttt{EXISTS } \$J : P[\$J/I] \texttt{ AND } I = T[\$J/I]) = C'$
$\overline{\text{TRANS}(\texttt{var } I\texttt{=}E\texttt{;}C,P) = \texttt{assert } P^+\texttt{; var } I\texttt{=}E\texttt{;}C'}$

$\text{POST}(C_1,P) = Q$
$\text{TRANS}(C_1,P) = C_1'$
$\text{TRANS}(C_2,Q) = C_2'$
$\overline{\text{TRANS}(C_1\texttt{;}C_2,P) = \texttt{assert } P^+\texttt{; } C_1'\texttt{;}C_2'}$

$E \simeq F$
$\text{TRANS}(C, P \texttt{ AND } F) = C'$
$\overline{\text{TRANS}(\texttt{if (}E\texttt{) } C,P) = \texttt{assert } P^+\texttt{; if (}E\texttt{) } C'}$

$E \simeq F$
$\text{TRANS}(C_1, P \texttt{ AND } F) = C_1'$
$\text{TRANS}(C_2, P \texttt{ AND !}F) = C_2'$
$\overline{\text{TRANS}(\texttt{if (}E\texttt{) } C_1 \texttt{ else } C_2,P) =}$
$\texttt{assert } P^+\texttt{; if (}E\texttt{) } C_1' \texttt{ else } C_2'$

Figure 5.30: The Assertion Calculus with Interruptions (Part 1/3)

Assertion Calculus: Rules for Non-Loops

$\textsc{trans}(\texttt{continue}, P) = \texttt{assert}\ P^+\texttt{;}\ \texttt{continue}$

$\textsc{trans}(\texttt{break}, P) = \texttt{assert}\ P^+\texttt{;}\ \texttt{break}$

$\textsc{trans}(\texttt{return}\ E, P) = \texttt{assert}\ P^+\texttt{;}\ \texttt{return}\ E$

$\textsc{trans}(\texttt{throw}\ I_k\ E, P) = \texttt{assert}\ P^+\texttt{;}\ \texttt{throw}\ I_k\ E$

$\$J$ does not occur in Q
$\#I_s$ does not occur in Q
$\textsc{post}(C_1, P) = Q$
$\textsc{trans}(C_1, P) = C_1'$
$\textsc{trans}(C_2,$
$\quad \texttt{EXISTS}\ \$J\texttt{:}\ \texttt{EXSTATE}\ \#I_s\texttt{:}$
$\quad\quad Q[\#I_s/\texttt{now}][\$J/I_v]\ \texttt{AND}$
$\quad\quad \#I_s\texttt{.throws}\ I_k\ \texttt{AND}\ \#I_s\texttt{.value} = I_v) = C_2'$

$\textsc{trans}(\texttt{try}\ C_1\ \texttt{catch}\ (I_k\ I_v)\ C_2, P) =$
$\quad \texttt{assert}\ P^+\texttt{;}\ \texttt{try}\ C_1'\ \texttt{catch}\ (I_k\ I_v)\ C_2'$

Figure 5.31: The Assertion Calculus with Interruptions (Part 2/3)

Assertion Calculus: Rules for Loops

$$\frac{\begin{array}{l} E \simeq H \\ C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \\ J_1,\ldots,J_n \text{ is a renaming of } I_1,\ldots,I_n \\ \$J_1,\ldots,\$J_n \text{ do not occur in } P \\ \#I_s \text{ does not occur in } P \\ \textsc{trans}(C, \\ \quad H \texttt{ AND} \\ \quad \texttt{EXISTS } \$J_1,\ldots,\$J_n\texttt{: EXSTATE } \#I_s\texttt{:} \\ \quad\quad P[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]) = C' \end{array}}{\textsc{trans}(\texttt{while(}E\texttt{) } C, P) = \texttt{assert } P^+\texttt{; while(}E\texttt{) } C'}$$

$$\frac{\begin{array}{l} E \simeq H \\ C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \\ J_1,\ldots,J_n \text{ is a renaming of } I_1,\ldots,I_n \\ \$J_1,\ldots,\$J_n \text{ do not occur in } P \\ \#I_s, \#I_t \text{ do not occur in } P \\ \mathit{Invariant}(G,H,F)_{I_1,\ldots,I_n} \\ \textsc{trans}(C, \\ \quad H \texttt{ AND} \\ \quad \texttt{EXISTS } \$J_1,\ldots,\$J_n\texttt{: EXSTATE } \#I_s, \#I_t\texttt{:} \\ \quad\quad P[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n] \texttt{ AND} \\ \quad\quad (G[\texttt{now/next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}] \\ \quad\quad\quad [\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n] \texttt{ =>} \\ \quad\quad\quad G[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n] \\ \quad\quad\quad\quad [\#I_t/\texttt{next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}] \texttt{ AND} \\ \quad\quad\quad (\texttt{!(}\#I_t\texttt{.continues OR }\#I_t\texttt{.breaks) =>} \\ \quad\quad\quad\quad \texttt{now ==} \#I_t\texttt{)))} = C' \end{array}}{\textsc{trans}(\texttt{while(}E\texttt{) } C, P) = \texttt{assert } P^+\texttt{; while(}E\texttt{) } C'}$$

Figure 5.32: The Assertion Calculus with Interruptions (Part 3/3)

5.8 Computing Assertions

This section generalizes the assertion calculus presented in Section 4.3 to take into account the possibility of control flow interruptions. Again the judgement $\text{TRANS}(C,P) = P'$ constructs a version C' of command C (which is assumed to execute in a state satisfying precondition P) that is annotated with `assert` commands such that the meaning of C is preserved. The interpretation of judgements presented in Figure 5.30 is generalized to take into account that the prestates of commands are always executing. Correspondingly, all assertion conditions have now form (P `AND now.executes`) which is abbreviated to P^+.

The rules of the calculus are presented in Figures 5.30, 5.31, and 5.32. The following theorem states the corresponding soundness claim.

Theorem (Soundness of the Assertion Calculus) Assume the condition denoted by *DifferentVariables*. If $\text{TRANS}(C,P) = C'$ can be derived from the rules of the assertion calculus of the command language, then it is true that

$$\begin{array}{l} P \text{ has no primed program variables and no occurrence of } \texttt{next} \Rightarrow \\ \quad \forall s, s' \in \mathit{State} : [\![\texttt{now.executes}]\!](s) \Rightarrow \\ \quad\quad ([\![P]\!](s) \wedge [\![C]\!](s,s') \Leftrightarrow [\![C']\!](s,s')) \end{array}$$

Assume

(1a) *DifferentVariables*

Take C, P, and C' such that $\text{TRANS}(C,P) = C'$ can be derived and assume

(1b) P has no primed program variables and no occurrence of `next`

Now take arbitrary $s, s' \in \mathit{State}$. We prove

$$\begin{array}{l} [\![\texttt{now.executes}]\!](s) \Rightarrow \\ \quad ([\![P]\!](s) \wedge [\![C]\!](s,s') \Leftrightarrow [\![C']\!](s,s')) \end{array}$$

Again, we only show the "left to right" direction by induction on the derivation of $\text{TRANS}(C,P) = C'$. The following subsections cover all cases for the last step of such a derivation.

From (1b) and the rules, we can immediately deduce that in every derivation $\text{TRANS}(C',P') = C'$ that matches the premise of a rule which has conclusion $\text{TRANS}(C,P) = C$, the formula P' has no primed variables; we thus assume in the

proofs that the induction hypothesis immediately implies the core claim $\forall s, s' \in \mathit{State} : [\![\texttt{now.executes}]\!](s) \Rightarrow ([\![P']\!](s) \wedge [\![C']\!](s,s') \Rightarrow [\![C']\!](s',s'))$. □

Actually, we will only show detailed proofs for some of the cases including all those where interruptions play a role; the other proofs are analogous to those of the calculus for programs without interruptions.

5.8.1 Assignment

$$\textsc{trans}(I{=}E, P) = \texttt{assert}\ P^{+}\texttt{;}\ I{=}E$$

Soundness Proof We have to prove

(a) $[\![\texttt{now.executes}]\!](s) \Rightarrow$
$[\![P]\!](s) \wedge [\![I{=}E]\!](s,s') \Rightarrow [\![\texttt{assert}\ P^{+}\texttt{;}\ I{=}E]\!](s,s')$

We assume

(2) $[\![\texttt{now.executes}]\!](s)$

(3) $[\![P]\!](s)$

(4) $[\![I{=}E]\!](s,s')$

By the definition of $[\![\_]\!]$, it suffices to show

(b.1) $[\![P]\!](s)$

(b.2) $\mathit{executes}(\mathit{control}(s))$

(b.3) $[\![I{=}E]\!](s,s')$

From (3), we know (b.1). From (2) and the definition of $[\![\_]\!]$, we know (b.2). From (4), we know (b.3). □

5.8.2 Command Sequence

$$\frac{\textsc{post}(C_1, P) = Q \quad \textsc{trans}(C_1, P) = C_1' \quad \textsc{trans}(C_2, Q) = C_2'}{\textsc{trans}(C_1\texttt{;}\,C_2, P) = \texttt{assert}\ P^{+}\texttt{;}\ C_1'\texttt{;}\,C_2'}$$

Soundness Proof We have to prove

$$\text{(a)}\quad \begin{array}{l}[\![\texttt{now.executes}]\!](s) \Rightarrow \\ \quad [\![P]\!](s) \wedge [\![C_1\,;C_2]\!](s,s') \Rightarrow [\![\texttt{assert}\ P^+\texttt{;}\ C_1'\,;C_2']\!](s,s')\end{array}$$

We assume

$$\text{(2)}\quad [\![\texttt{now.executes}]\!](s)$$
$$\text{(3)}\quad [\![P]\!](s)$$
$$\text{(4)}\quad [\![C_1\,;C_2]\!](s,s')$$

By the definition of $[\![\,\_\,]\!]$, it suffices to show

$$\text{(b.1)}\quad [\![P]\!](s)$$
$$\text{(b.2)}\quad \mathit{executes}(\mathit{control}(s)$$
$$\text{(b.3)}\quad \begin{array}{l}\exists s_0 \in \mathit{State}: \\ \quad [\![C_1']\!](s,s_0) \wedge \\ \quad \text{IF } \mathit{executes}(\mathit{control}(s_0)) \text{ THEN } [\![C_2']\!](s_0,s') \text{ ELSE } s' = s_0\end{array}$$

From the premises and the induction hypothesis, we know

$$\text{(5)}\quad \text{POST}(C_1,P) = Q$$
$$\text{(6)}\quad \begin{array}{l}\forall s,s' \in \mathit{State}: [\![\texttt{now.executes}]\!](s) \Rightarrow \\ \quad [\![P]\!](s) \wedge [\![C_1]\!](s,s') \Rightarrow [\![C_1']\!](s,s')\end{array}$$
$$\text{(7)}\quad \begin{array}{l}\forall s,s' \in \mathit{State}: [\![\texttt{now.executes}]\!](s) \Rightarrow \\ \quad [\![Q]\!](s) \wedge [\![C_2]\!](s,s') \Rightarrow [\![C_2']\!](s,s')\end{array}$$

From (3), we know (b.1). From (2) and the definition of $[\![\,\_\,]\!]$, we know (b.2).

From (4) and the definition of $[\![\,\_\,]\!]$, we know for some $s_0 \in \mathit{State}$

$$\text{(8)}\quad [\![C_1]\!](s,s_0)$$
$$\text{(9)}\quad \text{IF } \mathit{executes}(\mathit{control}(s_0)) \text{ THEN } [\![C_2]\!](s_0,s') \text{ ELSE } s' = s_0$$

To show (b.3), it suffices to show

$$\text{(b.3.a.1)}\quad [\![C_1']\!](s,s_0)$$
$$\text{(b.3.a.2)}\quad \text{IF } \mathit{executes}(\mathit{control}(s_0)) \text{ THEN } [\![C_2']\!](s_0,s') \text{ ELSE } s' = s_0$$

From (2), (3), (6), and (8), we know (b.3.a.1).

If $\neg \mathit{executes}(\mathit{control}(s_0))$, by (9), we know (b.3.a.2).

We thus may proceed with the proof of (b.3.a.2) under the assumption

(10) $executes(control(s_0))$

From (9) and (10), we know

(11) $[\![C_2]\!](s_0, s')$

From (10) and the definition of $[\![\_]\!]$, we know

(12) $[\![\texttt{now.executes}]\!](s_0)$

Let us assume

(e) $[\![Q]\!](s_0)$

From (e), (7), (10), (11), and (12), we know (b.3.a.2).

It remains to show (e), i.e. for arbitrary $e \in Environment$

(f) $[\![Q]\!](e)(s_0, s_0)$

From (1b), (5), and the soundness of the postcondition calculus, we know

(13) $\forall e \in Environment, s, s' \in State : [\![\texttt{now.executes}]\!](s) \Rightarrow$
$[\![P]\!](e)(s,s) \wedge [\![C_1]\!](s,s') \Rightarrow [\![Q]\!](e)(s',s')$

From (2), (3), (8), and (13), we know (f). □

5.8.3 Catch Exception

$\$J$ does not occur in Q
$\#I_s$ does not occur in Q
$\text{POST}(C_1, P) = Q$
$\text{TRANS}(C_1, P) = C_1'$
$\text{TRANS}(C_2,$
  EXISTS $\$J$: EXSTATE $\#I_s$:
    $Q[\#I_s/\texttt{now}][\$J/I_v]$ AND
    $\#I_s$.throws I_k AND $\#I_s$.value $= I_v) = C_2'$
$\text{TRANS}(\texttt{try}\ C_1\ \texttt{catch}\ (I_k\ I_v)\ C_2, P) =$
  assert P^+; try C_1' catch $(I_k\ I_v)\ C_2'$

Soundness Proof We have to prove

$$
\begin{array}{ll}
 & [\![\texttt{now.executes}]\!](s) \Rightarrow \\
\text{(a)} & \quad [\![P]\!](s) \wedge [\![\texttt{try}\ C_1\ \texttt{catch}\ (I_k\ I_v)\ C_2]\!](s,s') \Rightarrow \\
 & \quad [\![\texttt{assert}\ P^+\texttt{;}\ \texttt{try}\ C_1'\ \texttt{catch}\ (I_k\ I_v)\ C_2']\!](s,s')
\end{array}
$$

We assume

(2) $[\![\texttt{now.executes}]\!](s)$

(3) $[\![P]\!](s)$

(4) $[\![\texttt{try}\ C_1\ \texttt{catch}\ (I_k\ I_v)\ C_2]\!](s,s')$

By the definition of $[\![\_]\!]$, it suffices to show

(b.1) $[\![P]\!](s)$

(b.2) $\mathit{executes}(\mathit{control}(s)$

(b.3)
$$
\begin{array}{l}
\exists s_0, s_1, s_2 \in \mathit{State}: \\
\quad [\![C_1']\!](s,s_0) \wedge \\
\quad \text{IF } \mathit{throws}(\mathit{control}(s_0)) \wedge \mathit{key}(\mathit{control}(s_0)) = I_k \text{ THEN} \\
\qquad s_1 = \mathit{write}(\mathit{execute}(s_0), I_v, \mathit{value}(\mathit{control}(s_0))) \wedge \\
\qquad [\![C_2']\!](s_1,s_2) \wedge \\
\qquad s' = \mathit{write}(s_2, I_v, \mathit{read}(s_0, I_v)) \\
\quad \text{ELSE } s' = s_0
\end{array}
$$

From the premises and the induction hypothesis, we know

(5) $\$J$ does not occur Q

(6) $\#I_s$ does not occur in Q

(7) $\text{POST}(C_1, P) = Q$

(8)
$$
\begin{array}{l}
\forall s, s' \in \mathit{State}: [\![\texttt{now.executes}]\!](s) \Rightarrow \\
\quad [\![P]\!](s) \wedge [\![C_1]\!](s,s') \Rightarrow [\![C_1']\!](s,s')
\end{array}
$$

(9)
$$
\begin{array}{l}
\forall s, s' \in \mathit{State}: [\![\texttt{now.executes}]\!](s) \Rightarrow \\
\quad [\![\texttt{EXISTS}\ \$J\texttt{:}\ \texttt{EXSTATE}\ \#I_s\texttt{:} \\
\qquad Q[\#I_s/\texttt{now}][\$J/I_v]\ \texttt{AND} \\
\qquad \#I_s\texttt{.throws}\ I_k\ \texttt{AND}\ \#I_s\texttt{.value} = I_v]\!](s) \wedge [\![C_2]\!](s,s') \Rightarrow \\
\quad [\![C_2']\!](s,s')
\end{array}
$$

From (3), we know (b.1). From (2) and the definition of $[\![\_]\!]$, we know (b.2). From (4) and the definition of $[\![\_]\!]$, we know for some $s_0, s_1, s_2 \in \mathit{State}$

(10) $[\![C_1]\!](s, s_0)$

IF $throws(control(s_0)) \wedge key(control(s_0)) = I_k$ THEN
$s_1 = write(execute(s_0), I_v, value(control(s_0))) \wedge$
(11) $[\![C_2]\!](s_1, s_2) \wedge$
$s' = write(s_2, I_v, read(s_0, I_v))$
ELSE $s' = s_0$

To show (b.3), it suffices to show

(b.3.a.1) $[\![C_1']\!](s, s_0)$

IF $throws(control(s_0)) \wedge key(control(s_0)) = I_k$ THEN
$s_1 = write(execute(s_0), I_v, value(control(s_0))) \wedge$
(b.3.a.2) $[\![C_2']\!](s_1, s_2) \wedge$
$s' = write(s_2, I_v, read(s_0, I_v))$
ELSE $s' = s_0$

From (2), (3), (8), and (10), we know (b.3.a.1).

If $\neg(throws(control(s_0)) \wedge key(control(s_0)) = I_k)$, by (11), we know (b.3.a.2).

We thus may proceed with the proof of (b.3.a.2) under the assumptions

(12) $throws(control(s_0))$

(13) $key(control(s_0)) = I_k$

From (11), (12), and (13), we know

(14) $s_1 = write(execute(s_0), I_v, value(control(s_0)))$

(15) $[\![C_2]\!](s_1, s_2)$

(16) $s' = write(s_2, I_v, read(s_0, I_v))$

By (12), (13), (14), and (16), to show (b.3.a.2), it suffices to show

(b.3.a.2.a) $[\![C_2']\!](s_1, s_2)$

From (14), (CD1), and (CW), we know

(17) $executes(control(s_1))$

From (17) and the definition of $[\![\_]\!]$, we know

(18) $[\![\texttt{now.executes}]\!](s_1)$

From (9), (15), and (18), to show (b.3.a.2.a), it suffices to show

(b.3.a.2.b) $[\![$ `EXISTS $`J` :` `EXSTATE #`I_s` :`
$Q[\#I_s/\texttt{now}][\$J/I_v]$ `AND`
`#`I_s`.throws` I_k `AND #`I_s`.value` $= I_v$ $]\!](s_1)$

i.e. for arbitrary $e \in \mathit{Environment}$

(b.3.a.2.c) $\exists v \in \mathit{Value}, c \in \mathit{Control}:$
$[\![Q[\#I_s/\texttt{now}][\$J/I_v]]\!](e[I_s \mapsto c]_c[J \mapsto v])(s_1,s_1) \wedge$
$\mathit{throws}(c) \wedge \mathit{key}(c) = I_k \wedge \mathit{value}(c) = \mathit{read}(s_1, I_v)$

We define

(19) $e_0 := e[I_s \mapsto \mathit{control}(s_0)]_c$
$e_1 := e_0[J \mapsto \mathit{read}(s_0, I_v)]$

By (19), to show (b.3.a.2.c), it suffices to show

(b.3.a.2.d.1) $[\![Q[\#I_s/\texttt{now}][\$J/I_v]]\!](e_1)(s_1,s_1)$

(b.3.a.2.d.2) $\mathit{throws}(\mathit{control}(s_0))$

(b.3.a.2.d.3) $\mathit{key}(\mathit{control}(s_0)) = I_k$

(b.3.a.2.d.4) $\mathit{value}(\mathit{control}(s_0)) = \mathit{read}(s_1, I_v)$

From (12), we know (b.3.a.2.d.2).

From (13), we know (b.3.a.2.d.3).

From (14) and (RW1), we know (b.3.a.2.d.4).

From (1a), (1b), (7), and the soundness of the postcondition calculus, we know

(20) Q has no primed program variables and no occurrence of `next`

(21) $\forall e \in \mathit{Environment}, s, s' \in \mathit{State} : [\![\texttt{now.executes}]\!](s) \Rightarrow$
$[\![P]\!](e)(s,s) \wedge [\![C_1]\!](s,s') \Rightarrow [\![Q]\!](e)(s',s')$

From (2), (3), (10), and (21), we know

(22) $[\![Q]\!](e)(s_0,s_0)$

From (6), (19), (22), and (CNOF2), we know

(23) $[\![Q[\#I_s/\texttt{now}]]\!](e_0)(s_0,s_0)$

From (14), (WS), (CD2), (NEQ), (AVE), and (TRE), we know

(24) $s_1 = s_0$ EXCEPT I_v

From (5), (19), (23), (24), (CNOF0), and (PMVF1'), we know

(25) $[\![Q[\#I_s/\texttt{now}][\$J/I_v]]\!](e_1)(s_1,s_0)$

From (20), (25), (PVF2'), and (PVFNE), we know (b.3.a.2.d.1). □

5.8.4 While Loop (Without Invariant)

$$\frac{\begin{array}{l} E \simeq H \\ C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \\ J_1,\ldots,J_n \text{ is a renaming of } I_1,\ldots,I_n \\ \$J_1,\ldots,\$J_n \text{ do not occur in } P \\ \#I_s \text{ does not occur in } P \\ \textsc{trans}(C, \\ \quad H \texttt{ AND} \\ \quad \texttt{EXISTS } \$J_1,\ldots,\$J_n\texttt{: EXSTATE } \#I_s\texttt{:} \\ \qquad P[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]) = C' \end{array}}{\textsc{trans}(\texttt{while (}E\texttt{) } C, P) = \texttt{assert } P^+\texttt{; while (}E\texttt{) } C'}$$

Soundness Proof We have to prove

$$\begin{array}{ll} & [\![\texttt{now.executes}]\!](s) \Rightarrow \\ \text{(a)} & \quad [\![P]\!](s) \wedge [\![\texttt{while (}E\texttt{) } C]\!](s,s') \Rightarrow \\ & \qquad [\![\texttt{assert } P^+\texttt{; while (}E\texttt{) } C']\!](s,s') \end{array}$$

We assume

$$\begin{array}{ll} \text{(2)} & [\![\texttt{now.executes}]\!](s) \\ \text{(3)} & [\![P]\!](s) \\ \text{(4)} & [\![\texttt{while (}E\texttt{) } C]\!](s,s') \end{array}$$

By the definition of $[\![\,\_\,]\!]$, it suffices to show

$$\begin{array}{ll} \text{(b.1)} & [\![P]\!](s) \\ \text{(b.2)} & \mathit{executes}(\mathit{control}(s)) \\ \text{(b.3)} & \begin{array}{l} \exists k \in \mathbb{N}, t, u \in \mathit{State}^\infty : \\ \quad \mathit{finiteExecution}(k,t,u,s,[\![E]\!],[\![C']\!]) \wedge \\ \quad ([\![E]\!](t(k)) \neq \textsc{true} \vee \\ \qquad \neg(\mathit{executes}(\mathit{control}(u(k))) \vee \\ \qquad\quad \mathit{continues}(\mathit{control}(u(k))))) \wedge \\ \quad t(k) = s' \end{array} \end{array}$$

From the premises and the induction hypothesis, we know

$$\begin{array}{ll} \text{(5)} & E \simeq H \\ \text{(6)} & C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \end{array}$$

(7) $J_1, \ldots, J_n$ is a renaming of $I_1, \ldots, I_n$

(8) $\$J_1, \ldots, \$J_n$ do not occur in P

(9) $\#I_s$ does not occur in P

(10)
$$\begin{array}{l}\forall s, s' \in \mathit{State}: \\ \quad [\![\texttt{now.executes}]\!](s) \wedge \\ \quad [\![H \texttt{ AND} \\ \qquad \texttt{EXISTS } \$J_1, \ldots, \$J_n \texttt{: EXSTATE } \#I_s \texttt{:} \\ \qquad\quad P[\#I_s/\texttt{now}][\$J_1/I_1, \ldots, \$J_n/I_n]]\!](s) \wedge \\ \quad [\![C]\!](s, s') \Rightarrow \\ \qquad [\![C']\!](s, s')\end{array}$$

From (3), we know (b.1).

From (4) and the definition of $[\![\_]\!]$, we know (b.2).

From (4) and the definition of $[\![\_]\!]$, we know for some $k \in \mathbb{N}, t, u \in \mathit{State}^\infty$

(11) $\mathit{finiteExecution}(k, t, u, s, [\![E]\!], [\![C]\!])$

(12) $[\![E]\!](t(k)) \neq \text{TRUE} \vee$
$\neg(\mathit{executes}(\mathit{control}(u(k))) \vee \mathit{continues}(\mathit{control}(u(k))))$

(13) $t(k) = s'$

To show (b.3), from (12) and (13), it suffices to show

(b.3.a) $\mathit{finiteExecution}(k, t, u, s, [\![E]\!], [\![C']\!])$

i.e., by the definition of *finiteExecution*,

(b.3.b.1) $t(0) = s$

(b.3.b.2) $u(0) = s$

(b.3.b.3)
$$\begin{array}{l}\forall i \in \mathbb{N}_k: \\ \quad \neg \mathit{breaks}(\mathit{control}(u(i))) \wedge \mathit{executes}(\mathit{control}(t(i))) \wedge \\ \quad [\![E]\!](t(i)) = \text{TRUE} \wedge [\![C']\!](t(i), u(i+1)) \wedge \\ \quad \text{IF } \mathit{continues}(\mathit{control}(u(i+1))) \vee \mathit{breaks}(\mathit{control}(u(i+1))) \\ \qquad \text{THEN } t(i+1) = \mathit{execute}(u(i+1)) \\ \qquad \text{ELSE } t(i+1) = u(i+1)\end{array}$$

From (11) and the definition of *finiteExecution*, we know

(14) $t(0) = s$

(15) $u(0) = s$

$$
\begin{array}{ll}
 & \forall i \in \mathbb{N}_k : \\
 & \quad \neg \mathit{breaks}(\mathit{control}(u(i))) \wedge \mathit{executes}(\mathit{control}(t(i))) \wedge \\
(16) & \quad [\![E]\!](t(i)) = \text{TRUE} \wedge [\![C]\!](t(i), u(i+1)) \wedge \\
 & \quad \text{IF } \mathit{continues}(\mathit{control}(u(i+1))) \vee \mathit{breaks}(\mathit{control}(u(i+1))) \\
 & \qquad \text{THEN } t(i+1) = \mathit{execute}(u(i+1)) \\
 & \qquad \text{ELSE } t(i+1) = u(i+1)
\end{array}
$$

From (14), we know (b.3.b.1).

From (15), we know (b.3.b.2).

To show (b.3.b.3), we show for arbitrary $i \in \mathbb{N}_k$

(b.3.b.3.a.1) $\neg \mathit{breaks}(\mathit{control}(u(i))$

(b.3.b.3.a.2) $\mathit{executes}(\mathit{control}(t(i)))$

(b.3.b.3.a.3) $[\![E]\!](t(i)) = \text{TRUE}$

(b.3.b.3.a.4) $[\![C']\!](t(i), u(i+1))$

$$
\begin{array}{ll}
 & \text{IF } \mathit{continues}(\mathit{control}(u(i+1))) \vee \mathit{breaks}(\mathit{control}(u(i+1))) \\
\text{(b.3.b.3.a.5)} & \quad \text{THEN } t(i+1) = \mathit{execute}(u(i+1)) \\
 & \quad \text{ELSE } t(i+1) = u(i+1)
\end{array}
$$

From (16), we know (b.3.b.a.1), (b.3.b.a.2), (b.3.b.a.3), (b.3.b.a.5) and

(17) $[\![C]\!](t(i), u(i+1))$

To show (b.3.b.3.a.4), from (10), (17), (b.3.b.3.a.2) and the definition of $[\![\_]\!]$, it suffices to show

$$
\begin{array}{ll}
 & [\![H \ \texttt{AND} \\
\text{(b.3.b.3.a.4.a)} & \quad \texttt{EXISTS } \$J_1, \ldots, \$J_n \texttt{: EXSTATE } \#I_s \texttt{:} \\
 & \qquad P[\#I_s/\texttt{now}][\$J_1/I_1, \ldots, \$J_n/I_n]]\!](t(i))
\end{array}
$$

i.e. by the definition of $[\![\_]\!]$, for arbitrary $e \in \mathit{Environment}$,

(b.3.b.3.a.4.b.1) $[\![H]\!](e)(t(i), t(i))$

$$
\begin{array}{ll}
 & \exists v_1, \ldots, v_n \in \mathit{Value}, c \in \mathit{Control} : \\
\text{(b.3.b.3.a.4.b.2)} & \quad [\![P[\#I_s/\texttt{now}][\$J_1/I_1, \ldots, \$J_n/I_n]]\!] \\
 & \qquad (e[I_s \mapsto c]_c[J_1 \mapsto v_1, \ldots, J_n \mapsto v_n])(t(i), t(i))
\end{array}
$$

From (5), (16), and the definition of $\simeq$, we know (b.3.b.3.a.4.b.1).

From (6), the soundness of the verification calculus with interruptions, and the definition of $\simeq$, we know

(18) $\forall s, s' \in \mathit{Store}:$
$\quad \mathit{executes}(\mathit{control}(s)) \wedge [\![C]\!](s,s') \Rightarrow s = s' \text{ EXCEPT } I_1, \ldots, I_n$

From (16) and (18), we know

(19) $\forall i \in \mathbb{N}_k : t(i) = u(i+1) \text{ EXCEPT } I_1, \ldots, I_n$

From (16), (CD2), and (REE), we know

(20) $\forall i \in \mathbb{N}_k : t(i+1) \text{ EQUALS } u(i+1)$

From (14), (19), (20), (NEQ), (AVE), and (TRE), we know

(21) $s = t(i) \text{ EXCEPT } I_1, \ldots, I_n$

We define

(22) $e_0 := e[I_s \mapsto \mathit{control}(s)]_c$
$e_1 := e_0[J_1 \mapsto \mathit{read}(s, I_1), \ldots, J_n \mapsto \mathit{read}(s, I_n)]$

From (22), to show (b.3.b.3.a.4.b.2), it suffices to show

(b.3.b.3.a.4.b.2.a) $[\![P[\#I_s/\mathtt{now}][\$J_1/I_1, \ldots, \$J_n/I_n]]\!](e_1)(t(i), t(i))$

From (3) and the definition of $[\![\_]\!]$, we know

(23) $[\![P]\!](e)(s,s)$

From (9), (22), (23), and (CNOF2), we know

(24) $[\![P[\#I_s/\mathtt{now}]]\!](e_0)(s,s)$

From (7), (8), (21), (22), (24), (CNOF0), and (PMVF1'), we know

(25) $[\![P[\#I_s/\mathtt{now}][\$J_1/I_1, \ldots, \$J_n/I_n]]\!](e_1)(t(i), s)$

From (1b), (25), (PVF2'), and (PVFNE), we know (b.3.b.3.a.4.b.2.a). □

5.8.5 While Loop (With Invariant)

$$\frac{\begin{array}{l}
E \simeq H \\
C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \\
J_1,\ldots,J_n \text{ is a renaming of } I_1,\ldots,I_n \\
\$J_1,\ldots,\$J_n \text{ do not occur in } P \\
\#I_s, \#I_t \text{ do not occur in } P \\
\mathit{Invariant}(G,H,F)_{I_1,\ldots,I_n} \\
\textsc{trans}(C, \\
\quad H \texttt{ AND} \\
\quad \texttt{EXISTS } \$J_1,\ldots,\$J_n\texttt{: EXSTATE } \#I_s,\#I_t\texttt{:} \\
\quad\quad P[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n] \texttt{ AND} \\
\quad\quad (G[\texttt{now}/\texttt{next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}] \\
\quad\quad\quad [\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n] \texttt{ =>} \\
\quad\quad\quad G[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n] \\
\quad\quad\quad\quad [\#I_t/\texttt{next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}] \texttt{ AND} \\
\quad\quad\quad \texttt{(!(\#}I_t\texttt{.continues OR \#}I_t\texttt{.breaks) =>} \\
\quad\quad\quad\quad \texttt{now == \#}I_t\texttt{)))} = C'
\end{array}}{\textsc{trans}(\texttt{while(}E\texttt{) }C,P) = \texttt{assert } P^+\texttt{; while(}E\texttt{) }C'}$$

Soundness Proof We have to prove

$$\begin{array}{ll}
 & [\![\texttt{now.executes}]\!](s) \Rightarrow \\
\text{(a)} & \quad [\![P]\!](s) \wedge [\![\texttt{while(}E\texttt{) }C]\!](s,s') \Rightarrow \\
 & \quad\quad [\![\texttt{assert } P^+\texttt{; while(}E\texttt{) }C']\!](s,s')
\end{array}$$

We assume

(2) $[\![\texttt{now.executes}]\!](s)$

(3) $[\![P]\!](s)$

(4) $[\![\texttt{while(}E\texttt{) }C]\!](s,s')$

By the definition of $[\![\,\_\,]\!]$, it suffices to show

(b.1) $[\![P]\!](s)$

(b.2) $\mathit{executes}(\mathit{control}(s))$

(b.3)
$$\begin{array}{l}
\exists k \in \mathbb{N}, t, u \in \mathit{State}^\infty : \\
\quad \mathit{finiteExecution}(k,t,u,s,[\![E]\!],[\![C']\!]) \wedge \\
\quad ([\![E]\!](t(k)) \neq \textsc{true} \vee \\
\quad\quad \neg(\mathit{executes}(\mathit{control}(u(k))) \vee \\
\quad\quad\quad \mathit{continues}(\mathit{control}(u(k))))) \wedge \\
\quad t(k) = s'
\end{array}$$

From the premises and the induction hypothesis, we know

(5) $E \simeq H$

(6) $C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}$

(7) $J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$

(8) $\$J_1,\ldots,\$J_n$ does not occur in P

(9) $\#I_s, \#I_t$ do not occur in P

(10) $Invariant(G,H,F)_{I_1,\ldots,I_n}$

(11)
$\forall s,s' \in State :$
$[\![\texttt{now.executes}]\!](s) \wedge$
$[\![H$ AND
EXISTS $\$J_1,\ldots,\$J_n$: EXSTATE $\#I_s,\#I_t$:
$P[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]$ AND
$(G[\texttt{now/next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]$
$[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]$ =>
$G[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]$
$[\texttt{now/next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]$ AND
(!(#I_t.continues OR #I_t.breaks) =>
now == #I_t))$]\!](s) \wedge$
$[\![C]\!](s,s') \Rightarrow$
$[\![C']\!](s,s')$

From (3), we know (b.1).

From (4) and the definition of $[\![\_]\!]$, we know (b.2).

From (4) and the definition of $[\![\_]\!]$, we know for some $k \in \mathbb{N}, t,u \in State^\infty$

(12) $finiteExecution(k,t,u,s,[\![E]\!],[\![C]\!])$

(13) $[\![E]\!](t(k)) \neq \text{TRUE} \vee$
$\neg(executes(control(u(k))) \vee continues(control(u(k))))$

(14) $t(k) = s'$

To show (b.3), from (13) and (14), it suffices to show

(b.3.a) $finiteExecution(k,t,u,s,[\![E]\!],[\![C']\!])$

i.e., by the definition of *finiteExecution*,

(b.3.b.1) $t(0) = s$

(b.3.b.2) $u(0) = s$

(b.3.b.3)
$$\begin{array}{l}\forall i \in \mathbb{N}_k : \\ \quad \neg \mathit{breaks}(\mathit{control}(u(i))) \wedge \mathit{executes}(\mathit{control}(t(i))) \wedge \\ \quad [\![E]\!](t(i)) = \text{TRUE} \wedge [\![C']\!](t(i), u(i+1)) \wedge \\ \quad \text{IF } \mathit{continues}(\mathit{control}(u(i+1))) \vee \mathit{breaks}(\mathit{control}(u(i+1))) \\ \quad\quad \text{THEN } t(i+1) = \mathit{execute}(u(i+1)) \\ \quad\quad \text{ELSE } t(i+1) = u(i+1)\end{array}$$

From (12) and the definition of *finiteExecution*, we know

(15) $t(0) = s$

(16) $u(0) = s$

(17)
$$\begin{array}{l}\forall i \in \mathbb{N}_k : \\ \quad \neg \mathit{breaks}(\mathit{control}(u(i))) \wedge \mathit{executes}(\mathit{control}(t(i))) \wedge \\ \quad [\![E]\!](t(i)) = \text{TRUE} \wedge [\![C]\!](t(i), u(i+1)) \wedge \\ \quad \text{IF } \mathit{continues}(\mathit{control}(u(i+1))) \vee \mathit{breaks}(\mathit{control}(u(i+1))) \\ \quad\quad \text{THEN } t(i+1) = \mathit{execute}(u(i+1)) \\ \quad\quad \text{ELSE } t(i+1) = u(i+1)\end{array}$$

From (15), we know (b.3.b.1).

From (16), we know (b.3.b.2).

To show (b.3.b.3), we show for arbitrary $i \in \mathbb{N}_k$

(b.3.b.3.a.1) $\neg \mathit{breaks}(\mathit{control}(u(i))$

(b.3.b.3.a.2) $\mathit{executes}(\mathit{control}(t(i)))$

(b.3.b.3.a.3) $[\![E]\!](t(i)) = \text{TRUE}$

(b.3.b.3.a.4) $[\![C']\!](t(i), u(i+1))$

(b.3.b.3.a.5)
$$\begin{array}{l}\text{IF } \mathit{continues}(\mathit{control}(u(i+1))) \vee \mathit{breaks}(\mathit{control}(u(i+1))) \\ \quad \text{THEN } t(i+1) = \mathit{execute}(u(i+1)) \\ \quad \text{ELSE } t(i+1) = u(i+1)\end{array}$$

From (17), we know (b.3.b.a.1), (b.3.b.a.2), (b.3.b.a.3), (b.3.b.a.5) and

(18) $[\![C]\!](t(i), u(i+1))$

To show (b.3.b.3.a.4), from (11), (18), (b.3.b.3.a.2) and the definition of $[\![\_]\!]$, it suffices to show

$$
\begin{array}{ll}
 & [\![H \ \texttt{AND} \\
 & \quad \texttt{EXISTS } \$J_1,\ldots,\$J_n\texttt{: EXSTATE } \#I_s\texttt{:} \\
 & \quad\quad P[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n] \ \texttt{AND} \\
 & \quad\quad (G[\texttt{now}/\texttt{next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}] \\
\text{(b.3.b.3.a.4.a)} & \quad\quad\quad [\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n] \texttt{ =>} \\
 & \quad\quad\quad G[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n] \\
 & \quad\quad\quad\quad [\texttt{now}/\texttt{next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}] \ \texttt{AND} \\
 & \quad\quad\quad (\texttt{!(\#}I_t\texttt{.continues OR \#}I_t\texttt{.breaks) =>} \\
 & \quad\quad\quad\quad \texttt{now == \#}I_t\texttt{))}]\!](t(i))
\end{array}
$$

i.e. by the definition of $[\![\_]\!]$, for arbitrary $e \in \mathit{Environment}$,

$$
\begin{array}{ll}
\text{(b.3.b.3.a.4.b.1)} & [\![H]\!](e)(t(i),t(i)) \\
 & \exists v_1,\ldots,v_n \in \mathit{Value}, c_s, c_t \in \mathit{Control}: \\
 & \quad [\![P[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n] \ \texttt{AND} \\
 & \quad\quad (G[\texttt{now}/\texttt{next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}] \\
 & \quad\quad\quad [\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n] \texttt{ =>} \\
\text{(b.3.b.3.a.4.b.2)} & \quad\quad\quad G[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n] \\
 & \quad\quad\quad\quad [\texttt{now}/\texttt{next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}] \ \texttt{AND} \\
 & \quad\quad\quad\quad (\texttt{!(\#}I_t\texttt{.continues OR \#}I_t\texttt{.breaks) =>} \\
 & \quad\quad\quad\quad\quad \texttt{now == \#}I_t\texttt{))}]\!] \\
 & \quad\quad (e[I_s \mapsto c_s, I_t \mapsto c_t]_c[J_1 \mapsto v_1,\ldots,J_n \mapsto v_n])(t(i),t(i))
\end{array}
$$

From (5), (17), and the definition of $\simeq$, we know (b.3.b.3.a.4.b.1).

From (6), the soundness of the verification calculus with interruptions, and the definition of $\simeq$, we know

$$
\begin{array}{ll}
(19) & \forall s,s' \in \mathit{Store}: \\
 & \quad \mathit{executes}(\mathit{control}(s)) \wedge [\![C]\!](s,s') \Rightarrow s = s' \ \textsc{except}\ I_1,\ldots,I_n
\end{array}
$$

From (17) and (19), we know

$$(20) \quad \forall i \in \mathbb{N}_k : t(i) = u(i+1) \ \textsc{except}\ I_1,\ldots,I_n$$

From (17), (CD2), and (REE), we know

$$(21) \quad \forall i \in \mathbb{N}_k : t(i+1) \ \textsc{equals}\ u(i+1)$$

From (15), (20), (21), (NEQ), (AVE), and (TRE), we know

$$(22) \quad s = t(i) \ \textsc{except}\ I_1,\ldots,I_n$$

We define

(23) $e_0 := e[I_s \mapsto control(s), I_t \mapsto control(u(i))]_c$
$e_1 := e_0[J_1 \mapsto read(s, I_1), \ldots, J_n \mapsto read(s, I_n)]$

From (23), to show (b.3.b.3.a.4.b.2), it suffices to show

(b.3.b.3.a.4.b.2.a.1) $[\![P[\#I_s/\texttt{now}][\$J_1/I_1, \ldots, \$J_n/I_n]]\!](e_1)(t(i), t(i))$

(b.3.b.3.a.4.b.2.a.2)
$$[\![G[\texttt{now}/\texttt{next}][I_1/I_1', \ldots, I_n/I_n'] \\ [\#I_s/\texttt{now}][\$J_1/I_1, \ldots, \$J_n/I_n]]\!](e_1)(t(i), t(i)) \Rightarrow \\ [\![G[\#I_s/\texttt{now}][\$J_1/I_1, \ldots, \$J_n/I_n] \\ [\texttt{now}/\texttt{next}][I_1/I_1', \ldots, I_n/I_n']]\!](e_1)(t(i), t(i)) \wedge \\ (\neg(continues(control(u(i))) \vee breaks(control(u(i)))) \Rightarrow \\ control(t(i)) = control(u(i)))$$

From (3) and the definition of $[\![\_]\!]$, we know

(24) $[\![P]\!](e)(s, s)$

From (9), (23), (24), (MVF1'), and (CNOF2), we know

(25) $[\![P[\#I_s/\texttt{now}]]\!](e_0)(s, s)$

From (7), (8), (22), (23), (25), (CNOF0), and (PMVF1'), we know

(26) $[\![P[\#I_s/\texttt{now}][\$J_1/I_1, \ldots, \$J_n/I_n]]\!](e_1)(t(i), s)$

From (1b), (26), (PVF2'), and (PVFNE), we know (b.3.b.3.a.4.b.2.a.1).

To show (b.3.b.3.a.4.b.2.a.2), we assume

(27)
$$[\![G[\texttt{now}/\texttt{next}][I_1/I_1', \ldots, I_n/I_n'] \\ [\#I_s/\texttt{now}][\$J_1/I_1, \ldots, \$J_n/I_n]]\!](e_1)(t(i), t(i))$$

and show

(b.3.b.3.a.4.b.2.a.2.a)
$$[\![G[\#I_s/\texttt{now}][\$J_1/I_1, \ldots, \$J_n/I_n] \\ [\#I_t/\texttt{next}][I_1/I_1', \ldots, I_n/I_n']]\!](e_1)(t(i), t(i))$$

(b.3.b.3.a.4.b.2.a.2.b)
$$\neg(continues(control(u(i))) \vee breaks(control(u(i)))) \Rightarrow \\ control(t(i)) = control(u(i))$$

From (15), (16), and (17), we know (b.3.b.3.a.4.b.2.a.2.b).

It remains to show (b.3.b.3.a.4.b.2.a.2.a). From (10) and the definition of *Invariant*, we know

(28) $\$I_1, \ldots, \$I_n, \#I_s, \#I_t$ do not occur in $G, H,$ and F

From (7), (22), (23), (27), (28), (CNEF1), and (PMVF1'), we know

(29) $[\![G[\texttt{now/next}][I_1/I_1\texttt{'}, \ldots, I_n/I_n\texttt{'}][\#I_s/\texttt{now}]]\!](e_0)(s, t(i))$

From (23), (28), (29), (MVF1'), and (CNOF2), we know

(30) $[\![G[\texttt{now/next}][I_1/I_1\texttt{'}, \ldots, I_n/I_n\texttt{'}]]\!](e)(s, t(i))$

We define

(31) $s_0 := \mathit{writes}(t(i), I_1, \mathit{read}(s, I_1), \ldots, I_n, \mathit{read}(s, I_n))$

From (30), (31), and (PPVF1'), we know

(32) $[\![G[\texttt{now/next}]]\!](e)(s, s_0)$

From (32) and (PNNF2), we know

(33) $[\![G]\!](e)(s, (\mathit{store}(s_0), \mathit{control}(s)))$

From (31) and (WSE), we know

(34) $s_0 = t(i)$ EXCEPT $I_1, \ldots, I_n$

From (22), (34), and (TRE), we know

(35) $s_0 = s$ EXCEPT $I_1, \ldots, I_n$

From (31) and (RWE), we know

(36) $\mathit{read}(s_0, I_1) = \mathit{read}(s, I_1) \wedge \ldots \wedge \mathit{read}(s_0, I_n) = \mathit{read}(s, I_n)$

From (35), (36), (RVE), and (NEQ), we know

(37) s_0 EQUALS s

From (33), (37), (CD0), and (CD4), we know

(38) $[\![G]\!](e)(s, s)$

From (6), (10), and (38), we can derive (as demonstrated in the proof of the soundness of the invariant rule in Section 5.4.3)

(39) $[\![G]\!](e)(s,u(i))$

From (23), (28), (39), and (CNOF2), we know

(40) $[\![G[\#I_s/\texttt{now}]]\!](e_0)(s,u(i))$

From (7), (22), (23), (28), (40), (CNOF0), and (PMVF1'), we know

(41) $[\![G[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]]\!](e_1)(t(i),u(i))$

From (23), (28), (41) and (CNEF2), we know

(42) $[\![G[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n][\#I_t/\texttt{next}]]\!](e_1)(t(i),u(i))$

From (15), (16), (17), (REE), and (CD2), we know

(43) $t(i)$ EQUALS $u(i)$

From (43), (CD0), and (CD4), we know

(44) $u(i) = (\mathit{store}(t(i)),\mathit{control}(u(i)))$

From (42), (44), (CNEF0), and (PVFNE), we know

(45) $[\![G[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n][\#I_t/\texttt{next}]]\!](e_1)(t(i),t(i))$

From (45), (IDE), and (PPVF2'), we know (b.3.b.3.a.4.b.2.a.2.a). □

5.9 Expressions and Interruptions

While the mathematical expression $1/0$ just denotes an unknown value, the evaluation of a program expression `1/0` typically lets the program abort with a runtime error. To adequately model this behavior, we first redefine the semantics of expressions and commands to take undefined expressions into account. We then show that the new semantics is (in a certain sense) consistent with the original one.

Subsequently we present two approaches to preserve the validity of the original rules of reasoning without having to consider undefined program expressions: the first one is to verify that the execution of a program does not lead to the evaluation of undefined expressions such that the original semantics of the program coincides with the new semantics. The second one does not a priori rule out states with undefined expressions but modifies a program by introducing "checking commands" such that the interpretation of the modified program is in both versions of the semantics (almost) the same.

Expressions and Interruptions: Definitions

$\bot := \text{SUCH } v : v \notin \mathit{Value}$
$\mathit{Value}_\bot := \mathit{Value} \cup \{\bot\}$
$\mathit{StateFunction}_\bot := \mathit{State} \to \mathit{Value}_\bot$
$\mathit{StatePredicate} := \mathbb{P}(\mathit{State})$

$\mathit{finiteExecution}_\bot :$
$\mathbb{P}(\mathbb{N} \times \mathit{State}^\infty \times \mathit{State}^\infty \times \mathit{State} \times \mathit{StateFunction}_\bot \times \mathit{StateRelation})$
$\mathit{finiteExecution}_\bot(k,t,u,s,E,C) \Leftrightarrow$
$\quad t(0) = s \wedge u(0) = s \wedge$
$\quad \forall i \in \mathbb{N}_k :$
$\qquad \neg \mathit{breaks}(\mathit{control}(u(i))) \wedge \mathit{executes}(\mathit{control}(t(i))) \wedge$
$\qquad E(t(i)) \neq \bot \wedge E(t(i)) = \text{TRUE} \wedge C(t(i), u(i+1)) \wedge$
$\qquad \text{IF } \mathit{continues}(\mathit{control}(u(i+1))) \vee \mathit{breaks}(\mathit{control}(u(i+1)))$
$\qquad\quad \text{THEN } t(i+1) = \mathit{execute}(u(i+1))$
$\qquad\quad \text{ELSE } t(i+1) = u(i+1)$

$\mathit{EXP} := \text{SUCH } k : \ k \in \mathit{Key}$

$\mathit{expthrow} : \mathit{State} \to \mathit{State}$
$\mathit{expthrow}(s) = \mathit{throw}(s, \mathit{EXP}, \mathit{read}(s, \texttt{VAL}))$

$\mathit{expthrows} : \mathbb{P}(\mathit{Control})$
$\mathit{expthrows}(c) \Leftrightarrow \mathit{throws}(c) \wedge \mathit{key}(c) = \mathit{EXP}$

Figure 5.33: Expressions and Interruptions: Definitions

Expressions and Interruptions: Valuation Functions

$[\![\_]\!]_\text{D} : \textbf{Expression} \to \textbf{\textit{StatePredicate}}$
$[\![E]\!]_\text{D}(s) \Leftrightarrow \ldots$

$[\![\_]\!]_\bot : \textbf{Expression} \to \textbf{\textit{StateFunction}}_\bot$
$[\![E]\!]_\bot(s) = \text{IF } [\![E]\!]_\text{D}(s) \text{ THEN } [\![E]\!](s) \text{ ELSE } \bot$

Figure 5.34: Expressions and Interruptions: Valuation Functions (1/3)

Expressions and Interruptions: Valuation Functions (Contd)

$[\![\_]\!]_\perp$: **Command** $\rightarrow$ ***StateRelation***

...

$[\![I{=}E]\!]_\perp(s,s') \Leftrightarrow$
  LET $v = [\![E]\!]_\perp(s)$ IN
  IF $v = \perp$
    THEN $s' = expthrow(s)$
    ELSE $s' = write(s,I,v)$

$[\![\texttt{var}\ I{=}E\ \texttt{;}\ C]\!]_\perp(s,s') \Leftrightarrow$
  LET $v = [\![E]\!]_\perp(s)$ IN
  IF $v = \perp$ THEN
    $s' = expthrow(s)$
  ELSE
    $\exists s_0, s_1 \in State$:
      $s_0 = write(s,I,v) \wedge [\![C]\!]_\perp(s_0,s_1) \wedge$
      $s' = write(s_1,I,read(s,I))$

$[\![\texttt{if (}E\texttt{)}\ C]\!]_\perp(s,s') \Leftrightarrow$
  LET $v = [\![E]\!]_\perp(s)$ IN
  IF $v = \perp$ THEN
    $s' = expthrow(s)$
  ELSE IF $v =$ TRUE THEN
    $[\![C]\!]_\perp(s,s')$
  ELSE
    $s' = s$

$[\![\texttt{if (}E\texttt{)}\ C_1\ \texttt{else}\ C_2]\!]_\perp(s,s') \Leftrightarrow$
  LET $v = [\![E]\!]_\perp(s)$ IN
  IF $v = \perp$ THEN
    $s' = expthrow(s)$
  ELSE IF $v =$ TRUE THEN
    $[\![C_1]\!]_\perp(s,s')$
  ELSE
    $[\![C_2]\!]_\perp(s,s')$

Figure 5.35: Expressions and Interruptions: Valuation Functions (2/3)

Expressions and Interruptions: Valuation Functions (Contd)

$$
\begin{array}{l}
[\![\texttt{return}\ E]\!]_\perp(s,s') \Leftrightarrow \\
\quad \text{LET } v = [\![E]\!]_\perp(s) \text{ IN} \\
\quad \text{IF } v = \perp \\
\qquad \text{THEN } s' = \mathit{expthrow}(s) \\
\qquad \text{ELSE } s' = \mathit{return}(s,v) \\
[\![\texttt{throw}\ I\ E]\!]_\perp(s,s') \Leftrightarrow \\
\quad \text{LET } v = [\![E]\!]_\perp(s) \text{ IN} \\
\quad \text{IF } v = \perp \\
\qquad \text{THEN } s' = \mathit{expthrow}(s) \\
\qquad \text{ELSE } s' = \mathit{throw}(s,I,v) \\
[\![\texttt{while (}E\texttt{) }C]\!]_\perp(s,s') \Leftrightarrow \\
\quad \exists k \in \mathbb{N}, t, u \in \mathit{State}^\infty : \\
\qquad \mathit{finiteExecution}_\perp(k,t,u,s,[\![E]\!]_\perp,[\![C]\!]_\perp) \wedge \\
\qquad ([\![E]\!]_\perp(t(k)) = \perp \vee \\
\qquad\quad [\![E]\!]_\perp(t(k)) \neq \text{TRUE} \vee \\
\qquad\quad \neg(\mathit{executes}(\mathit{control}(u(k))) \vee \\
\qquad\qquad \mathit{continues}(\mathit{control}(u(k))))) \wedge \\
\qquad \text{IF } (\mathit{executes}(\mathit{control}(u(k))) \vee \mathit{continues}(\mathit{control}(u(k)))) \wedge \\
\qquad\quad [\![E]\!]_\perp(t(k)) = \perp \\
\qquad\quad \text{THEN } \mathit{expthrow}(t(k)) = s' \\
\qquad\quad \text{ELSE } t(k) = s'
\end{array}
$$

Figure 5.36: Expressions and Interruptions: Valuation Functions (3/3)

5.9.1 Programs with Undefined Expressions

We redefine the semantics of expressions by a new valuation function $[\![\_]\!]_\perp$: Expression $\to$ *Value*$_\perp$ where *Value*$_\perp$ includes an additional element $\perp$ interpreted as "undefined". The "well-definedness" status of an expression E in a state s is denoted by the formula $[\![E]\!]_{\mathrm{D}}(s)$; if this yields "true", then $[\![E]\!]_\perp$ is an element of *Value*, i.e. different from $\perp$. The behavior of $[\![\_]\!]_{\mathrm{D}}$ is constrained by the following assumption.

Axiom (Well-Defined Expressions) The "well-definedness" status of an expression E only depends on the store:

$$\begin{array}{l}\forall E \in \text{Expression}, s, s' \in \mathit{State} : \\ \quad s \text{ EQUALS } s' \Rightarrow ([\![E]\!]_{\mathrm{D}}(s) \Leftrightarrow [\![E]\!]_{\mathrm{D}}(s'))\end{array}$$

The semantics of commands is correspondingly redefined by a valuation function $[\![\_]\!]_\perp$ which generates a poststate with an "expression evaluation exception" whenever it encounters an expression with an undefined value.

Figures 5.33 to 5.36 list the corresponding auxiliary definitions and the definitions of the valuation functions for expressions and for all commands involving expressions (the rules for the other commands are identical to the original rules except that every application of the valuation function $[\![\_]\!]$ on commands is replaced by an application of $[\![\_]\!]_\perp$).

5.9.2 Relationship to the Original Semantics

The new expression semantics is consistent with the original one in that, if the value of an expression is defined, it is the same in both versions.

Lemma (Partial Expressions)

$$\begin{array}{l}\forall E \in \text{Expression} : \\ \quad \forall s \in \mathit{State} : \\ \quad\quad [\![E]\!]_\perp(s) \neq \perp \Rightarrow [\![E]\!]_\perp(s) = [\![E]\!](s)\end{array}$$

Proof Take arbitrary $E \in$ Expression and $s \in$ *State* and assume

(1) $[\![E]\!]_\perp(s) \neq \perp$

We have to show

(a) $[\![E]\!]_\perp(s) = [\![E]\!](s)$

From (1) and the definitions of $[\![\_]\!]_D$, we know (a). □

Likewise, the new command semantics is consistent with the original one in the sense that, if the execution of a command may yield a poststate that does not indicate an "expression evaluation exception", this poststate is also possible in the original semantics under the provision that the command does not have any subcommand that catches such exceptions.

Lemma (Commands with Partial Expressions)

$$\begin{array}{l}\forall C \in \text{Command}: \\ \quad C \text{ has no subcommand } \texttt{try}\ C_1\ \texttt{catch}\,(EXP\ I)\ C_2 \Rightarrow \\ \quad\quad \forall s, s' \in State : executes(control(s)) \wedge [\![C]\!]_\perp(s, s') \Rightarrow \\ \quad\quad\quad [\![C]\!](s, s') \vee expthrows(control(s'))\end{array}$$

Proof Take arbitrary $C_0 \in \text{Command}$ and $s, s' \in State$ and assume

(1a) C_0 has no subcommand $\texttt{try}\ C_1\ \texttt{catch}\,(EXP\ I)\ C_2$

(1b) $executes(control(s))$

(1c) $[\![C_0]\!]_\perp(s, s')$

(1d) $\neg expthrows(control(s'))$

We have to show

(a) $[\![C_0]\!](s, s')$

We proceed by induction on the structure of command C_0.In the following, we only show those cases where expressions and/or interruptions actually play a role.

- Case $C_0 = I{=}E$: we have to show

  (b) $[\![I{=}E]\!](s, s')$

  We define

  (2) $v := [\![E]\!]_\perp(s)$

  From (1c), (2), and the definition of $[\![\_]\!]_\perp$, we know

$$
\begin{array}{ll}
 & \text{IF } v = \bot \\
(3) & \quad \text{THEN } s' = \mathit{expthrow}(s) \\
 & \quad \text{ELSE } s' = \mathit{write}(s, I, v)
\end{array}
$$

From (1d), (3), and the definitions of *expthrow* and *expthrows*, we know

(4) $v \neq \bot$
(5) $s' = \mathit{write}(s, I, v)$

From (2), (4), and Lemma "Partial Expressions", we know

(6) $v = [\![E]\!](s)$

From (5), (6), and the definition of $[\![\_]\!]$, we know (b).

- Case $C_0 = \texttt{var}\ I\texttt{=}E\texttt{;}\ C$: we have to show

(b) $[\![\texttt{var}\ I\texttt{=}E\texttt{;}\ C]\!](s, s')$

We define

(2) $v := [\![E]\!]_\bot(s)$

From (1c), (2), and the definition of $[\![\_]\!]_\bot$, we know

$$
\begin{array}{ll}
 & \text{IF } v = \bot \text{ THEN} \\
 & \quad s' = \mathit{expthrow}(s) \\
 & \text{ELSE} \\
(3) & \quad \exists s_0, s_1 \in \mathit{State} : \\
 & \qquad s_0 = \mathit{write}(s, I, v) \wedge [\![C]\!]_\bot(s_0, s_1) \wedge \\
 & \qquad s' = \mathit{write}(s_1, I, \mathit{read}(s, I))
\end{array}
$$

From (1d), (3), and the definitions of *expthrow* and *expthrows*, we know for some $s_0, s_1 \in \mathit{State}$

(4) $v \neq \bot$
(5) $s_0 = \mathit{write}(s, I, v)$
(6) $[\![C]\!]_\bot(s_0, s_1)$
(7) $s' = \mathit{write}(s_1, I, \mathit{read}(s, I))$

From (2), (4), and Lemma "Partial Expressions", we know

(8) $v = [\![E]\!](s)$

From (1b), (5), and (CW), we know

(9) $\mathit{executes}(\mathit{control}(s_0))$

From (1d), (7), and (CW), we know

(10) $\neg \mathit{expthrows}(\mathit{control}(s_1))$

From (1a), (6), (9), (10), and the induction hypothesis, we know

(11) $[\![C]\!](s_0, s_1)$

From (5), (7), (11), and the definition of $[\![\_]\!]$, we know (b).

- Case $C_0 = C_1\,;\ C_2$: we have to show

  (b) $[\![C_1\,;\ C_2]\!](s, s')$

  i.e., by the definition of $[\![\_]\!]$,

  (c) $\exists s_0 \in \mathit{State}$:
  $[\![C_1]\!](s, s_0) \wedge$
  IF $\mathit{executes}(\mathit{control}(s_0))$ THEN $[\![C_2]\!](s_0, s')$ ELSE $s' = s_0$

  From (1c) and the definition of $[\![\_]\!]_\perp$, we know for some $s_0 \in \mathit{State}$

  (2) $[\![C_1]\!]_\perp(s, s_0)$
  (3) IF $\mathit{executes}(\mathit{control}(s_0))$ THEN $[\![C_2]\!]_\perp(s_0, s')$ ELSE $s' = s_0$

  We proceed by case distinction:

  - Case $\mathit{executes}(\mathit{control}(s_0))$: from this condition and (3), we know

    (4) $[\![C_2]\!]_\perp(s_0, s')$

    From (1a), (1d), (4), the case condition, and the induction hypothesis, we know

    (5) $[\![C_2]\!](s_0, s')$

    From the case condition and the definitions of *executes* and *expthrows*, we know

    (6) $\neg \mathit{expthrows}(\mathit{control}(s_0))$

    From (1a), (1b), (2), (6), and the induction hypothesis, we know

    (7) $[\![C_1]\!](s, s_0)$

    From (5), (7), and the case condition, we know (c).
  - Case $\neg \mathit{executes}(\mathit{control}(s_0))$: from the case condition and the definition of (4), we know

    (8) $s' = s_0$

    From (1d) and (8), we know

    (9) $\neg \mathit{expthrows}(\mathit{control}(s_0))$

    From (1a), (1b), (2), (9), and the induction hypothesis, we know

    (10) $[\![C_1]\!](s, s_0)$

    From (8), (10), and the case condition, we know (c).

- Case $C_0 =$ `if (`E`)` C: we have to show

(b) $[\![\texttt{if (}E\texttt{) }C]\!](s,s')$

i.e., by the definition of $[\![\texttt{␣}]\!]$,

(c) IF $[\![E]\!](s) =$ TRUE THEN $[\![C]\!](s,s')$ ELSE $s' = s$

We define

(2) $v = [\![E]\!]_\perp(s)$

From (1c) and the definition of $[\![\texttt{␣}]\!]_\perp$, we know

(3)
IF $v = \perp$ THEN
  $s' = \mathit{expthrow}(s)$
ELSE IF $v =$ TRUE THEN
  $[\![C]\!]_\perp(s,s')$
ELSE
  $s' = s$

From (1d), (3), and the definitions of *expthrow* and *expthrows*, we know

(4) $v \neq \perp$

(5)
IF $v =$ TRUE
  THEN $[\![C]\!]_\perp(s,s')$
  ELSE $s' = s$

From (2), (4), and Lemma "Partial Expressions", we know

(6) $v = [\![E]\!](s)$

We proceed by case distinction:

- Case $v =$ TRUE: from the case condition and (5), we know
  (7) $[\![C]\!]_\perp(s,s')$
  From (1a), (1b), (1d), (7), and the induction hypothesis, we know
  (8) $[\![C]\!](s,s')$
  From (6), (8), and the case condition, we know (c).
- Case $v \neq$ TRUE: from the case condition and (5), we know
  (9) $s' = s$
  From (6), (9), and the case condition, we know (c).

• Case $C_0 = \texttt{if (}E\texttt{) }C_1\texttt{ else }C_2$: we have to show

(b) $[\![\texttt{if (}E\texttt{) }C_1\texttt{ else }C_2]\!](s,s')$

i.e., by the definition of $[\![\texttt{␣}]\!]$,

(c) IF $[\![E]\!](s) =$ TRUE THEN $[\![C_1]\!](s,s')$ ELSE $[\![C_2]\!](s,s')$

We define

(2) $v = [\![E]\!]_\perp(s)$

From (1c) and the definition of $[\![\_]\!]_\perp$, we know

(3)
IF $v = \perp$ THEN
  $s' = expthrow(s)$
ELSE IF $v =$ TRUE THEN
  $[\![C_1]\!]_\perp(s, s')$
ELSE
  $[\![C_2]\!]_\perp(s, s')$

From (1d), (3), and the definitions of *expthrow* and *expthrows*, we know

(4) $v \neq \perp$

(5)
IF $v =$ TRUE
  THEN $[\![C_1]\!]_\perp(s, s')$
  ELSE $[\![C_2]\!]_\perp(s, s')$

From (2), (4), and Lemma "Partial Expressions", we know

(6) $v = [\![E]\!](s)$

We proceed by case distinction:

- Case $v =$ TRUE: from the case condition and (5), we know
  (7) $[\![C_1]\!]_\perp(s, s')$
  From (1a), (1b), (1d), (7), and the induction hypothesis, we know
  (8) $[\![C_1]\!](s, s')$
  From (6), (8), and the case condition, we know (c).
- Case $v \neq$ TRUE: from the case condition and (5), we know
  (9) $[\![C_2]\!]_\perp(s, s')$
  From (1a), (1b), (1d), (9), and the induction hypothesis, we know
  (10) $[\![C_2]\!](s, s')$
  From (6), (10), and the case condition, we know (c).

- Case $C_0 =$ `return` E: we have to show

  (b) $[\![\texttt{return}\ E]\!](s, s')$

  We define

  (2) $v := [\![E]\!]_\perp(s)$

  From (1c), (2), and the definition of $[\![\_]\!]_\perp$, we know

$$\begin{array}{ll} & \text{IF } v = \bot \\ (3) & \quad \text{THEN } s' = \mathit{expthrow}(s) \\ & \quad \text{ELSE } s' = \mathit{return}(s, v) \end{array}$$

From (1d), (3), and the definitions of *expthrow* and *expthrows*, we know

(4) $v \neq \bot$
(5) $s' = \mathit{return}(s, v)$

From (2), (4), and Lemma "Partial Expressions", we know

(6) $v = [\![E]\!](s)$

From (5), (6), and the definition of $[\![\_]\!]$, we know (b).

- Case $C_0 = \texttt{throw}\ I\ E$: we have to show

(b) $[\![\texttt{throw}\ I\ E]\!](s, s')$

We define

(2) $v := [\![E]\!]_\bot(s)$

From (1c), (2), and the definition of $[\![\_]\!]_\bot$, we know

$$\begin{array}{ll} & \text{IF } v = \bot \\ (3) & \quad \text{THEN } s' = \mathit{expthrow}(s) \\ & \quad \text{ELSE } s' = \mathit{throw}(s, I, v) \end{array}$$

From (1d), (3), and the definitions of *expthrow* and *expthrows*, we know

(4) $v \neq \bot$
(5) $s' = \mathit{throw}(s, I, v)$

From (2), (4), and Lemma "Partial Expressions", we know

(6) $v = [\![E]\!](s)$

From (5), (6), and the definition of $[\![\_]\!]$, we know (b).

- Case $C_0 = \texttt{try}\ C_1\ \texttt{catch}\,(I_k\ I_v)\ C_2$: we have to show

(b) $[\![\texttt{try}\ C_1\ \texttt{catch}\,(I_k\ I_v)\ C_2]\!](s, s')$

i.e., by the definition of $[\![\_]\!]$,

$$
\begin{array}{ll}
 & \exists s_0, s_1, s_2 \in \mathit{State} : \\
 & \quad [\![C_1]\!](s, s_0) \wedge \\
 & \quad \text{IF } \mathit{throws}(\mathit{control}(s_0)) \wedge \mathit{key}(\mathit{control}(s_0)) = I_k \text{ THEN} \\
\text{(c)} & \qquad s_1 = \mathit{write}(\mathit{execute}(s_0), I_v, \mathit{value}(\mathit{control}(s_0))) \wedge \\
 & \qquad [\![C_2]\!](s_1, s_2) \wedge \\
 & \qquad s' = \mathit{write}(s_2, I_v, \mathit{read}(s_0, I_v)) \\
 & \quad \text{ELSE } s' = s_0
\end{array}
$$

From (1c) and the definition of $[\![\_]\!]_\perp$, we know for some $s_0, s_1, s_2 \in \mathit{State}$

$$
\begin{array}{ll}
\text{(2)} & [\![C_1]\!]_\perp(s, s_0) \\
 & \text{IF } \mathit{throws}(\mathit{control}(s_0)) \wedge \mathit{key}(\mathit{control}(s_0)) = I_k \text{ THEN} \\
 & \qquad s_1 = \mathit{write}(\mathit{execute}(s_0), I_v, \mathit{value}(\mathit{control}(s_0))) \wedge \\
\text{(3)} & \qquad [\![C_2]\!]_\perp(s_1, s_2) \wedge \\
 & \qquad s' = \mathit{write}(s_2, I_v, \mathit{read}(s_0, I_v)) \\
 & \quad \text{ELSE } s' = s_0
\end{array}
$$

We proceed by case distinction:

- Case $\mathit{throws}(\mathit{control}(s_0)) \wedge \mathit{key}(\mathit{control}(s_0)) = I_k$: from the case condition and (3), we know

  (4) $s_1 = \mathit{write}(\mathit{execute}(s_0), I_v, \mathit{value}(\mathit{control}(s_0)))$
  (5) $[\![C_2]\!]_\perp(s_1, s_2)$
  (6) $s' = \mathit{write}(s_2, I_v, \mathit{read}(s_0, I_v))$

  From (4), (CD1), and (CW), we know

  (7) $\mathit{executes}(\mathit{control}(s_1))$

  From (1d), (6), (CW), and the definition of *expthrows*, we know

  (8) $\neg \mathit{expthrows}(\mathit{control}(s_2))$

  From (1a), (5), (7), (8), and the induction hypothesis, we know

  (9) $[\![C_2]\!](s_1, s_2)$

  From (1a), we know

  (10) $I_k \neq \mathit{EXP}$

  From (10), the case condition, and the definition of *expthrows*, we know

  (11) $\neg \mathit{expthrows}(\mathit{control}(s_0))$

  From (1a), (1b), (2), (11), and the induction hypothesis, we know

  (12) $[\![C_1]\!](s, s_0)$

  From (4), (6), (9), (12), and the case condition, we know (c).

- Case $\neg(\mathit{throws}(\mathit{control}(s_0)) \wedge \mathit{key}(\mathit{control}(s_0)) = I_k)$: from the case condition and (3), we know

  (13) $s' = s_0$

From (1d) and (13), we know

(14) $\neg \mathit{expthrows}(\mathit{control}(s_0))$

From (1a), (1b), (2), (14), and the induction hypothesis, we know

(15) $[\![C_1]\!](s, s_0)$

From (13), (15), and the case condition, we know (c).

- Case $C_0 = \texttt{while (}E\texttt{) }C$: we have to show

(b) $[\![\texttt{while (}E\texttt{) }C]\!](s, s')$

i.e., by the definition of $[\![\_]\!]$,

(c)
$$\begin{array}{l}\exists k \in \mathbb{N}, t, u \in \mathit{State}^\infty : \\ \quad \mathit{finiteExecution}(k, t, u, s, [\![E]\!], [\![C]\!]) \wedge \\ \quad ([\![E]\!](t(k)) \neq \text{TRUE} \vee \\ \qquad \neg(\mathit{executes}(\mathit{control}(u(k))) \vee \\ \qquad\quad \mathit{continues}(\mathit{control}(u(k))))) \wedge \\ \quad t(k) = s'\end{array}$$

From (1c) and the definition of $[\![\_]\!]_\perp$, we know for some $k \in \mathbb{N}, t, u \in \mathit{State}^\infty$

(2) $\mathit{finiteExecution}_\perp(k, t, u, s, [\![E]\!]_\perp, [\![C]\!]_\perp)$

(3)
$$\begin{array}{l}[\![E]\!]_\perp(t(k)) = \perp \vee \\ [\![E]\!]_\perp(t(k)) \neq \text{TRUE} \vee \\ \neg(\mathit{executes}(\mathit{control}(u(k))) \vee \mathit{continues}(\mathit{control}(u(k))))\end{array}$$

(4)
$$\begin{array}{l}\text{IF } (\mathit{executes}(\mathit{control}(u(k))) \vee \mathit{continues}(\mathit{control}(u(k)))) \wedge \\ \quad [\![E]\!]_\perp(t(k)) = \perp \\ \quad \text{THEN } \mathit{expthrow}(t(k)) = s' \\ \quad \text{ELSE } t(k) = s'\end{array}$$

From (4), (1d), and the definitions of *expthrows* and *expthrow*, we know

(5)
$$\begin{array}{l}\neg((\mathit{executes}(\mathit{control}(u(k))) \vee \mathit{continues}(\mathit{control}(u(k)))) \wedge \\ \quad [\![E]\!]_\perp(t(k)) = \perp)\end{array}$$

(6) $t(k) = s'$

To show (c), it suffices to show

(d.1) $\mathit{finiteExecution}(k, t, u, s, [\![E]\!], [\![C]\!])$

(d.2)
$$\begin{array}{l}[\![E]\!](t(k)) \neq \text{TRUE} \vee \\ \neg(\mathit{executes}(\mathit{control}(u(k))) \vee \mathit{continues}(\mathit{control}(u(k))))\end{array}$$

(d.3) $t(k) = s'$

From (2) and the definition of $\mathit{finiteExecution}_\perp$, we know

(7) $t(0) = s$

(8) $u(0) = s$
$\forall i \in \mathbb{N}_k :$
$\neg \mathit{breaks}(\mathit{control}(u(i))) \wedge \mathit{executes}(\mathit{control}(t(i))) \wedge$
$[\![E]\!]_\perp(t(i)) \neq \perp \wedge$
(9) $[\![E]\!]_\perp(t(i)) = \text{TRUE} \wedge [\![C]\!]_\perp(t(i), u(i+1)) \wedge$
IF $\mathit{continues}(\mathit{control}(u(i+1))) \vee \mathit{breaks}(\mathit{control}(u(i+1)))$
THEN $t(i+1) = \mathit{execute}(u(i+1))$
ELSE $t(i+1) = u(i+1)$

From (9), we know

(10) $\forall i \in \mathbb{N}_k : [\![E]\!]_\perp(t(i)) \neq \perp$
(11) $\forall i \in \mathbb{N}_k : [\![E]\!]_\perp(t(i)) = \text{TRUE}$
(12) $\forall i \in \mathbb{N}_k : \mathit{executes}(\mathit{control}(t(i)))$
(13) $\forall i \in \mathbb{N}_k : [\![C]\!]_\perp(t(i), u(i+1))$

From (10), and Lemma "Partial Expressions", we know

(14) $\forall i \in \mathbb{N}_k : [\![E]\!]_\perp(t(i)) = [\![E]\!](t(i))$

From (11) and (14), we know

(15) $\forall i \in \mathbb{N}_k : [\![E]\!](t(i)) = \text{TRUE}$

From (1d) and (6), we know

(16) $\neg \mathit{expthrows}(\mathit{control}(t(k)))$

From (9), (16), and the definition of *expthrows*, we can show

(17) $\forall i \in \mathbb{N}_k : \neg \mathit{expthrows}(\mathit{control}(u(i+1)))$

From (1a), (12), (13), (17) and the induction hypothesis, we know

(18) $\forall i \in \mathbb{N}_k : [\![C]\!](t(i), u(i+1))$

From (7), (8), (9), (15), (18), and the definition of *finiteExecution*, we know (d.1).

To show (d.2), we proceed by case distinction:

- Case $[\![E]\!]_\perp(t(k)) = \perp$: from the case condition and (5), we know
  (19) $\neg(\mathit{executes}(\mathit{control}(u(k))) \vee \mathit{continues}(\mathit{control}(u(k))))$
  From (19), we know (d.2).
- Case $[\![E]\!]_\perp(t(k)) \neq \perp$: from the case condition and (3), know (d.2).

From (6), we know (d.3). □

Well-Defined Expressions: Definitions

$$\_ \overset{\mathrm{D}}{\simeq} \_ : \mathbb{P}(\text{Expression} \times \text{Formula})$$
$$E \overset{\mathrm{D}}{\simeq} F_D \Leftrightarrow$$
$$\quad F_D \text{ does not depend on the poststate} \wedge$$
$$\quad \forall s \in \mathit{State} : [\![E]\!]_{\mathrm{D}}(s) \Leftrightarrow [\![F_D]\!](s)$$

Well-Defined Expressions: Judgements

$$C \checkmark P \Leftrightarrow$$
$$\quad \forall s, s' \in \mathit{State} : \mathit{executes}(\mathit{control}(s)) \wedge [\![P]\!](s) \Rightarrow$$
$$\quad\quad ([\![C]\!](s, s') \Leftrightarrow [\![C]\!]_{\perp}(s, s'))$$

Figure 5.37: Definitions for Well-Defined Expressions

5.9.3 Avoiding Undefined Expressions

As shown in the previous section, the program semantics that takes into account undefined program expressions is related but not identical to the original program semantics: in the new semantics programs may yield additional "expression evaluation exception" states. Consequently one can expect that the rules of reasoning for the new semantics are related to the rules of the original semantics but become substantially more complicated.

One may however argue that programs that yield such exceptional states are erroneous anyway. Rather than defining new rules to deal with these states, we might thus concentrate our efforts on avoiding these states. If we can ensure that these states do not occur in a program run, we can resort to the simpler program semantics and the corresponding simpler rules of reasoning.

In pursue of this goal, Figure 5.37 introduces a new kind of judgement $C \checkmark P$ that informally states that the execution of command C in a state in which formula P holds does not lead to the evaluation of any undefined expressions. Formally, the judgement guarantees that on such states the original semantics of C coincides with the semantics of C where exceptions are raised when undefined expressions are encountered.

Figures 5.38 to 5.41 give the rules for deriving such judgements. In essence, they are generalizations of the rules for computing assertions presented in Section 5.8 that forward from a command the knowledge about the state in which the com-

mand is executed to its subcommands; rather than constructing annotations, this information is now used to determine whether an expression that has to be evaluated in a state is indeed well-defined in that state. The rules make use of the relation $\overset{\mathrm{D}}{\simeq}$ introduced in Figure 5.37 that relates an expression E to a formula P which is true if and only if the expression is well-defined.

The rules ultimately yield proof obligations of the form

$$\forall s \in \mathit{State} : [\![(\texttt{now.executes AND } P) \texttt{ => } F_D]\!](s)$$

where P is a condition guaranteed on the current state and F_D defines the well-definedness condition of an expression to be evaluated.

Most rules are fairly obvious (comparing them with the rules presented in Section 5.8 and the definition of $[\![\_]\!]_\perp$), the major exception are the two rules for loops (with and without an invariant): the loop expression not only has to be well-defined in the initial state but also after every loop iteration. For this purpose, from the condition R known to hold at the beginning of the execution of the loop body, the postcondition Q holding after the execution of the body is derived.

The soundness claim for the presented calculus is stated below.

Lemma (Soundness of Well-Defined Expressions) If a judgement $C \checkmark P$ can be derived, then the following is true:

$$\begin{aligned}\forall s, s' \in \mathit{State} : \mathit{executes}(\mathit{control}(s)) \wedge [\![P]\!](s) \Rightarrow \\ ([\![C]\!](s,s') \Leftrightarrow [\![C]\!]_\perp(s,s'))\end{aligned}$$

Proof Remains open but is related to the proof of Theorem "Soundness of the Assertion Calculus". □

5.9.4 Checking for Undefined Expressions

In the previous subsection, we have described how to avoid the necessity to deal with undefined program expressions by verifying that they cannot occur. However, in real programs the exception raised by the evaluation of an undefined expression need not necessarily lead to immediate program abortion but it may be caught by a handler and processed such that the program may continue in a normal way. Therefore we now investigate another possibility of dealing with undefined program expressions: rather than verifying statically that these expressions cannot occur, we "protect" the evaluation of every expression by a preceding runtime check that raises an exception if the expression is undefined.

Well-Defined Expressions: Rules

$$\frac{E \overset{\text{D}}{\simeq} F_D \quad \forall s \in \mathit{State} : [\![(\texttt{now.executes AND } P) \texttt{ => } F_D]\!](s)}{I\texttt{=}E \checkmark P}$$

$$\frac{\$J \text{ does not occur in } P \quad C \checkmark \texttt{EXISTS } \$J : P[\$J/I]}{\texttt{var } I\texttt{; } C \checkmark P}$$

$$\frac{E \overset{\text{D}}{\simeq} F_D \quad \forall s \in \mathit{State} : [\![(\texttt{now.executes AND } P) \texttt{ => } F_D]\!](s) \quad \$J \text{ does not occur in } P \text{ and } T \quad C \checkmark \texttt{EXISTS } \$J : P[\$J/I] \texttt{ AND } I = T[\$J/I]}{\texttt{var } I\texttt{=}E\texttt{; } C \checkmark P}$$

$$\frac{\text{POST}(C_1, P) = Q \quad C_1 \checkmark P \quad C_2 \checkmark Q}{C_1\texttt{;}\, C_2 \checkmark P}$$

$$\frac{E \overset{\text{D}}{\simeq} F_D \quad \forall s \in \mathit{State} : [\![(\texttt{now.executes AND } P) \texttt{ => } F_D]\!](s) \quad E \simeq F \quad C \checkmark P \texttt{ AND } F}{\texttt{if (}E\texttt{) } C \checkmark P}$$

$$\frac{E \overset{\text{D}}{\simeq} F_D \quad \forall s \in \mathit{State} : [\![(\texttt{now.executes AND } P) \texttt{ => } F_D]\!](s) \quad E \simeq F \quad C_1 \checkmark P \texttt{ AND } F \quad C_1 \checkmark P \texttt{ AND !}F}{\texttt{if (}E\texttt{) } C_1 \texttt{ else } C_2 \checkmark P}$$

Figure 5.38: Rules for Well-Defined Expressions (1/4)

Well-Defined Expressions: Rules (Contd)

`continue` $\checkmark P$

`break` $\checkmark P$

$$\frac{E \simeq F_D \quad \forall s \in \mathit{State} : [\![(\texttt{now.executes AND } P) \texttt{ => } F_D]\!](s)}{\texttt{return } E \checkmark P}$$

$$\frac{E \simeq F_D \quad \forall s \in \mathit{State} : [\![(\texttt{now.executes AND } P) \texttt{ => } F_D]\!](s)}{\texttt{throw } I_k\ E \checkmark P}$$

$$\frac{\begin{array}{l}\$J \text{ does not occur in } Q \\ \#I_s \text{ does not occur in } Q \\ \textsc{post}(C_1, P) = Q \\ C_1 \checkmark P \\ C_2 \checkmark \\ \quad \texttt{EXISTS } \$J\texttt{: EXSTATE } \#I_s\texttt{:} \\ \qquad Q[\#I_s/\texttt{now}][\$J/I_v] \texttt{ AND} \\ \qquad \#I_s\texttt{.throws } I_k \texttt{ AND } \#I_s\texttt{.value} = I_v\end{array}}{\texttt{try } C_1 \texttt{ catch(}I_k\ I_v\texttt{) } C_2 \checkmark P}$$

Figure 5.39: Rules for Well-Defined Expressions (2/4)

Well-Defined Expressions: Rules (Contd)

$E \overset{\mathrm{D}}{\simeq} F_D$
$\forall s \in State : [\![(\texttt{now.executes AND } P) \texttt{ => } F_D]\!](s)$
$E \simeq H$
$C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}$
$J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$
$\$J_1,\ldots,\$J_n$ do not occur in P
$\#I_s$ does not occur in P
R =
  H AND
  EXISTS $\$J_1,\ldots,\$J_n$: EXSTATE $\#I_s$:
    $P[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]$
$C \checkmark R$
$\textsc{post}(C,R) = Q$
$\forall s \in State$:
  $[\![((\texttt{now.executes OR now.continues})$
    AND $Q)$ => $F_D]\!](s)$

---

while(E) $C \checkmark P$

Figure 5.40: Rules for Well-Defined Expressions (3/4)

Well-Defined Expressions: Rules (Contd)

$E \overset{\text{D}}{\simeq} F_D$
$\forall s \in State : [\![(\texttt{now.executes AND } P) \texttt{ => } F_D]\!](s)$
$E \simeq H$
$C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}$
$J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n$
$\$J_1,\ldots,\$J_n$ do not occur in P
$\#I_s, \#I_t$ do not occur in P
$Invariant(G,H,F)_{I_1,\ldots,I_n}$
$R =$
H AND
EXISTS $\$J_1,\ldots,\$J_n$: EXSTATE $\#I_s, \#I_t$:
$P[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]$ AND
$(G[\texttt{now}/\texttt{next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]$
$[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]$ =>
$G[\#I_s/\texttt{now}][\$J_1/I_1,\ldots,\$J_n/I_n]$
$[\#I_t/\texttt{next}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}]$ AND
(!(#I_t.continues OR #I_t.breaks) =>
now == #I_t))
$C \checkmark R$
$\text{POST}(C,R) = Q$
$\forall s \in State$:
$[\![((\texttt{now.executes OR now.continues})$
AND $Q)$ => $F_D]\!](s)$

---

while(E) $C \checkmark P$

Figure 5.41: Rules for Well-Defined Expressions (4/4)

Checked Program Calculus: Judgements

$$\text{CHECK}(C) = C' \Leftrightarrow$$
$$\quad \forall s, s' \in \mathit{State} : \mathit{executes}(\mathit{control}(s)) \Rightarrow$$
$$\quad\quad [\![C]\!]_\perp(s, s') \Leftrightarrow [\![C']\!](s, s')$$

Checked Program Calculus: Definitions

$$\llcorner \overset{\text{D}}{\simeq} \lrcorner : \mathbb{P}(\text{Expression} \times \text{Expression})$$
$$E \overset{\text{D}}{\simeq} E_D :\Leftrightarrow \forall s \in \mathit{State} : [\![E]\!]_\text{D}(s) \Leftrightarrow [\![E_D]\!](s) = \text{TRUE}$$

$$\text{CHECK } E_D \equiv \texttt{if (!}E_D\texttt{) throw } \mathit{EXP}\ \texttt{VAL}$$

Figure 5.42: Checked Program Calculus (1/4)

In more detail, our strategy is as follows: the definition of the expression value $[\![E]\!]_\perp$ is based on a predicate $[\![E]\!]_\text{D}$ which determines those states in which it is safe to determine the expression value $[\![E]\!]$ without risking program abortion. Assume we can give an expression E_D which evaluates to TRUE in exactly these states. Then all expression evaluations may be guarded by a pseudo-command CHECK E_D which raises an "expression valuation exception", if the evaluation of E_D does not yield TRUE. Thus it may become possible to preserve the original program semantics but apply it to the modified program that "simulates" the program semantics with undefined expressions; consequently the original rules of reasoning remain still valid (but are applied to the modified program).

Based on this idea, Figures 5.42 and 5.43 introduce a calculus of judgements $\text{CHECK}(C) = C'$ for the derivation of a "checked program" C' from a program C; the semantics of C in the model with expression interruptions is the same as the semantics of C' in the original model. The listed rules cover only the crucial commands (with the rules for loops being delegated to the following subsection); for all other commands, C' is the same as C (with all subcommands replaced by their checked counterparts).

The pseudo-command CHECK E_D can be translated into the regular command `if (!`E_D`) throw` *EXP* `VAL` which may trigger an exception with key *EXP*.

The following lemma states the crucial relationship between checked programs and their unchecked counterparts.

Lemma (Checked Commands) If a judgement $\text{CHECK}(C) = C'$ can be derived from the rules of the checked program calculus, then we have

Checked Program Calculus: Rules

$$\frac{E \overset{\text{D}}{\simeq} E_D}{\text{CHECK}(I = E) = \text{CHECK } E_D\texttt{;}\ I = E}$$

$$\frac{E \overset{\text{D}}{\simeq} E_D \quad \text{CHECK}(C) = C'}{\text{CHECK}(\texttt{var } I\texttt{=}E\texttt{;}\ C) = \text{CHECK } E_D\texttt{;}\ \texttt{var } I\texttt{=}E\texttt{;}\ C'}$$

$$\frac{E \overset{\text{D}}{\simeq} E_D \quad \text{CHECK}(C) = C'}{\text{CHECK}(\texttt{if (}E\texttt{)}\ C) = \text{CHECK } E_D\texttt{;}\ \texttt{if (}E\texttt{)}\ C'}$$

$$\frac{E \overset{\text{D}}{\simeq} E_D \quad \text{CHECK}(C_1) = C_1' \quad \text{CHECK}(C_2) = C_2'}{\begin{array}{l}\text{CHECK}(\texttt{if (}E\texttt{)}\ C_1\ \texttt{else}\ C_2) = \\ \quad \text{CHECK } E_D\texttt{;}\ \texttt{if (}E\texttt{)}\ C_1'\ \texttt{else}\ C_2'\end{array}}$$

$$\frac{E \overset{\text{D}}{\simeq} E_D}{\text{CHECK}(\texttt{return } E) = \text{CHECK } E_D\texttt{;}\ \texttt{return } E}$$

$$\frac{E \overset{\text{D}}{\simeq} E_D}{\text{CHECK}(\texttt{throw } I\ E) = \text{CHECK } E_D\texttt{;}\ \texttt{throw } I\ E}$$

$$\frac{\ldots}{\text{CHECK}(\texttt{while (}E\texttt{)}\ C) = \ldots}$$

Figure 5.43: Checked Program Calculus (2/4)

$$C \text{ has no subcommand } \texttt{try}\ C_1\ \texttt{catch}\,(EXP\ I)\ C_2 \Rightarrow \\ \forall s, s' \in State : executes(control(s)) \Rightarrow \\ [\![C]\!]_\perp(s,s') \Leftrightarrow [\![C']\!](s,s')$$

The constraint "C has no subcommand $\texttt{try}\ C_1\ \texttt{catch}\,(EXP\ I)\ C_2$" is needed for dealing with checked loops as described in the next section (actually, it is stronger than necessary, as will be explained later).

Proof Take arbitrary commands C_0 and C_0' such that $\textsc{check}(C_0) = C_0'$ can be derived. Take arbitrary $s, s' \in State$ and assume

(1a) C_0 has no subcommand $\texttt{try}\ C_1\ \texttt{catch}\,(EXP\ I)\ C_2$

(1b) $executes(control(s))$

We show

(a) $[\![C_0]\!]_\perp(s,s') \Leftrightarrow [\![C']\!](s,s')$

We proceed by induction on the structure of C_0. In the following, we only cover the cases where the form of C_0' differs from C_0; for the other cases, the result follows directly from the induction hypothesis.

- Case $C_0 = I = E$: we have to show

  (b) $[\![I = E]\!]_\perp(s,s') \Leftrightarrow [\![\textsc{check}\ E_D\texttt{;}\ I = E]\!](s,s')$

  From the rule hypothesis, we know

  (2) $E \overset{\mathrm{D}}{\simeq} E_D$

  From the definitions of $[\![\_]\!]_\perp$, `check`, and $[\![\_]\!]_\perp$, we know

  (3)
  $$[\![I = E]\!]_\perp(s,s') \Leftrightarrow \\ \quad \textsc{let}\ v = [\![E]\!]_\perp(s)\ \textsc{in} \\ \quad \textsc{if}\ v = \perp \\ \quad\quad \textsc{then}\ s' = expthrow(s) \\ \quad\quad \textsc{else}\ s' = write(s, I, v)$$

  (4)
  $$[\![\textsc{check}\ E_D\texttt{;}\ I = E]\!](s,s') \Leftrightarrow \\ \quad \textsc{if}\ [\![E_D]\!](s) \neq \textsc{true} \\ \quad\quad \textsc{then}\ s' = throw(s, EXP, read(s, \texttt{VAL})) \\ \quad\quad \textsc{else}\ s' = write(s, I, [\![E]\!](s))$$

  From (2) and the definitions of $\overset{\mathrm{D}}{\simeq}$ and $[\![\_]\!]_\perp$, we know

(5) $[\![E_D]\!](s) \neq \text{TRUE} \Leftrightarrow [\![E]\!]_\perp(s) = \perp$

From (3), (4), (5), the definition of *expthrow*, and Lemma "Partial Expressions", we know (b).

- Case $C_0 = \texttt{var}\ I\texttt{=}E\texttt{;}\ C$: we have to show

  (b) $[\![\texttt{var}\ I\texttt{=}E\texttt{;}\ C]\!]_\perp(s,s') \Leftrightarrow [\![\text{CHECK}\ E_D\texttt{;}\ \texttt{var}\ I\texttt{=}E\texttt{;}\ C']\!](s,s')$

  From the rule hypotheses, we know

  (2) $E \overset{\text{D}}{\simeq} E_D$
  (3) $\text{CHECK}(C) = C'$

  From the definitions of $[\![\_]\!]_\perp$, `check`, and $[\![\_]\!]_\perp$, we know

(4)
$$
\begin{array}{l}
[\![\texttt{var}\ I\texttt{=}E\texttt{;}\ C]\!]_\perp(s,s') \Leftrightarrow \\
\quad \text{LET } v = [\![E]\!]_\perp(s) \text{ IN} \\
\quad \text{IF } v = \perp \text{ THEN} \\
\qquad s' = \mathit{expthrow}(s) \\
\quad \text{ELSE} \\
\qquad \exists s_0, s_1 \in \mathit{State}: \\
\qquad\quad s_0 = \mathit{write}(s, I, v) \wedge [\![C]\!]_\perp(s_0, s_1) \wedge \\
\qquad\quad s' = \mathit{write}(s_1, I, \mathit{read}(s, I))
\end{array}
$$

(5)
$$
\begin{array}{l}
[\![\text{CHECK}\ E_D\texttt{;}\ \texttt{var}\ I\texttt{=}E\texttt{;}\ C']\!](s,s') \Leftrightarrow \\
\quad \text{IF } [\![E_D]\!](s) \neq \text{TRUE THEN} \\
\qquad s' = \mathit{throw}(s, \mathit{EXP}, \mathit{read}(s, \texttt{VAL})) \\
\quad \text{ELSE} \\
\qquad \exists s_0, s_1 \in \mathit{State}: \\
\qquad\quad s_0 = \mathit{write}(s, I, [\![E]\!](s)) \wedge [\![C']\!](s_0, s_1) \wedge \\
\qquad\quad s' = \mathit{write}(s_1, I, \mathit{read}(s, I))
\end{array}
$$

  From (2) and the definitions of $\overset{\text{D}}{\simeq}$ and $[\![\_]\!]_\perp$, we know

  (6) $[\![E_D]\!](s) \neq \text{TRUE} \Leftrightarrow [\![E]\!]_\perp(s) = \perp$

  We proceed by case distinction:

  - Case $[\![E]\!]_\perp(s) = \perp$: From the case condition, (4), (5), (6), and the definition of *expthrow*, we know (b).
  - Case $[\![E]\!]_\perp(s) \neq \perp$: From the case condition, (1b), (3), (4), (5), (6), and the induction hypothesis, we know (b).

- Case $C_0 = \texttt{if (}E\texttt{)}\ C$: we have to show

  (b) $[\![\texttt{if (}E\texttt{)}\ C]\!]_\perp(s,s') \Leftrightarrow [\![\text{CHECK}\ E_D\texttt{;}\ \texttt{if (}E\texttt{)}\ C']\!](s,s')$

From the rule hypotheses, we know

(2) $E \overset{\mathrm{D}}{\simeq} E_D$
(3) $\mathrm{CHECK}(C) = C'$

From the definitions of $[\![\_]\!]_\perp$, `check`, and $[\![\_]\!]_\perp$, we know

(4)
$$
\begin{array}{l}
[\![\texttt{if (}E\texttt{) } C]\!]_\perp(s,s') \Leftrightarrow \\
\quad \text{LET } v = [\![E]\!]_\perp(s) \text{ IN} \\
\quad \text{IF } v = \perp \text{ THEN} \\
\quad\quad s' = \mathit{expthrow}(s) \\
\quad \text{ELSE IF } v = \text{TRUE THEN} \\
\quad\quad [\![C]\!]_\perp(s,s') \\
\quad \text{ELSE} \\
\quad\quad s' = s
\end{array}
$$

(5)
$$
\begin{array}{l}
[\![\text{CHECK } E_D\texttt{; if (}E\texttt{) } C']\!](s,s') \Leftrightarrow \\
\quad \text{IF } [\![E_D]\!](s) \neq \text{TRUE THEN} \\
\quad\quad s' = \mathit{throw}(s, \mathit{EXP}, \mathit{read}(s, \texttt{VAL})) \\
\quad \text{ELSE IF } [\![E]\!](s) = \text{TRUE THEN} \\
\quad\quad [\![C']\!](s,s') \\
\quad \text{ELSE} \\
\quad\quad s' = s
\end{array}
$$

From (2) and the definitions of $\overset{\mathrm{D}}{\simeq}$ and $[\![\_]\!]_\perp$, we know

(6) $[\![E_D]\!](s) \neq \text{TRUE} \Leftrightarrow [\![E]\!]_\perp(s) = \perp$

We proceed by case distinction:

- Case $[\![E]\!]_\perp(s) = \perp$: From the case condition, (4), (5), (6), and the definition of *expthrow*, we know (b).
- Case $[\![E]\!]_\perp(s) \neq \perp \wedge [\![E]\!]_\perp(s) = \text{TRUE}$: From this condition, Lemma "Partial Expressions", (1b), (3), (4), (5), (6), and the induction hypothesis, we know (b).
- Case $[\![E]\!]_\perp(s) \neq \perp \wedge [\![E]\!]_\perp(s) \neq \text{TRUE}$: From this condition, Lemma "Partial Expressions", (4), (5), and (6), we know (b).

- Case $C_0 = \texttt{if (}E\texttt{) } C_1 \texttt{ else } C_2$: we have to show

(b)
$$
\begin{array}{l}
[\![\texttt{if (}E\texttt{) } C_1 \texttt{ else } C_2]\!]_\perp(s,s') \Leftrightarrow \\
[\![\text{CHECK } E_D\texttt{; if (}E\texttt{) } C_1' \texttt{ else } C_2']\!](s,s')
\end{array}
$$

From the rule hypotheses, we know

(2) $E \overset{\mathrm{D}}{\simeq} E_D$

(3) $\text{CHECK}(C_1) = C_1'$
(4) $\text{CHECK}(C_2) = C_2'$

From the definitions of $[\![\_]\!]_\perp$, `check`, and $[\![\_]\!]_\perp$, we know

(5)
$$
\begin{array}{l}
[\![\texttt{if (}E\texttt{) } C_1 \texttt{ else } C_2]\!]_\perp(s,s') \Leftrightarrow \\
\quad \text{LET } v = [\![E]\!]_\perp(s) \text{ IN} \\
\quad \text{IF } v = \perp \text{ THEN} \\
\qquad s' = \mathit{expthrow}(s) \\
\quad \text{ELSE IF } v = \text{TRUE THEN} \\
\qquad [\![C_1]\!]_\perp(s,s') \\
\quad \text{ELSE} \\
\qquad [\![C_2]\!]_\perp(s,s')
\end{array}
$$

(6)
$$
\begin{array}{l}
[\![\text{CHECK } E_D\texttt{; if (}E\texttt{) } C_1' \texttt{ else } C_2']\!](s,s') \Leftrightarrow \\
\quad \text{IF } [\![E_D]\!](s) \neq \text{TRUE THEN} \\
\qquad s' = \mathit{throw}(s, \mathit{EXP}, \mathit{read}(s, \texttt{VAL})) \\
\quad \text{ELSE IF } [\![E]\!](s) = \text{TRUE THEN} \\
\qquad [\![C_1']\!](s,s') \\
\quad \text{ELSE} \\
\qquad [\![C_2']\!](s,s')
\end{array}
$$

From (2) and the definitions of $\overset{\text{D}}{\simeq}$ and $[\![\_]\!]_\perp$, we know

(7) $[\![E_D]\!](s) \neq \text{TRUE} \Leftrightarrow [\![E]\!]_\perp(s) = \perp$

We proceed by case distinction:

- Case $[\![E]\!]_\perp(s) = \perp$: From the case condition, (5), (6), (7), and the definition of *expthrow*, we know (b).
- Case $[\![E]\!]_\perp(s) \neq \perp \wedge [\![E]\!]_\perp(s) = \text{TRUE}$: From this condition, Lemma "Partial Expressions", (1b), (3), (5), (6), (7), and the induction hypothesis, we know (b).
- Case $[\![E]\!]_\perp(s) \neq \perp \wedge [\![E]\!]_\perp(s) \neq \text{TRUE}$: From this condition, Lemma "Partial Expressions", (1b), (4), (5), (6), (7), and the induction hypothesis, we know (b).

- Case $C_0 = \texttt{return } E$: we have to show

(b) $[\![\texttt{return } E]\!]_\perp(s,s') \Leftrightarrow [\![\text{CHECK } E_D\texttt{; return } E]\!](s,s')$

From the rule hypothesis, we know

(2) $E \overset{\text{D}}{\simeq} E_D$

From the definitions of $[\![\_]\!]_\perp$, `check`, and $[\![\_]\!]_\perp$, we know

$$
\begin{array}{ll}
 & [\![\texttt{return}\ E]\!]_\perp(s,s') \Leftrightarrow \\
 & \quad \text{LET } v = [\![E]\!]_\perp(s) \text{ IN} \\
(3) & \quad \text{IF } v = \perp \\
 & \qquad \text{THEN } s' = \mathit{expthrow}(s) \\
 & \qquad \text{ELSE } s' = \mathit{return}(s,v) \\
 & [\![\text{CHECK } E_D\texttt{;}\ \texttt{return}\ E]\!](s,s') \Leftrightarrow \\
 & \quad \text{IF } [\![E_D]\!](s) \neq \text{TRUE} \\
(4) & \qquad \text{THEN } s' = \mathit{throw}(s,\mathit{EXP},\mathit{read}(s,\texttt{VAL})) \\
 & \qquad \text{ELSE } s' = \mathit{return}(s,[\![E]\!](s))
\end{array}
$$

From (2) and the definitions of $\overset{\text{D}}{\simeq}$ and $[\![\_]\!]_\perp$, we know

(5) $[\![E_D]\!](s) \neq \text{TRUE} \Leftrightarrow [\![E]\!]_\perp(s) = \perp$

From (3), (4), (5), the definition of *expthrow*, and Lemma "Partial Expressions", we know (b).

- Case $C_0 = \texttt{throw}\ I\ E$: we have to show

(b) $[\![\texttt{throw}\ I\ E]\!]_\perp(s,s') \Leftrightarrow [\![\text{CHECK } E_D\texttt{;}\ \texttt{throw}\ I\ E]\!](s,s')$

From the rule hypothesis, we know

(2) $E \overset{\text{D}}{\simeq} E_D$

From the definitions of $[\![\_]\!]_\perp$, `check`, and $[\![\_]\!]_\perp$, we know

$$
\begin{array}{ll}
 & [\![\texttt{throw}\ I\ E]\!]_\perp(s,s') \Leftrightarrow \\
 & \quad \text{LET } v = [\![E]\!]_\perp(s) \text{ IN} \\
(3) & \quad \text{IF } v = \perp \\
 & \qquad \text{THEN } s' = \mathit{expthrow}(s) \\
 & \qquad \text{ELSE } s' = \mathit{throw}(s,I,v) \\
 & [\![\text{CHECK } E_D\texttt{;}\ \texttt{throw}\ I\ E]\!](s,s') \Leftrightarrow \\
 & \quad \text{IF } [\![E_D]\!](s) \neq \text{TRUE} \\
(4) & \qquad \text{THEN } s' = \mathit{throw}(s,\mathit{EXP},\mathit{read}(s,\texttt{VAL})) \\
 & \qquad \text{ELSE } s' = \mathit{throw}(s,I,[\![E]\!](s))
\end{array}
$$

From (2) and the definitions of $\overset{\text{D}}{\simeq}$ and $[\![\_]\!]_\perp$, we know

(5) $[\![E_D]\!](s) \neq \text{TRUE} \Leftrightarrow [\![E]\!]_\perp(s) = \perp$

From (3), (4), (5), the definition of *expthrow*, and Lemma "Partial Expressions", we know (b). □

Checked Program Calculus: Rules

$$\frac{E \overset{\mathrm{D}}{\simeq} E_D \quad \textsc{check}(C) = C' \quad [C']^{\texttt{FALSE},F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}}{\textsc{check}(\texttt{while (}E\texttt{) }C) = \textsc{check}\ E_D\texttt{; while (}E\texttt{) (}C'\texttt{; }\textsc{check}\ E_D\texttt{)}}$$

Figure 5.44: Checked Program Calculus (3/4)

5.9.5 Checking for Undefined Expressions in Loops

The checking of loops becomes a bit tricky due to the possibility of loop continuations triggered by the `continue` command. To simplify the further presentation, we first rule out this possibility and later deal with the general case.

Loops without `continue`

If the loop body cannot be left in a continuing state, it suffices to check the loop expression before the loop and after every iteration of the loop body. The rule in Figure 5.44 covers this situation.

To show the correctness of this rule, we have to show

(b) $\llbracket \texttt{while (}E\texttt{) }C \rrbracket_\perp(s,s') \Leftrightarrow \llbracket \textsc{check}\ E_D\texttt{; while (}E\texttt{) (}C'\texttt{; }\textsc{check}\ E_D\texttt{)} \rrbracket(s,s')$

From the rule hypotheses, we know

(2) $E \overset{\mathrm{D}}{\simeq} E_D$

(3) $\textsc{check}(C) = C'$

(4) $[C']^{\texttt{FALSE},F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n}$

From the definitions of $\llbracket \_ \rrbracket_\perp$, `check`, and $\llbracket \_ \rrbracket_\perp$, we know

$$
\begin{array}{l}
[\![\texttt{while (}E\texttt{) } C]\!]_\perp(s,s') \Leftrightarrow \\
\quad \exists k \in \mathbb{N}, t, u \in \mathit{State}^\infty : \\
\qquad \mathit{finiteExecution}_\perp(k,t,u,s,[\![E]\!]_\perp,[\![C]\!]_\perp) \wedge \\
\qquad ([\![E]\!]_\perp(t(k)) = \perp \vee \\
\qquad\quad [\![E]\!]_\perp(t(k)) \neq \text{TRUE} \vee \\
\qquad\quad \neg(\mathit{executes}(\mathit{control}(u(k))) \vee \\
\qquad\qquad \mathit{continues}(\mathit{control}(u(k))))) \wedge \\
\qquad \text{IF } (\mathit{executes}(\mathit{control}(u(k))) \vee \mathit{continues}(\mathit{control}(u(k)))) \wedge \\
\qquad\quad [\![E]\!]_\perp(t(k)) = \perp \\
\qquad\quad \text{THEN } \mathit{expthrow}(t(k)) = s' \\
\qquad\quad \text{ELSE } t(k) = s'
\end{array}
\tag{5}
$$

$$
\begin{array}{l}
[\![\text{CHECK } E_D\texttt{; while (}E\texttt{) (}C'\texttt{; }\text{CHECK } E_D\texttt{)}]\!](s,s') \Leftrightarrow \\
\quad \text{IF } [\![E_D]\!](s) \neq \text{TRUE THEN} \\
\qquad s' = \mathit{throw}(s, \mathit{EXP}, \mathit{read}(s, \texttt{VAL})) \\
\quad \text{ELSE} \\
\qquad \exists k \in \mathbb{N}, t, u \in \mathit{State}^\infty : \\
\qquad\quad \mathit{finiteExecution}(k,t,u,s,[\![E]\!],[\![C'\texttt{; }\text{CHECK } E_D]\!]) \wedge \\
\qquad\quad ([\![E]\!](t(k)) \neq \text{TRUE} \vee \\
\qquad\qquad \neg(\mathit{executes}(\mathit{control}(u(k))) \vee \\
\qquad\qquad\quad \mathit{continues}(\mathit{control}(u(k))))) \wedge \\
\qquad\quad t(k) = s'
\end{array}
\tag{6}
$$

We proceed by showing both directions of (b).

Proof of $\Rightarrow$ We assume

$$[\![\texttt{while (}E\texttt{) } C]\!]_\perp(s,s') \tag{7}$$

and show

$$[\![\text{CHECK } E_D\texttt{; while (}E\texttt{) (}C'\texttt{; }\text{CHECK } E_D\texttt{)}]\!](s,s') \tag{c}$$

From (5) and (7), we know for some $k \in \mathbb{N}, t, u \in \mathit{State}^\infty$

$$\mathit{finiteExecution}_\perp(k,t,u,s,[\![E]\!]_\perp,[\![C]\!]_\perp) \tag{8}$$

$$
\begin{array}{l}
[\![E]\!]_\perp(t(k)) = \perp \vee \\
[\![E]\!]_\perp(t(k)) \neq \text{TRUE} \vee \\
\neg(\mathit{executes}(\mathit{control}(u(k))) \vee \mathit{continues}(\mathit{control}(u(k))))
\end{array}
\tag{9}
$$

$$
\begin{array}{l}
\text{IF } (\mathit{executes}(\mathit{control}(u(k))) \vee \mathit{continues}(\mathit{control}(u(k)))) \wedge \\
\qquad [\![E]\!]_\perp(t(k)) = \perp \\
\text{THEN } \mathit{expthrow}(t(k)) = s' \\
\text{ELSE } t(k) = s'
\end{array}
\tag{10}
$$

From (6), to show (c), it suffices to show

$$
\begin{array}{ll}
 & \text{IF } [\![E_D]\!](s) \neq \text{TRUE THEN} \\
 & \quad s' = \mathit{throw}(s, \mathit{EXP}, \mathit{read}(s, \texttt{VAL})) \\
 & \text{ELSE} \\
 & \quad \exists k \in \mathbb{N}, t, u \in \mathit{State}^\infty : \\
\text{(d)} & \qquad \mathit{finiteExecution}(k, t, u, s, [\![E]\!], [\![C' \text{; CHECK } E_D]\!]) \wedge \\
 & \qquad ([\![E]\!](t(k)) \neq \text{TRUE} \vee \\
 & \qquad\quad \neg(\mathit{executes}(\mathit{control}(u(k))) \vee \\
 & \qquad\qquad \mathit{continues}(\mathit{control}(u(k))))) \wedge \\
 & \qquad t(k) = s'
\end{array}
$$

We proceed by case distinction.

As for the first case, we assume $[\![E_D]\!](s) \neq \text{TRUE}$: from the case condition, to show (d), we may show

(e) $s' = \mathit{throw}(s, \mathit{EXP}, \mathit{read}(s, \texttt{VAL}))$

From (2), the case condition, and the definition of $[\![\_]\!]_\bot$, we know

(11) $[\![E]\!]_\bot(s) = \bot$

From (8), (11), and the definition of $\mathit{finiteExecution}_\bot$, we know

(12) $k = 0$

(13) $s = t(k)$

(14) $s = u(k)$

From (1b), (10), (12), (13), and (14), we know

(15) $\mathit{expthrow}(t(k)) = s'$

From (15) and the definition of *expthrow*, we know (e).

As for the second case, we assume $[\![E_D]\!](s) = \text{TRUE}$ and define $t_0 : \mathit{State}^\infty$ and $u_0 : \mathit{State}^\infty$ as

$$
\begin{array}{ll}
 & t_0(i) := \\
 & \quad \text{IF } (\mathit{executes}(\mathit{control}(u(i))) \vee \mathit{continues}(\mathit{control}(u(i)))) \wedge \\
 & \qquad\quad [\![E]\!]_\bot(t(i)) = \bot \\
 & \qquad \text{THEN } \mathit{expthrow}(t(i)) \\
 & \qquad \text{ELSE } t(i) \\
\text{(16)} & u_0(i) := \\
 & \quad \text{IF } (\mathit{executes}(\mathit{control}(u(i))) \vee \mathit{continues}(\mathit{control}(u(i)))) \wedge \\
 & \qquad\quad [\![E]\!]_\bot(t(i)) = \bot \\
 & \qquad \text{THEN } \mathit{expthrow}(u(i)) \\
 & \qquad \text{ELSE } u(i)
\end{array}
$$

From the case condition, to show (d), it suffices to show

(e.1) $\mathit{finiteExecution}(k, t_0, u_0, s, [\![E]\!], [\![C'\,;\ \text{CHECK}\ E_D]\!])$

(e.2) $[\![E]\!](t_0(k)) \neq \text{TRUE} \vee$
$\neg(\mathit{executes}(\mathit{control}(u_0(k))) \vee \mathit{continues}(\mathit{control}(u_0(k))))$

(e.3) $t_0(k) = s'$

From (2), the case condition, and the definition of $[\![\llcorner]\!]_\perp$, we know

(17) $[\![E]\!]_\perp(s) \neq \perp$

(18) $[\![E]\!]_\perp(s) = [\![E]\!](s)$

We proceed by case distinction.

In the case of $k = 0$, from (7), the definition of $\mathit{finiteExecution}_\perp$, we know

(19) $s = t(k)$

(20) $s = u(k)$

From (16), (17), (18), (19), (20), the case condition, and the definition of *finiteExecution*, we know (e.1).

From (9), (16), (17), (18), (19), and (20), we know (e.2).

From (10), (16), (17), and (19), we know (e.3).

In the case of $k > 0$, from (8) and the definition of $\mathit{finiteExecution}_\perp$, we know

(21) $t(0) = s$

(22) $u(0) = s$

(23)
$\forall i \in \mathbb{N}_k :$
$\quad \neg\mathit{breaks}(\mathit{control}(u(i))) \wedge \mathit{executes}(\mathit{control}(t(i))) \wedge$
$\quad [\![E]\!]_\perp(t(i)) \neq \perp \wedge [\![E]\!]_\perp(t(i)) = \text{TRUE} \wedge$
$\quad [\![C]\!]_\perp(t(i), u(i+1)) \wedge$
$\quad \text{IF}\ \mathit{continues}(\mathit{control}(u(i+1))) \vee \mathit{breaks}(\mathit{control}(u(i+1)))$
$\quad\quad \text{THEN}\ t(i+1) = \mathit{execute}(u(i+1))$
$\quad\quad \text{ELSE}\ t(i+1) = u(i+1)$

From (1a), (3), (4), (23), the soundness of the verification calculus with interruptions, and the induction hypothesis, we know

(24) $\neg\mathit{continues}(\mathit{control}(u(k)))$

From (9) and (24), we know

$$(25)\quad \begin{array}{l} [\![E]\!]_\perp(t(k)) = \perp \vee [\![E]\!]_\perp(t(k)) \neq \text{TRUE} \vee \\ \neg \mathit{executes}(\mathit{control}(u(k))) \end{array}$$

We first show (e.2) and (e.3). From (25), we have the following cases:

- Case $\neg \mathit{executes}(\mathit{control}(u(k)))$:

  From (16), (24) and the case condition, we know (e.2).

  From (10), (16), (24), and the case condition, we know (e.3).

- Case $\mathit{executes}(\mathit{control}(u(k)) \wedge [\![E]\!]_\perp(t(k)) = \perp$:

  From (10) and the case condition, we know

  $$(26)\quad \mathit{expthrow}(t(k)) = s'$$

  From (16) and the case condition, we know

  $$(27)\quad \begin{array}{l} t_0(k) = \mathit{expthrow}(t(k)) \\ u_0(k) = \mathit{expthrow}(u(k)) \end{array}$$

  From (27), the definition of *expthrow*, (CD1), and Lemma "Control State Predicates", we know

  $$(28)\quad \neg \mathit{executes}(\mathit{control}(u_0(k)))$$
  $$(29)\quad \neg \mathit{continues}(\mathit{control}(u_0(k)))$$

  From (28) and (29), we know (e.2).

  From (26) and (27), we know (e.3).

- Case $\mathit{executes}(\mathit{control}(u(k)) \wedge [\![E]\!]_\perp(t(k)) \neq \perp \wedge [\![E]\!]_\perp(t(k)) \neq \text{TRUE}$:

  From (16), the case condition, and the definition of $[\![\_]\!]_\perp$, we know (e.2).

  From (10), (16), and the case condition, we know (e.3).

It remains to show (e.1). From (23), we know

$$(30)\quad \forall i \in \mathbb{N}_k : [\![E]\!]_\perp(t(i)) \neq \perp$$
$$(31)\quad \forall i \in \mathbb{N}_k : [\![E]\!]_\perp(t(i)) = \text{TRUE}$$
$$(32)\quad \forall i \in \mathbb{N}_k : \mathit{executes}(\mathit{control}(t(i)))$$
$$(33)\quad \forall i \in \mathbb{N}_k : [\![C]\!]_\perp(t(i), u(i+1))$$

From (30), and Lemma "Partial Expressions", we know

(34) $\forall i \in \mathbb{N}_k : [\![E]\!]_\perp(t(i)) = [\![E]\!](t(i))$

From (31) and (34), we know

(35) $\forall i \in \mathbb{N}_k : [\![E]\!](t(i)) = \text{TRUE}$

From (4), (32), (33), and the induction hypothesis, we know

(36) $\forall i \in \mathbb{N}_k : [\![C']\!](t(i), u(i+1))$

From the definitions of CHECK, *expthrow* and $[\![\_]\!]$, we know

(37)
$$
\begin{array}{l}
\forall s, s' \in \mathit{State} : \\
\quad [\![C' \,;\, \text{CHECK } E_D]\!])(s, s') \Leftrightarrow \\
\quad \exists s_0 \in \mathit{State} : \\
\qquad [\![C']\!](s, s_0) \wedge \\
\qquad \text{IF } \mathit{executes}(\mathit{control}(s_0)) \wedge \neg [\![E_D]\!](s_0) \\
\qquad\quad \text{THEN } s' = \mathit{expthrow}(s_0) \\
\qquad\quad \text{ELSE } s' = s_0
\end{array}
$$

To show (e.1), from (37) and the definition of *finiteExecution*, it suffices to show

(e.1.a) $t_0(0) = s$

(e.1.b) $u_0(0) = s$

(e.1.c)
$$
\begin{array}{l}
\forall i \in \mathbb{N}_k : \\
\quad \neg \mathit{breaks}(\mathit{control}(u_0(i))) \wedge \mathit{executes}(\mathit{control}(t_0(i))) \wedge \\
\quad [\![E]\!](t_0(i)) = \text{TRUE} \wedge \\
\quad \exists s_0 \in \mathit{State} : \\
\qquad [\![C']\!](t_0(i), s_0) \wedge \\
\qquad \text{IF } \mathit{executes}(\mathit{control}(s_0)) \wedge [\![E_D]\!](s_0) \neq \text{TRUE} \\
\qquad\quad \text{THEN } u_0(i+1) = \mathit{expthrow}(s_0) \\
\qquad\quad \text{ELSE } u_0(i+1) = s_0 \wedge \\
\quad \text{IF } \mathit{continues}(\mathit{control}(u_0(i+1))) \vee \mathit{breaks}(\mathit{control}(u_0(i+1))) \\
\qquad \text{THEN } t_0(i+1) = \mathit{execute}(u_0(i+1)) \\
\qquad \text{ELSE } t_0(i+1) = u_0(i+1)
\end{array}
$$

From (16), (17), (21), and (22), we know (e.1.a) and (e.1.b).

To show (e.1.c), we take arbitrary $i \in \mathbb{N}_k$ and show

(e.1.c.1) $\neg \mathit{breaks}(\mathit{control}(u_0(i)))$

(e.1.c.2) $\mathit{executes}(\mathit{control}(t_0(i)))$

(e.1.c.3) $[\![E]\!](t_0(i)) = \text{TRUE}$

$\exists s_0 \in State :$
$[\![C']\!](t_0(i), s_0) \wedge$
(e.1.c.4) IF $executes(control(s_0)) \wedge [\![E_D]\!](s_0) \neq \text{TRUE}$
THEN $u_0(i+1) = expthrow(s_0)$
ELSE $u_0(i+1) = s_0$

IF $continues(control(u_0(i+1))) \vee breaks(control(u_0(i+1)))$
(e.1.c.5) THEN $t_0(i+1) = execute(u_0(i+1))$
ELSE $t_0(i+1) = u_0(i+1)$

From (16) and (30), we know

(38) $t_0(i) = t(i)$

(39) $u_0(i) = u(i)$

From (23) and (38), we know (e.1.c.1).

From (23) and (39), we know (e.1.c.2).

From (35) and (38), we know (e.1.c.3).

To show (e.1.c.4), it suffices to show

(e.1.c.4.a.1) $[\![C']\!](t_0(i), u(i+1))$

IF $executes(control(u(i+1))) \wedge [\![E_D]\!](u(i+1)) \neq \text{TRUE}$
(e.1.c.4.a.2) THEN $u_0(i+1) = expthrow(u(i+1))$
ELSE $u_0(i+1) = u(i+1)$

From (36) and (38), we know (e.1.c.4.a.1).

To show (e.1.c.4.a.2), we consider three cases:

- Case $executes(control(u(i+1))) \wedge [\![E_D]\!](u(i+1)) \neq \text{TRUE}$: we show

(e.1.c.4.a.2.a) $u_0(i+1) = expthrow(u(i+1))$

From the case condition and the definition of $[\![\_]\!]_\perp$, we know

(40) $[\![E]\!]_\perp(u(i+1)) = \perp$

From (23), the case, and Lemma "State Control Predicates", we know

(41) $t(i+1) = u(i+1)$

From (16), (40), (41), and the case condition, we know (e.1.c.4.a.2.a).

- Case $executes(control(u(i+1))) \wedge [\![E_D]\!](u(i+1)) = \text{TRUE}$: we show

(e.1.c.4.a.2.a) $u_0(i+1) = u(i+1)$

From the case condition and the definition of $[\![\_]\!]_\perp$, we know

(42) $[\![E]\!]_\perp(u(i+1)) \neq \perp$

From (23), the case, and Lemma "State Control Predicates", we know

(43) $t(i+1) = u(i+1)$

From (16), (42), (43), and the case condition, we know (e.1.c.4.a.2.a).

- Case $\neg executes(control(u(i+1)))$: it suffices to show

(e.1.c.4.a.2.a) $u_0(i+1) = u(i+1)$

From (1a), (3), (4), (32), (36), the soundness of the verification calculus with interruptions, and the induction hypothesis, we know

(44) $\neg continues(control(u(i+1)))$

From (16), (44), and the case condition, we know (e.1.c.4.a.2.a).

To show (e.1.c.5), by Lemma "State Control Predicates", it suffices to consider the following cases:

- Case $continues(control(u_0(i+1))) \vee breaks(control(u_0(i+1)))$: it suffices to show

(e.1.c.5.a) $t_0(i+1) = execute(u_0(i+1))$

From (1a), (3), (4), (32), (e.1.c.2), (e.1.c.4.a.1), the soundness of the verification calculus with interruptions, and the induction hypothesis, we know

(45) $\neg continues(control(u_0(i+1)))$

From (45) and the case condition, we know

(46) $breaks(control(u_0(i+1)))$

From (46) and Lemma "Control Predicates" we know

(47) $\neg executes(control(u_0(i+1)))$

From (16), (45), and (47), we know

(48) $u_0(i+1) = u(i+1)$

From (23), (46), and (48), we know

(49) $t(i+1) = execute(u_0(i+1))$

From (16), (45), (47), (48), and (49), we know (e.1.c.5.a).

- Case $returns(control(u_0(i+1))) \vee executes(control(u_0(i+1)))$: by Lemma "State Control Predicates", it suffices to show

(e.1.c.5.a) $t_0(i+1) = u_0(i+1)$

From (16), the case condition, Lemma "State Control Predicates", and the definition of *expthrow*, we know

(50) $t_0(i+1) = t(i+1)$
(51) $u_0(i+1) = u(i+1)$

From (23), (50), (51), the case condition, and Lemma "State Control Predicates", we know (e.1.c.5.a).

- Case $throws(u_0(i+1))$: it suffices to show

(e.1.c.5.a) $t_0(i+1) = u_0(i+1)$

We assume

(52) $t_0(i+1) \neq u_0(i+1)$

and show a contradiction. From (16) and (52), we know

(53) $t(i+1) \neq u(i+1)$

From (23) and (53), we know

(54) $continues(control(u(i+1))) \vee breaks(u(i+1))$

From (1a), (3), (4), (32), (36), the soundness of the verification calculus with interruptions, and the induction hypothesis, we know

(55) $\neg continues(control(u(i+1)))$

From (54) and (55), we know

(56) $breaks(u(i+1))$

But (16), (56), and Lemma "State Control Predicates" together contradict the case condition.

Proof of $\Leftarrow$ We assume

(7) $[\![\text{CHECK } E_D\texttt{;}\ \texttt{while (}E\texttt{)}\ (C'\texttt{;}\ \text{CHECK } E_D)]\!](s,s')$

and show

(c) $[\![\texttt{while (}E\texttt{)}\ C]\!]_\perp(s,s')$

From (6) and (7), we know

$$
\begin{array}{l}
\text{IF } [\![E_D]\!](s) \neq \text{TRUE THEN} \\
\quad s' = \mathit{throw}(s, \mathit{EXP}, \mathit{read}(s, \texttt{VAL})) \\
\text{ELSE} \\
\quad \exists k \in \mathbb{N}, t, u \in \mathit{State}^\infty : \\
\quad\quad \mathit{finiteExecution}(k,t,u,s,[\![E]\!],[\![C'\texttt{;}\ \text{CHECK } E_D]\!]) \wedge \\
\quad\quad ([\![E]\!](t(k)) \neq \text{TRUE} \vee \\
\quad\quad\quad \neg(\mathit{executes}(\mathit{control}(u(k))) \vee \\
\quad\quad\quad\quad \mathit{continues}(\mathit{control}(u(k))))) \wedge \\
\quad\quad t(k) = s'
\end{array}
\tag{8}
$$

From (5), to show (c), it suffices to show

$$
\begin{array}{l}
\exists k \in \mathbb{N}, t, u \in \mathit{State}^\infty : \\
\quad \mathit{finiteExecution}_\perp(k,t,u,s,[\![E]\!]_\perp,[\![C]\!]_\perp) \wedge \\
\quad ([\![E]\!]_\perp(t(k)) = \perp \vee \\
\quad\quad [\![E]\!]_\perp(t(k)) \neq \text{TRUE} \vee \\
\quad\quad \neg(\mathit{executes}(\mathit{control}(u(k))) \vee \\
\quad\quad\quad \mathit{continues}(\mathit{control}(u(k))))) \wedge \\
\quad \text{IF } (\mathit{executes}(\mathit{control}(u(k))) \vee \mathit{continues}(\mathit{control}(u(k)))) \wedge \\
\quad\quad [\![E]\!]_\perp(t(k)) = \perp \\
\quad\quad \text{THEN } \mathit{expthrow}(t(k)) = s' \\
\quad\quad \text{ELSE } t(k) = s'
\end{array}
\tag{d}
$$

We proceed by case distinction.

As for the first case, we assume $[\![E_D]\!](s) \neq \text{TRUE}$.

From the case condition and (8), we know

(9) $s' = \mathit{throw}(s, \mathit{EXP}, \mathit{read}(s, \texttt{VAL}))$

We define $t : \mathit{State}^\infty$ and $u : \mathit{State}^\infty$ as

$$
\begin{array}{l}
t(i) := \text{IF } i = 0 \text{ THEN } s \text{ ELSE } s' \\
u(i) := \text{IF } i = 0 \text{ THEN } s \text{ ELSE } s'
\end{array}
\tag{10}
$$

To show (d), it suffices to show

(d.1) $\mathit{finiteExecution}_{\perp}(0,t,u,s,[\![E]\!]_{\perp},[\![C]\!]_{\perp})$

(d.2) $[\![E]\!]_{\perp}(t(0)) = \perp \vee$
$[\![E]\!]_{\perp}(t(0)) \neq \text{TRUE} \vee$
$\neg(\mathit{executes}(\mathit{control}(u(0))) \vee \mathit{continues}(\mathit{control}(u(0))))$

(d.3) IF $(\mathit{executes}(\mathit{control}(u(0))) \vee \mathit{continues}(\mathit{control}(u(0)))) \wedge$
$[\![E]\!]_{\perp}(t(0)) = \perp$
THEN $\mathit{expthrow}(t(0)) = s'$
ELSE $t(0) = s'$

From (10) and the definition of *finiteExecution*, we know (d.1).

From (2), the case condition, and the definition of $[\![\,\_\,]\!]_{\perp}$, we know

(11) $[\![E]\!]_{\perp}(s) = \perp$

From (10) and (11), we know

(12) $[\![E]\!]_{\perp}(t(0)) = \perp$

From (12), we know (d.2).

From (1b), (9), (10), (12), and the definition of *expthrow*, we know (d.3).

As for the second case, we assume $[\![E_D]\!](s) = \text{TRUE}$.

From (2), the case condition and the definition of $[\![\,\_\,]\!]_{\perp}$, we know

(13) $[\![E]\!]_{\perp}(s) \neq \perp$

(14) $[\![E]\!]_{\perp}(s) = [\![E]\!](s)$

From the case condition and (8), we know for some $k \in \mathbb{N}, t, u \in \mathit{State}^{\infty}$:

(15) $\mathit{finiteExecution}(k,t,u,s,[\![E]\!],[\![C'\,;\ \text{CHECK}\ E_D]\!])$

(16) $[\![E]\!](t(k)) \neq \text{TRUE} \vee$
$\neg(\mathit{executes}(\mathit{control}(u(k))) \vee \mathit{continues}(\mathit{control}(u(k))))$

(17) $t(k) = s'$

From (15) and the definition of *finiteExecution*, we know

(18) $t(0) = s$

(19) $u(0) = s$

$$
\begin{array}{l}
\forall i \in \mathbb{N}_k : \\
\quad \neg \mathit{breaks}(\mathit{control}(u(i))) \wedge \mathit{executes}(\mathit{control}(t(i))) \wedge \\
\quad [\![E]\!](t(i)) = \text{TRUE} \wedge [\![C' \text{; CHECK } E_D]\!](t(i), u(i+1)) \wedge \\
\quad \text{IF } \mathit{continues}(\mathit{control}(u(i+1))) \vee \mathit{breaks}(\mathit{control}(u(i+1))) \\
\qquad \text{THEN } t(i+1) = \mathit{execute}(u(i+1)) \\
\qquad \text{ELSE } t(i+1) = u(i+1)
\end{array}
\tag{20}
$$

From (20) and the definition of $[\![\_]\!]$, we know

$$
\begin{array}{l}
\forall i \in \mathbb{N}_k : \exists s_0 \in \mathit{State} : \\
\quad [\![C']\!](t(i), s_0) \wedge \\
\quad \text{IF } \mathit{executes}(\mathit{control}(s_0)) \wedge [\![E_D]\!](s_0) \neq \text{TRUE} \\
\qquad \text{THEN } u(i+1) = \mathit{expthrow}(s_0) \\
\qquad \text{ELSE } u(i+1) = s_0
\end{array}
\tag{21}
$$

We define $u_0 : \mathit{State}^\infty$ as

$$
\begin{array}{l}
u_0(i) := \\
\quad \text{IF } i = 0 \text{ THEN} \\
\qquad s \\
\quad \text{ELSE SUCH } s_0 \in \mathit{State} : \\
\qquad [\![C']\!](t(i-1), s_0) \wedge \\
\qquad \text{IF } \mathit{executes}(\mathit{control}(s_0)) \wedge [\![E_D]\!](s_0) \neq \text{TRUE} \\
\qquad\quad \text{THEN } u(i) = \mathit{expthrow}(s_0) \\
\qquad\quad \text{ELSE } u(i) = s_0
\end{array}
\tag{22}
$$

From (21) and (22), we know

$$
\begin{array}{l}
\forall i \in \mathbb{N}_k : \\
\quad [\![C']\!](t(i), u_0(i+1)) \wedge \\
\quad \text{IF } \mathit{executes}(\mathit{control}(u_0(i+1))) \wedge [\![E_D]\!](u_0(i+1)) \neq \text{TRUE} \\
\qquad \text{THEN } u(i+1) = \mathit{expthrow}(u_0(i+1)) \\
\qquad \text{ELSE } u(i+1) = u_0(i+1)
\end{array}
\tag{23}
$$

We also define $t_0 : \mathit{State}^\infty$ as

$$
\begin{array}{l}
t_0(i) := \\
\quad \text{IF } i = 0 \text{ THEN} \\
\qquad s \\
\quad \text{ELSE} \\
\qquad \text{IF } \mathit{continues}(\mathit{control}(u_0(i))) \vee \mathit{breaks}(\mathit{control}(u_0(i))) \\
\qquad\quad \text{THEN } \mathit{execute}(u_0(i)) \\
\qquad\quad \text{ELSE } u_0(i)
\end{array}
\tag{24}
$$

From (24), we know

$$
\begin{array}{ll}
(25) & \forall i \in \mathbb{N}_k : \\
& \quad \text{IF } \mathit{continues}(\mathit{control}(u_0(i+1))) \vee \mathit{breaks}(\mathit{control}(u_0(i+1))) \\
& \qquad \text{THEN } t_0(i+1) = \mathit{execute}(u_0(i+1)) \\
& \qquad \text{ELSE } t_0(i+1) = u_0(i+1)
\end{array}
$$

From (18), (19), (20), (21), (22), (23), (24), and (25), we know

$$
\begin{array}{ll}
(26) & u(0) = u_0(0) \\
(27) & t(0) = t_0(0) \\
(28) & \forall i \in \mathbb{N}_k : u(i) = u_0(i) \\
(29) & \forall i \in \mathbb{N}_k : t(i) = t_0(i) \\
(30) & k > 0 \ \Rightarrow \\
& \quad \text{IF } \mathit{executes}(\mathit{control}(u_0(k))) \wedge [\![E_D]\!](u_0(k)) \neq \text{TRUE} \\
& \qquad \text{THEN } u(k) = \mathit{expthrow}(u_0(k)) \\
& \qquad \text{ELSE } u(k) = u_0(k) \\
(31) & k > 0 \Rightarrow \\
& \quad \text{IF } \mathit{continues}(\mathit{control}(u_0(k))) \vee \mathit{breaks}(\mathit{control}(u_0(k))) \\
& \qquad \text{THEN } t_0(k) = \mathit{execute}(u_0(k)) \\
& \qquad \text{ELSE } t_0(k) = u_0(k) \\
(32) & k > 0 \Rightarrow \\
& \quad \text{IF } \mathit{continues}(\mathit{control}(u(k))) \vee \mathit{breaks}(\mathit{control}(u(k))) \\
& \qquad \text{THEN } t(k) = \mathit{execute}(u(k)) \\
& \qquad \text{ELSE } t(k) = u(k)
\end{array}
$$

To show (d), it now suffices to show

$$
\begin{array}{ll}
\text{(d.1)} & \mathit{finiteExecution}_\perp(k, t_0, u_0, s, [\![E]\!]_\perp, [\![C]\!]_\perp) \\
\text{(d.2)} & [\![E]\!]_\perp(t_0(k)) = \perp \vee \\
& [\![E]\!]_\perp(t_0(k)) \neq \text{TRUE} \vee \\
& \neg(\mathit{executes}(\mathit{control}(u_0(k))) \vee \mathit{continues}(\mathit{control}(u_0(k)))) \\
\text{(d.3)} & \text{IF } (\mathit{executes}(\mathit{control}(u_0(k))) \vee \mathit{continues}(\mathit{control}(u_0(k)))) \wedge \\
& \qquad [\![E]\!]_\perp(t_0(k)) = \perp \\
& \quad \text{THEN } \mathit{expthrow}(t_0(k)) = s' \\
& \quad \text{ELSE } t_0(k) = s'
\end{array}
$$

We proceed by case distinction.

In the first case, we assume

(33) $k = 0$

From (18), (19), (26), (27), and the definition of *finiteExecution*$_\perp$, we have (d.1).

From (13), (16), (18), (19), (26), (27), and (33), we have (d.2).

From (13), (18), (27), and (33), we have (d.3).

In the second case, we assume

(34) $k > 0$

To show (d.2), it suffices to assume

(35) $[\![E]\!]_\perp(t_0(k)) \neq \perp$

(36) $[\![E]\!]_\perp(t_0(k)) = \text{TRUE}$

(37) $executes(control(u_0(k))) \vee continues(control(u_0(k)))$

and show a contradiction.

From (4), (23), (34), (37), and the soundness of the verif. calculus, we know

(38) $executes(control(u_0(k)))$

From (31), (34), and (38), we know

(39) $t_0(k) = u_0(k)$

From (2), (35) and the definition of $[\![\_]\!]_\perp$, we know

(40) $[\![E_D]\!](t_0(k)) = \text{TRUE}$

From (30), (34), (38), (39), and (40), we know

(41) $u(k) = u_0(k)$

From (32), (34), (38), and (41), we know

(42) $t(k) = u(k)$

From (16), (36), (39), (41), (42), we know

(43) $\neg(executes(control(u(k)) \vee continues(control(u(k)))))$

From (37), (41), and (43), we have a contradiction.

To show (d.3), we proceed by case distinction.

- In the first case, we assume

  (44) $executes(control(u_0(k))) \vee continues(control(u_0(k)))$
  (45) $[\![E]\!]_\perp(t_0(k)) = \perp$

  and show

  (d.3.a) $expthrow(t_0(k)) = s'$

  From (4), (23), (34), (44), and the soundn. of the verif. calculus, we know

  (46) $executes(control(u_0(k)))$

  From (2), (45), and the definition of $[\![\_]\!]_\perp$, we know

  (47) $[\![E_D]\!](t_0(k)) \neq \text{TRUE}$

  From (31), (34), and (46), we know

  (48) $t_0(k) = u_0(k)$

  From (30), (46), (47), and (48), we know

  (49) $u(k) = expthrow(u_0(k))$

  From (32), (34), (49), and the definition of *expthrow*, we know

  (50) $t(k) = u(k)$

  From (17), (48), (49), and (50), we know (d.3.a).

- In the second case, we assume

  (51) $executes(control(u_0(k))) \vee continues(control(u_0(k)))$
  (52) $[\![E]\!]_\perp(t_0(k)) \neq \perp$

  and show

  (d.3.a) $t_0(k) = s'$

  From (4), (23), (34), (51), and the soundn. of the verif. calculus, we know

  (53) $executes(control(u_0(k)))$

  From (2), (52), and the definition of $[\![\_]\!]_\perp$, we know

  (54) $[\![E_D]\!](t_0(k)) = \text{TRUE}$

  From (31), (34), and (53), we know

  (55) $t_0(k) = u_0(k)$

  From (30), (54), and (44), we know

(56) $u(k) = u_0(k)$

From (32), (34), (53), (56), and Lemma "State Control Predicates", we know

(57) $t(k) = u(k)$

From (17), (55), (56), and (57), we know (d.3.a).

- In the third case, we assume

(59) $\neg executes(control(u_0(k)))$
(60) $\neg continues(control(u_0(k)))$

and show

(d.3.a) $t_0(k) = s'$

From (30), (34), and (59), we know

(61) $u(k) = u_0(k)$

First, we consider the case

(62) $breaks(control(u_0(k)))$

From (31), (34), and (62), we know

(63) $t_0(k) = execute(u_0(k))$

From (32), (34), (61), and (62), we know

(64) $t(k) = execute(u(k))$

From (61), (63), and (64), we know

(65) $t(k) = t_0(k)$

From (17) and (65), we know (d.3.a).

Now, we consider the case

(66) $\neg breaks(control(u_0(k)))$

From (31), (34), (60), and (65), we know

(67) $t_0(k) = u_0(k)$

From (32), (34), (60), (61), and (66), we know

(68) $t(k) = u(k)$

From (61), (67), and (68), we know

(69) $t(k) = t_0(k)$

From (17) and (69), we know (d.3.a).

It remains to show (d.1). By the definition of $\mathit{finiteExecution}_\perp$, we have to show

(d.1.a.1) $t_0(0) = s$

(d.1.a.2) $u_0(0) = s$

(d.2.a.3)
$$\begin{array}{l}\forall i \in \mathbb{N}_k : \\ \quad \neg \mathit{breaks}(\mathit{control}(u_0(i))) \wedge \mathit{executes}(\mathit{control}(t_0(i))) \wedge \\ \quad [\![E]\!]_\perp(t_0(i)) \neq \perp \wedge [\![E]\!]_\perp(t_0(i)) = \text{TRUE} \wedge [\![C]\!]_\perp(t_0(i), u_0(i+1)) \wedge \\ \quad \text{IF } \mathit{continues}(\mathit{control}(u_0(i+1))) \vee \mathit{breaks}(\mathit{control}(u_0(i+1))) \\ \qquad \text{THEN } t_0(i+1) = \mathit{execute}(u_0(i+1)) \\ \qquad \text{ELSE } t_0(i+1) = u_0(i+1)\end{array}$$

From (18), (19), (26), and (27), we know (d.1.a.1) and (d.1.a.2).

To show (d.2.a.3), we take arbitrary $i \in \mathbb{N}_k$ and show

(d.2.a.3.a.1) $\neg \mathit{breaks}(\mathit{control}(u_0(i)))$

(d.2.a.3.a.2) $\mathit{executes}(\mathit{control}(t_0(i)))$

(d.2.a.3.a.3) $[\![E]\!]_\perp(t_0(i)) \neq \perp$

(d.2.a.3.a.4) $[\![E]\!]_\perp(t_0(i)) = \text{TRUE}$

(d.2.a.3.a.5) $[\![C]\!]_\perp(t_0(i), u_0(i+1))$

(d.2.a.3.a.6)
$$\begin{array}{l}\text{IF } \mathit{continues}(\mathit{control}(u_0(i+1))) \vee \mathit{breaks}(\mathit{control}(u_0(i+1))) \\ \quad \text{THEN } t_0(i+1) = \mathit{execute}(u_0(i+1)) \\ \quad \text{ELSE } t_0(i+1) = u_0(i+1)\end{array}$$

From (20), (26), and (27), we know (d.2.a.3.a.1) and (d.2.a.3.a.2).

From (20), we know

(70) $[\![E]\!]_\perp(t(i)) = \text{TRUE}$

From (29) and (70), we know (d.2.a.3.a.3) and (d.2.a.3.a.4).

From (3), (23), (29), (d.2.a.3.a.2), and the ind. hypothesis., we know (d.2.a.3.a.5).

From (25), we know (d.2.a.3.a.6). □

Loops with `continue`

If the loop body contains instances of the command `continue`, the body may be left in a "continuing" state which bypasses the `check` after the body but directly proceeds with the evaluation of the loop expression. The only way to ensure that the loop expression is also well-defined in such "continuing" states is to precede every invocation of `continue` itself by a checking command which yields

```
// transformation of continue
{ check E_D; continue }
```

where E_D denotes the definedness condition of the enclosing loop.

However there are two caveats:

- First, we must make sure that the evaluation of E_D in the current state gives the same result as the evaluation in the state where the loop expression is evaluated. This is not necessarily the case, if the `continue` occurs inside the local declaration of a variable that has the same name as a variable on which the value of E_D depends. Thus we have to make sure that a local variable definition does not capture a variable referenced in E_D.
- Second, we must make sure that the exception raised by the `check` command is not caught inside the loop body; the easiest way to guarantee this is to prohibit subcommands of the form `try` C_1 `catch` (*EXP I*) C_2 in the loop body. This is the core reason for the condition (1a) in the proof of Lemma "Checked Commands".

For performing the transformation, the calculus is modified as sketched in Figure 5.45: the judgement $\text{CHECK}(C) = C'$ is generalized to a form $\text{CHECK}_{E'}(C) = C'$ which translates command C to its checked counterpart C' under the assumption that C occurs (directly) in the body of a loop with expression E'. The rules given in Figure 5.43 have to be correspondingly modified to "forward" the additional argument E from the conclusion to the premises (not shown). The rules for local variable declarations and definitions receive extra premises to avoid the problem of captured variables (in practice, programs can be automatically transformed to satisfy this premise by renaming local variables correspondingly).

Furthermore, two new rules are added: the first one describes the transformation of the command `continue` to include a check for the well-definedness of E; the second one generalizes the rule for while loops shown in Figure 5.44 by dropping the constraint that the loop body C must not give rise to a continuing state but

Checked Program Calculus: Judgements

$$
\begin{array}{l}
\text{CHECK}_{E'}(C) = C' \;\Leftrightarrow \\
\quad \forall s, s' \in \mathit{State} : \mathit{executes}(\mathit{control}(s)) \Rightarrow \\
\qquad (\llbracket C \rrbracket_\perp(s,s') \wedge \neg \llbracket C' \rrbracket(s,s') \;\Rightarrow \\
\qquad\quad \mathit{continues}(\mathit{control}(s')) \wedge \llbracket E' \rrbracket_\perp(s') = \perp \wedge \\
\qquad\quad \llbracket C' \rrbracket(s, \mathit{expthrow}(s')) \wedge \\
\qquad (\llbracket C' \rrbracket(s,s') \wedge \neg \llbracket C \rrbracket_\perp(s,s') \;\Rightarrow \\
\qquad\quad \exists s'' \in \mathit{State} : \llbracket C \rrbracket_\perp(s,s'') \wedge s' = \mathit{expthrow}(s'') \wedge \\
\qquad\qquad \mathit{continues}(\mathit{control}(s'')) \wedge \llbracket E' \rrbracket_\perp(s'') = \perp) \wedge \\
\qquad (\llbracket C' \rrbracket(s,s') \;\Rightarrow \\
\qquad\quad \neg(\mathit{continues}(\mathit{control}(s')) \wedge \llbracket E' \rrbracket_\perp(s') = \perp))
\end{array}
$$

Checked Program Calculus: Rules

$$
\frac{\begin{array}{l}\forall s_1, s_2 \in \mathit{State} : s_1 = s_2 \text{ EXCEPT } I \Rightarrow \llbracket E' \rrbracket_\perp(s_1) = \llbracket E' \rrbracket_\perp(s_2) \\ \text{CHECK}_{E'}(C) = C'\end{array}}{\text{CHECK}_{E'}(\texttt{var } I\texttt{; } C) = \texttt{var } I\texttt{; } C'}
$$

$$
\frac{\begin{array}{l}\forall s_1, s_2 \in \mathit{State} : s_1 = s_2 \text{ EXCEPT } I \Rightarrow \llbracket E' \rrbracket_\perp(s_1) = \llbracket E' \rrbracket_\perp(s_2) \\ E \overset{\text{D}}{\simeq} E_D \\ \text{CHECK}_{E'}(C) = C'\end{array}}{\text{CHECK}_{E'}(\texttt{var } I\texttt{=}E\texttt{; } C) = \texttt{check } E_D\texttt{; var } I\texttt{=}E\texttt{; } C'}
$$

$$
\frac{E' \overset{\text{D}}{\simeq} E_D}{\text{CHECK}_{E'}(\texttt{continue}) = \texttt{check } E_D\texttt{; continue}}
$$

$$
\frac{\begin{array}{l}E \overset{\text{D}}{\simeq} E_D \\ \text{CHECK}_E(C) = C'\end{array}}{\begin{array}{l}\text{CHECK}_{E'}(\texttt{while (}E\texttt{) } C) = \\ \quad \text{CHECK } E_D\texttt{; while (}E\texttt{) (}C'\texttt{; CHECK } E_D\texttt{)}\end{array}}
$$

Figure 5.45: Checked Program Calculus (4/4)

generalizing the checking transformation of loop body C to take into account the loop expression E.

The soundness claim depicted in Figure 5.45 is now not any more one of an unconditional equivalence of the original program and its checked version: the original program may result in a state which cannot be reached by the execution of the transformed program, provided that this is a continuing state in which the loop expression is undefined and that the transformed program generates an "expression evaluation exception" on that state; correspondingly, the transformed program may result in a state which cannot be reached by the execution of the original program provided that this state is derived from a continuing state with undefined loop expression reachable from the original program by raising an"expression evaluation exception"; in any case, the transformed program cannot yield a "continuing state" where the loop expression is undefined.

Lemma (Checked Commands Extended) If a judgement $\text{CHECK}_{E'}(C) = C'$ can be derived from the rules of the checked program calculus, then we have

$$
\begin{array}{l}
\mathit{DifferentVariables} \wedge \\
C \text{ has no subcommand } \texttt{try}\ C_1\ \texttt{catch}\,(\mathit{EXP}\ I)\ C_2 \ \Rightarrow \\
\quad \forall s, s' \in \mathit{State} : \mathit{executes}(\mathit{control}(s)) \Rightarrow \\
\qquad ([\![C]\!]_\perp(s,s') \wedge \neg [\![C']\!](s,s') \ \Rightarrow \\
\qquad\quad \mathit{continues}(\mathit{control}(s')) \wedge [\![E']\!]_\perp(s') = \perp \wedge \\
\qquad\quad [\![C']\!](s, \mathit{expthrow}(s'))) \wedge \\
\qquad ([\![C']\!](s,s') \wedge \neg [\![C]\!]_\perp(s,s') \ \Rightarrow \\
\qquad\quad \exists s'' \in \mathit{State} : [\![C]\!]_\perp(s,s'') \wedge s' = \mathit{expthrow}(s'') \wedge \\
\qquad\qquad \mathit{continues}(\mathit{control}(s'')) \wedge [\![E']\!]_\perp(s'') = \perp) \wedge \\
\qquad ([\![C']\!](s,s') \ \Rightarrow \\
\qquad\quad \neg(\mathit{continues}(\mathit{control}(s')) \wedge [\![E']\!]_\perp(s') = \perp))
\end{array}
$$

Proof Take arbitrary commands C_0 and C_0' and expression E' such that judgement $\text{CHECK}_{E'}(C_0) = C_0'$ can be derived. Take $s, s' \in \mathit{State}$ and assume

(1a) C_0 has no subcommand $\texttt{try}\ C_1\ \texttt{catch}\,(\mathit{EXP}\ I)\ C_2$

(1b) $\mathit{executes}(\mathit{control}(s))$

(1c) $\mathit{DifferentVariables}$

We show

$$
\begin{array}{ll}
 & [\![C_0]\!]_\perp(s,s') \wedge \neg [\![C_0']\!](s,s') \ \Rightarrow \\
\text{(a.1)} & \quad \mathit{continues}(\mathit{control}(s')) \wedge [\![E']\!]_\perp(s') = \perp \wedge \\
 & \quad [\![C_0']\!](s, \mathit{expthrow}(s'))
\end{array}
$$

$$\begin{array}{ll} & [\![C_0']\!](s,s') \wedge \neg [\![C_0]\!]_\perp(s,s') \Rightarrow \\ \text{(a.2)} & \quad \exists s'' \in \mathit{State} : [\![C_0]\!]_\perp(s,s'') \wedge s' = \mathit{expthrow}(s'') \wedge \\ & \quad\quad \mathit{continues}(\mathit{control}(s'')) \wedge [\![E']\!]_\perp(s'') = \perp \end{array}$$

$$\begin{array}{ll} & [\![C_0']\!](s,s') \Rightarrow \\ \text{(a.3)} & \quad \neg(\mathit{continues}(\mathit{control}(s')) \wedge [\![E']\!]_\perp(s') = \perp) \end{array}$$

We proceed by induction on the structure of C_0. but do not repeat the proofs for the commands previously presented in this section. Instead, we focus on the rules introduced in this section.

- Case $C_0 = \texttt{var}\ I\texttt{;}\ C$: from the premises of the rule, we know

  (2) $\forall s_1, s_2 \in \mathit{State} : s_1 = s_2 \text{ EXCEPT } I \Rightarrow [\![E']\!]_\perp(s_1) = [\![E']\!]_\perp(s_2)$
  (3) $\text{CHECK}_{E'}(C) = C'$

  To show (a.1), we assume

  (4) $[\![\texttt{var}\ I\texttt{;}\ C]\!]_\perp(s,s')$
  (5) $\neg[\![\texttt{var}\ I\texttt{;}\ C']\!](s,s')$

  and show

  (a.1.a.1) $\mathit{continues}(\mathit{control}(s'))$
  (a.1.a.2) $[\![E']\!]_\perp(s') = \perp$
  (a.1.a.3) $[\![\texttt{var}\ I\texttt{;}\ C']\!](s, \mathit{expthrow}(s'))$

  From (4) and the definition of $[\![\_]\!]_\perp$, we know for some $s_0, s_1 \in \mathit{State}$

  (6) $s_0 = s \text{ EXCEPT } I$
  (7) $[\![C]\!]_\perp(s_0,s_1)$
  (8) $s' = \mathit{write}(s_1, I, \mathit{read}(s,I))$

  From (5) and the definition of $[\![\_]\!]$, we know

  (9) $\neg(s_0 = s \text{ EXCEPT } I) \vee$
  $\neg[\![C']\!](s_0,s_1) \vee$
  $\neg(s' = \mathit{write}(s_1, I, \mathit{read}(s,I)))$

  From (6), (8), and (9), we know

  (10) $\neg[\![C']\!](s_0,s_1)$

  From (1a), (1b), (1c) (3), (7), (10), and the induction hypothesis, we know

  (11) $\mathit{continues}(\mathit{control}(s_1))$
  (12) $[\![E']\!]_\perp(s_1) = \perp$
  (13) $[\![C']\!](s_0, \mathit{expthrow}(s_1))$

From (8), (11), and (CW), we know (a.1.a.1).

From (1c), (2), (8), (12), and (WS), we know (a.1.a.2).

To show (a.1.a.3), from (6), (13), and the definition of $[\![\_]\!]$, it remains to show suffices to show

(a.1.a.3.a) $\mathit{expthrow}(s') = \mathit{write}(\mathit{expthrow}(s_1), I, \mathit{read}(s, I))$

From (8), (CW), and the definition of *expthrow*, we know (a.1.a.3.a.).

To show (a.2), we assume

(4) $[\![\texttt{var}\ I\texttt{;}\ C']\!](s, s')$
(5) $\neg [\![\texttt{var}\ I\texttt{;}\ C]\!]_\perp(s, s')$

and show

(a.1.a) $\exists s'' \in \mathit{State} : [\![\texttt{var}\ I\texttt{;}\ C]\!]_\perp(s, s'') \wedge s' = \mathit{expthrow}(s'') \wedge \mathit{continues}(\mathit{control}(s'')) \wedge [\![E']\!]_\perp(s'') = \perp$

From (4) and the definition of $[\![\_]\!]$, we know for some $s_0, s_1 \in \mathit{State}$

(6) $s_0 = s$ EXCEPT I
(7) $[\![C']\!](s_0, s_1)$
(8) $s' = \mathit{write}(s_1, I, \mathit{read}(s, I))$

From (5) and the definition of $[\![\_]\!]_\perp$, we know

(9) $\neg(s_0 = s \text{ EXCEPT } I) \vee \neg [\![C]\!]_\perp(s_0, s_1) \vee \neg(s' = \mathit{write}(s_1, I, \mathit{read}(s, I)))$

From (6), (8), and (9), we know

(10) $\neg [\![C]\!]_\perp(s_0, s_1)$

From (1a), (1b), (1c) (3), (7), (10), and the induction hypothesis, we know for some $s'' \in \mathit{State}$

(11) $[\![C]\!]_\perp(s_0, s'')$
(12) $s_1 = \mathit{expthrow}(s'')$
(13) $\mathit{continues}(\mathit{control}(s''))$
(14) $[\![E']\!]_\perp(s'') = \perp$

We define

(15) $s''' := \mathit{write}(s'', I, \mathit{read}(s, I))$

To show (a.1.a), it suffices to show

(a.1.b.1) $[\![\texttt{var}\ I\texttt{;}\ C]\!]_\perp(s, s''')$

(a.1.b.2) $s' = expthrow(s''')$
(a.1.b.3) $continues(control(s'''))$
(a.1.b.4) $[\![E']\!]_\perp(s''') = \perp$

From (6), (11), (15), and the definition of $[\![\_]\!]_\perp$, we know (a.1.b.1).

To show (a.1.b.2), from (8), (12), and (15), it suffices to show

(a.1.b.2.a) $write(expthrow(s''), I, read(s,I)) = expthrow(write(s'', I, read(s,I)))$

From (CW), (CD1), and the definition of *expthrow*, we know (a.1.b.2.a).

From (13), (15), and (CW), we know (a.1.b.3).

From (1c), (2), (14), (15), and (WS), we know (a.1.b.4).

To show (a.3), we assume

(4) $[\![\texttt{var}\ I\texttt{;}\ C']\!](s,s')$
(5) $continues(control(s'))$

and show

(a.3.a) $[\![E']\!]_\perp(s') \neq \perp$

From (4) and the definition of $[\![\_]\!]$, we know for some $s_0, s_1 \in State$

(6) $s_0 = s$ EXCEPT I
(7) $[\![C']\!](s_0, s_1)$
(8) $s' = write(s_1, I, read(s,I))$

From (1a), (1b), (1c) (3), (7), and the induction hypothesis, we know

(11) $\neg continues(control(s_1)) \vee [\![E']\!]_\perp(s_1) \neq \perp$

From (5), (8), (11), and (CW), we know

(12) $[\![E']\!]_\perp(s_1) \neq \perp$

From (1c), (2), (8), (12), and (WS), we know (a.3.a).

- Case $C_0 = \texttt{var}\ I\texttt{=}E\texttt{;}\ C$: analogous to the previous case.

- Case $C_0 = \texttt{continue}$: from the premise of the rule, we know

(2) $E' \overset{\mathrm{D}}{\simeq} E_D$

To show (a.1), we assume

(3) $[\![\texttt{continue}]\!]_\perp(s,s')$
(4) $\neg[\![\texttt{check}\ E_D\texttt{;}\ \texttt{continue}]\!](s,s')$

and show

(a.1.a.1) $\mathit{continues}(\mathit{control}(s'))$
(a.1.a.2) $[\![E']\!]_\perp(s') = \perp$
(a.1.a.3) $[\![\texttt{check}\ E_D\texttt{;}\ \texttt{continue}]\!](s, \mathit{expthrow}(s'))$

From (3) and the definition of $[\![\_]\!]_\perp$, we know

(5) $s' = \mathit{continue}(s)$

From (5) and (CD1), we know (a.1.a.1).

From (4) and the definition of $[\![\_]\!]$, we know

(6) IF $[\![E_D]\!](s) \neq$ TRUE
THEN $s' \neq \mathit{throw}(s, \mathit{EXP}, \mathit{read}(s, \texttt{VAL}))$
ELSE $s' \neq \mathit{continue}(s)$

From (5) and (6), we know

(7) $[\![E_D]\!](s) \neq$ TRUE
(8) $s' \neq \mathit{throw}(s, \mathit{EXP}, \mathit{read}(s, \texttt{VAL}))$

From (2), (7), and the definition of $[\![\_]\!]_\perp$, we know (a.1.a.2).

To show (a.1.a.3), by the definition of $[\![\_]\!]$, it suffices to show

(a.1.a.3.a) IF $[\![E_D]\!](s) \neq$ TRUE
THEN $\mathit{expthrow}(s') = \mathit{throw}(s, \mathit{EXP}, \mathit{read}(s, \texttt{VAL}))$
ELSE $\mathit{expthrow}(s') = \mathit{continue}(s)$

From (7), to show (a.1.a.3.a), it suffices to show

(a.1.a.3.b) $\mathit{expthrow}(s') = \mathit{throw}(s, \mathit{EXP}, \mathit{read}(s, \texttt{VAL}))$

From (5) and the definition of *expthrow*, it suffices to show

(a.1.a.3.c) $\mathit{throw}(\mathit{continue}(s), \mathit{EXP}, \mathit{read}(s, \texttt{VAL})) = \mathit{throw}(s, \mathit{EXP}, \mathit{read}(s, \texttt{VAL}))$

From the definitions of *throw* and *continue*, we know (a.1.a.3.c).

To show (a.2), we assume

(9) $[\![\texttt{check}\ E_D\texttt{;}\ \texttt{continue}]\!](s, s')$
(10) $\neg[\![\texttt{continue}]\!]_\perp(s, s')$

and show

(a.2.a.1) $[\![\texttt{continue}]\!]_\perp(s, \mathit{continue}(s))$
(a.2.a.2) $s' = \mathit{expthrow}(\mathit{continue}(s))$
(a.2.a.3) $\mathit{continues}(\mathit{control}(\mathit{continue}(s)))$

(a.2.a.4) $[\![E']\!]_\perp(\mathit{continue}(s)) = \perp$

From the definition of $[\![\_]\!]_\perp$, we have (a.2.a.1).

From (9) and the definition of $[\![\_]\!]$, we know

$$\begin{array}{ll} & \text{IF } [\![E_D]\!](s) \neq \text{TRUE} \\ (11) & \quad \text{THEN } s' = \mathit{throw}(s, \mathit{EXP}, \mathit{read}(s, \texttt{VAL})) \\ & \quad \text{ELSE } s' = \mathit{continue}(s) \end{array}$$

From (10) and the definition of $[\![\_]\!]_\perp$, we know

(12) $s' \neq \mathit{continue}(s)$

From (11) and (12), we know

(13) $[\![E_D]\!](s) \neq \text{TRUE}$
(14) $s' = \mathit{throw}(s, \mathit{EXP}, \mathit{read}(s, \texttt{VAL}))$

From (14) and the definitions of *throw*, *continue*, and *expthrow*, we know (a.2.a.2).

From (CD1), we have (a.2.a.3).

From (2), (13), and the definition of $[\![\_]\!]_\perp$, we know

(15) $[\![E']\!]_\perp(s) = \perp$

From (15), (CD2), and then Lemma "Well-defined Expressions", we know (a.2.a.4).

To show (a.3), we assume

(16) $[\![\texttt{check } E_D\texttt{; continue}]\!](s, s')$
(17) $\mathit{continues}(\mathit{control}(s'))$

and show

(a.3.a) $[\![E]\!]_\perp(s') \neq \perp$

From (16) and the definition of $[\![\_]\!]$, we know

$$\begin{array}{ll} & \text{IF } [\![E_D]\!](s) \neq \text{TRUE} \\ (18) & \quad \text{THEN } s' = \mathit{throw}(s, \mathit{EXP}, \mathit{read}(s, \texttt{VAL})) \\ & \quad \text{ELSE } s' = \mathit{continue}(s) \end{array}$$

From (17), (18), (CD1), and Lemma "State Control Predicates", we know

(19) $[\![E_D]\!](s) = \text{TRUE}$

From (2), (19), and the definition of $[\![\_]\!]_\perp$, we know

(20) $[\![E]\!]_\perp(s) \neq \perp$

From (14), (20), (CD2), and then Axiom "Well-Defined Expressions", we know (a.3.a).

- Case $C_0 = \texttt{while (}E\texttt{)}\ C$: from the premises of the rule, we know

  (2) $E \overset{\mathrm{D}}{\simeq} E_D$
  (3) $\mathrm{CHECK}_E(C) = C'$

  To show (a.1), we assume

  (4) $[\![\texttt{while (}E\texttt{)}\ C]\!]_\perp(s,s')$

  and show

  (a.1.a) $[\![\mathrm{CHECK}\ E_D\texttt{;}\ \texttt{while (}E\texttt{)}\ (C'\texttt{;}\ \mathrm{CHECK}\ E_D)]\!](s,s')$

  The proof follows essentially the lines of the proof of $\Rightarrow$ for loops without `continue` shown in the previous subsection. The only differences are in the subproofs of (e.2), (e.3), (e.1.c.4), and (e.1.c.5) which in the previous proof depended on the induction hypothesis and/or the assumption that C does not result in a continuing state. We repeat the proofs of these parts:

  We have to prove

  (e.2) $[\![E]\!](t_0(k)) \neq \mathrm{TRUE} \vee \neg(\mathit{executes}(\mathit{control}(u_0(k))) \vee \mathit{continues}(\mathit{control}(u_0(k))))$
  (e.3) $t_0(k) = s'$

  From (9), we have the following cases:

  - In the first case, we assume

    (24) $\neg\mathit{executes}(\mathit{control}(u(k)))$
    (25) $\neg\mathit{continues}(\mathit{control}(u(k)))$

    From (16), (24), and (25), we know (e.2).

    From (10), (16), (24), and (25), we know (e.3).

  - In the second case, we assume

    (24) $\mathit{executes}(\mathit{control}(u(k))) \vee \mathit{continues}(\mathit{control}(u(k)))$
    (25) $[\![E]\!]_\perp(t(k)) = \perp$

    From (10), (24), and (25), we know

    (26) $\mathit{expthrow}(t(k)) = s'$

    From (16), (24), and (25), we know

    (27) $t_0(k) = \mathit{expthrow}(t(k))$, $u_0(k) = \mathit{expthrow}(u(k))$

    From (27), the definition of *expthrow*, (CD1), and Lemma "Control State Predicates", we know

(28) $\neg executes(control(u_0(k)))$
(29) $\neg continues(control(u_0(k)))$

From (28) and (29), we know (e.2).

From (26) and (27), we know (e.3).

– In the third case, we assume

(24) $executes(control(u(k))) \vee continues(control(u(k)))$
(25) $[\![E]\!]_\perp(t(k)) \neq \perp$
(26) $[\![E]\!]_\perp(t(k)) \neq \text{TRUE}$

From (16), (25), and the definition of $[\![\_]\!]_\perp$, we know (e.2).

From (10), (16), and the case condition, we know (e.3).

We have to prove

(e.1.c.4)
$$\begin{array}{l}\exists s_0 \in State: \\ \quad [\![C']\!](t_0(i), s_0) \wedge \\ \quad \text{IF } executes(control(s_0)) \wedge [\![E_D]\!](s_0) \neq \text{TRUE} \\ \quad\quad \text{THEN } u_0(i+1) = expthrow(s_0) \\ \quad\quad \text{ELSE } u_0(i+1) = s_0\end{array}$$

We define

(34)
$$\begin{array}{l}s_0 := \\ \quad \text{IF } [\![C']\!](t_0(i), u(i+1)) \\ \quad\quad \text{THEN } u(i+1) \\ \quad\quad \text{ELSE } expthrow(u(i+1))\end{array}$$

From (3), (32), (33), and the induction hypothesis, we know

(35)
$$\begin{array}{l}[\![C']\!](t_0(i), u(i+1)) \vee \\ (continues(control(u(i+1))) \wedge \\ [\![E]\!]_\perp(u(i+1)) = \perp \wedge [\![C']\!](t_0(i), expthrow(u_0(i+1))))\end{array}$$

To show (e.1.c.4), it suffices to prove

(e.1.c.4.a.1) $[\![C']\!](t_0(i), s_0)$
$\text{IF } executes(control(s_0)) \wedge [\![E_D]\!](s_0) \neq \text{TRUE}$
(e.1.c.4.a.2) $\quad \text{THEN } u_0(i+1) = expthrow(s_0)$
$\quad \text{ELSE } u_0(i+1) = s_0$

In case of $[\![C']\!](t_0(i), u(i+1))$, by (16), (30), and (34), we know (e.1.c.4.a.1). Thus, by (35) we may proceed in the remainder of the proof of (e.1.c.4.a.1) with the assumptions

(36) $\neg[\![C']\!](t_0(i), u(i+1))$
(37) $continues(control(u(i+1)))$
(38) $[\![E]\!]_\perp(u(i+1)) = \perp$

(39) $[\![C']\!](t_0(i), expthrow(u_0(i+1)))$

From (16), (30), (34), (36), and (39), we know (e.1.c.4.a.1).

We first show (e.1.c.4.a.2) with the assumption

(40) $[\![C']\!](t_0(i), u(i+1))$

from which (34) implies

(41) $s_0 = u(i+1)$

We then consider three cases:

- Case $executes(control(u(i+1))) \wedge [\![E_D]\!](u(i+1)) \neq \text{TRUE}$: from (41), it suffices to show

(e.1.c.4.a.2.a) $u_0(i+1) = expthrow(u(i+1))$

From the case condition and the definition of $[\![\_]\!]_\perp$, we know

(42) $[\![E]\!]_\perp(u(i+1)) = \perp$

From (23), the case condition, and Lemma "State Control Predicates", we know

(43) $t(i+1) = u(i+1)$

From (16), (41), (42), (43), and finally the case condition, we know (e.1.c.4.a.2.a).

- Case $executes(control(u(i+1))) \wedge [\![E_D]\!](u(i+1)) = \text{TRUE}$: from (41), it suffices to show show

(e.1.c.4.a.2.a) $u_0(i+1) = u(i+1)$

From the case condition and the definition of $[\![\_]\!]_\perp$, we know

(44) $[\![E]\!]_\perp(u(i+1)) \neq \perp$

From (23), the case condition, and Lemma "State Control Predicates", we know

(45) $t(i+1) = u(i+1)$

From (16), (43), (44), (45), and finally the case condition, we know (e.1.c.4.a.2.a).

- Case $\neg executes(control(u(i+1)))$: from (41), it suffices to show

(e.1.c.4.a.2.a) $u_0(i+1) = u(i+1)$

From (16) and (30), we know

(46) $t_0(i) = t(i)$

From (32), (40), (46), and the induction hypothesis, we know

(47) $continues(control(u(i+1))) \Rightarrow [\![E]\!]_\perp(u(i+1)) \neq \perp$

From the case condition, (16), and (47), we know (e.1.c.4.a.2.a).

Now we show (e.1.c.4.a.2) with the assumption

(48) $\neg[\![C']\!](t_0(i), u(i+1))$

from which (34) implies

(49) $s_0 = \mathit{expthrow}(u(i+1))$

From (49), the definition of *expthrow*, (CD1), and Lemma "Control State Predicates", we know

(50) $\neg \mathit{executes}(\mathit{control}(s_0))$

From (49) and (50), to show (e.1.c.4.a.2), it suffices to show

(e.1.c.4.a.2.a) $u_0(i+1) = \mathit{expthrow}(u(i+1))$

From (16), it suffices to show

(e.1.c.4.a.2.b.1) $\mathit{executes}(\mathit{control}(u(i+1))) \vee \mathit{continues}(\mathit{control}(u(i+1)))$
(e.1.c.4.a.2.b.2) $[\![E]\!]_\perp(u(i+1)) = \perp$

From (16) and (30), we know

(51) $t_0(i) = t(i)$

From (33), (48), (51), and the ind. hypothesis, we know (e.1.c.4.a.2.b.1) and (e.1.c.4.a.2.b.2).

We prove

(e.1.c.5)
$$\begin{array}{l}\text{IF } \mathit{continues}(\mathit{control}(u_0(i+1))) \vee \mathit{breaks}(\mathit{control}(u_0(i+1))) \\ \quad \text{THEN } t_0(i+1) = \mathit{execute}(u_0(i+1)) \\ \quad \text{ELSE } t_0(i+1) = u_0(i+1)\end{array}$$

From (16) and (30), we know

(45) $t_0(i) = t(i)$

We first show (e.1.c.5) under the assumption

(46) $[\![C']\!](t(i), u(i+1))$

From (3), (32) and (46), we know by the induction hypothesis

(47) $\neg(\mathit{continues}(\mathit{control}(u(i+1))) \wedge [\![E]\!]_\perp(u(i+1)) = \perp)$

By Lemma "State Control Predicates", it suffices to consider these cases:

- Case $\mathit{continues}(\mathit{control}(u_0(i+1))) \vee \mathit{breaks}(\mathit{control}(u_0(i+1)))$: we show

(e.1.c.5.a) $t_0(i+1) = \mathit{execute}(u_0(i+1))$

From (16), (47), and the case condition, we know

(48) $t_0(i+1) = t(i+1)$
(49) $u_0(i+1) = u(i+1)$

From (23), (49), and the case condition, we know

(50) $t(i+1) = \mathit{execute}(u_0(i+1))$

From (48), (50), and the case condition, we know (e.1.c.5.a).

– Case $\mathit{returns}(\mathit{control}(u_0(i+1))) \vee \mathit{executes}(\mathit{control}(u_0(i+1)))$: by the Lemma "State Control Predicates", it suffices to show

(e.1.c.5.a) $t_0(i+1) = u_0(i+1)$

From (16), the case condition, Lemma "State Control Predicates", and the definition of *expthrow*, we know

(51) $t_0(i+1) = t(i+1)$
(52) $u_0(i+1) = u(i+1)$

From (23), (51), (52), the case condition, and Lemma "State Control Predicates", we know (e.1.c.5.a).

– Case $\mathit{throws}(u_0(i+1))$: it suffices to show

(e.1.c.5.a) $t_0(i+1) = u_0(i+1)$

We assume

(53) $t_0(i+1) \neq u_0(i+1)$

and show a contradiction. From (16) and (53), we know

(54) $t(i+1) \neq u(i+1)$

From (23) and (54), we know

(55) $\mathit{continues}(\mathit{control}(u(i+1))) \vee \mathit{breaks}(u(i+1))$

From (16), (47), (55), and Lemma "State Control Predicates", we know

(56) $u_0(i+1) = u(i+1)$

But (55), (56), and Lemma "State Control Predicates" together contradict the case condition.

We now show (e.1.c.5) under the assumption

(57) $\neg[\![C']\!](t(i), u(i+1))$

From (3), (32), (33), and (46), we know by the induction hypothesis

(58) $\mathit{continues}(\mathit{control}(u(i+1)))$
(59) $[\![E]\!]_{\perp}(u(i+1)) = \perp$
(60) $[\![C']\!](s, \mathit{expthrow}(s'))$

From (23) and (58), we know

(61) $t(i+1) = execute(u(i+1))$

From (59), (61), the definition of $[\![\_]\!]_\perp$, and Axiom "Well-defined Expressions", we know

(62) $[\![E]\!]_\perp(t(i+1)) = \perp$

From (16), (58), and (62), we know

(63) $t_0(i+1) = expthrow(t(i+1))$
(64) $u_0(i+1) = expthrow(u(i+1))$

From (64), and Lemma "State Control Predicates", it suffices to prove

(e.1.c.5.a) $t_0(i+1) = u_0(i+1)$

From (61), (63), and (64), it suffices to prove

(e.1.c.5.b) $expthrow(execute(u(i+1))) = expthrow(u(i+1))$

From the definition of *expthrow* and *execute*, we know (e.1.c.5.b).

To show (a.2), we assume

(4) $[\![\text{CHECK } E_D\texttt{; while (}E\texttt{) (}C'\texttt{; }\text{CHECK } E_D\texttt{)}]\!](s,s')$
(5) $\forall s'' \in State : [\![\texttt{while (}E\texttt{) }C]\!]_\perp(s,s'') \wedge s' = expthrow(s'') \Rightarrow \neg(continues(control(s'')) \wedge [\![E]\!]_\perp(s'') = \perp)$

and show

(a.1.a) $[\![\texttt{while (}E\texttt{) }C]\!]_\perp(s,s')$

The proof follows the lines of the proof of $\Leftarrow$ for loops without `continue` shown in the previous subsection. The only differences are in the subproofs of (d.2), (d.3), and (d.1) in the case $k > 0$ which in the previous proof depended on the induction hypothesis and/or the assumption that C does not result in a continuing state. We repeat the proofs of these parts:

We assume

(34) $k > 0$

From (20), (23) and (34), we know

(35) $executes(control(t(k-1)))$
(36) $[\![C']\!](t(k-1), u_0(k))$

From (3), (35), (36), and the induction hypothesis, we know

(37) $[\![C]\!]_\perp(t(k-1), u_0(k)) \vee expthrows(control(u_0(k)))$
(38) $\neg continues(control(u_0(k)) \vee [\![E]\!]_\perp(u_0(k))) \neq \perp$

We show

(d.2) $[\![E]\!]_\perp(t_0(k)) = \perp \vee$
$[\![E]\!]_\perp(t_0(k)) \neq \text{TRUE} \vee$
$\neg(\mathit{executes}(\mathit{control}(u_0(k))) \vee \mathit{continues}(\mathit{control}(u_0(k)))$

We assume

(39) $[\![E]\!]_\perp(t_0(k)) \neq \perp$
(40) $[\![E]\!]_\perp(t_0(k)) = \text{TRUE}$
(41) $\mathit{executes}(\mathit{control}(u_0(k))) \vee \mathit{continues}(\mathit{control}(u_0(k))$

and show a contradiction.

From (2), (39), and the definition of $[\![\_]\!]_\perp$ we know

(42) $[\![E_D]\!](t_0(k)) = \text{TRUE}$

From (41), we may proceed with two cases.

In case

(43) $\mathit{executes}(\mathit{control}(u_0(k)))$

we have from (31)

(44) $t_0(k) = u_0(k)$

From (30) and (44), we have

(45) $u(k) = u_0(k)$

From (32), (43), and (45), we have

(46) $t(k) = u(k)$

But (41), (42), (44), (45), and (46) contradict (16).

In case

(47) $\mathit{continues}(\mathit{control}(u_0(k)))$

we have with (30), (31), (32), and (34)

(48) $u(k) = u_0(k)$
(49) $t_0(k) = \mathit{execute}(u_0(k))$
(50) $t(k) = \mathit{execute}(u(k))$

From (48), (49), and (50), we know

(51) $t(k) = t_0(k)$

But (41), (42), (48), and (51) contradict (16).

We show

$$\text{(d.3)}\quad \begin{array}{l}\text{IF } (\mathit{executes}(\mathit{control}(u_0(k))) \vee \mathit{continues}(\mathit{control}(u_0(k)))) \wedge \\ \qquad [\![E]\!]_\perp(t_0(k)) = \perp \\ \text{THEN } \mathit{expthrow}(t_0(k)) = s' \\ \text{ELSE } t_0(k) = s'\end{array}$$

by case distinction:

– In the first case, we assume

(44) $\mathit{executes}(\mathit{control}(u_0(k))) \vee \mathit{continues}(\mathit{control}(u_0(k)))$
(45) $[\![E]\!]_\perp(t_0(k)) = \perp$

and show

(d.3.a) $\mathit{expthrow}(t_0(k)) = s'$

We first show

(d.3.a.1) $\neg\mathit{continues}(\mathit{control}(u_0(k)))$

by assuming

(46) $\mathit{continues}(\mathit{control}(u_0(k)))$

and deriving a contradiction.

From (1a), (20), (23), (34), and the induction hypothesis, we know

(47) $\neg(\mathit{continues}(\mathit{control}(u_0(k))) \wedge [\![E]\!]_\perp(u_0(k)) = \perp)$

From (46) and (47), we know

(48) $[\![E]\!]_\perp(u_0(k)) \neq \perp$

From (30), (34), and (30), we know

(49) $u(k) = u_0(k)$

From (31), (34), and (46), we know

(50) $t_0(k) = \mathit{execute}(u_(k))$

From (48), (49), (50), (CD2), and Axiom "Well-defined Expressions", we have a contradiction with (45).

From (44) and (d.3.a.1), we know

(51) $\mathit{executes}(\mathit{control}(u_0(k)))$

From (2), (45), and the definition of $[\![\_]\!]_\perp$, we know

(52) $[\![E_D]\!](t_0(k)) \neq \text{TRUE}$

From (31), (34), and (52), we know

(53) $t_0(k) = u_0(k)$

From (30), (52), (52), and (53), we know

(54) $u(k) = \mathit{expthrow}(u_0(k))$

From (32), (34), (54), and the definition of *expthrow*, we know

(55) $t(k) = u(k)$

From (17), (53), (54), and (55), we know (d.3.a).

– In the second case, we assume

(56) $executes(control(u_0(k))) \vee continues(control(u_0(k)))$
(57) $[\![E]\!]_\perp(t_0(k)) \neq \perp$

and show

(d.3.a) $t_0(k) = s'$

According to (56), we may proceed with two cases:

∗ In the first case, we assume

(58) $executes(control(u_0(k)))$

From (2), (57), and the definition of $[\![\_]\!]_\perp$, we know

(59) $[\![E_D]\!](t_0(k)) = \text{TRUE}$

From (31), (34), and (58), we know

(60) $t_0(k) = u_0(k)$

From (30), (44), and (59), we know

(61) $u(k) = u_0(k)$

From (32), (34), (58), (61), and Lemma "State Control Predicates", we know

(62) $t(k) = u(k)$

From (17), (60), (61), and (62), we know (d.3.a).

∗ In the second case, we assume

(63) $continues(control(u_0(k)))$

From (2), (57), and the definition of $[\![\_]\!]_\perp$, we know

(64) $[\![E_D]\!](t_0(k)) = \text{TRUE}$

From (31), (34), (64), (CD2), and Axiom "Well-defined Expressions", we know

(65) $[\![E_D]\!](u_0(k)) = \text{TRUE}$

From (30), (34), and (65), we know

(66) $u(k) = u_0(k)$

From (31), (34), and (63), we know

(67) $t_0(k) = execute(u_0(k))$

From (32), (34), (63), and (66), we know

(68) $t(k) = execute(u(k))$

From (66), (67), and (68), we know

(69) $t(k) = t_0(k)$

From (17) and (69), we know (d.3.a).

– In the third case, we assume

(70) $\neg executes(control(u_0(k)))$
(72) $\neg continues(control(u_0(k)))$

and show

(d.3.a) $t_0(k) = s'$

From (30), (34), and (60), we know

(72) $u(k) = u_0(k)$

First, we consider the case

(73) $\mathit{breaks}(\mathit{control}(u_0(k)))$

From (31), (34), and (73), we know

(74) $t_0(k) = \mathit{execute}(u_0(k))$

From (32), (34), (72), and (73), we know

(75) $t(k) = \mathit{execute}(u(k))$

From (72), (74), and (75), we know

(76) $t(k) = t_0(k)$

From (17) and (76), we know (d.3.a).

Now, we consider the case

(77) $\neg\mathit{breaks}(\mathit{control}(u_0(k)))$

From (31), (34), (71), and (76), we know

(78) $t_0(k) = u_0(k)$

From (32), (34), (71), (72), and (77), we know

(79) $t(k) = u(k)$

From (72), (78), and (79), we know

(80) $t(k) = t_0(k)$

From (17) and (80), we know (d.3.a).

It remains to show

(d.1) $\mathit{finiteExecution}_\perp(k, t_0, u_0, s, [\![E]\!]_\perp, [\![C]\!]_\perp)$

By the definition of $\mathit{finiteExecution}_\perp$, we have to show

(d.1.a.1) $t_0(0) = s$

(d.1.a.2) $u_0(0) = s$

$\forall i \in \mathbb{N}_k$:

$\neg\mathit{breaks}(\mathit{control}(u_0(i))) \wedge \mathit{executes}(\mathit{control}(t_0(i))) \wedge$

$[\![E]\!]_\perp(t_0(i)) \neq \perp \wedge [\![E]\!]_\perp(t_0(i)) = \text{TRUE} \wedge$

(d.2.a.3) $[\![C]\!]_\perp(t_0(i), u_0(i+1)) \wedge$

IF $\mathit{continues}(\mathit{control}(u_0(i+1))) \vee \mathit{breaks}(\mathit{control}(u_0(i+1)))$

THEN $t_0(i+1) = \mathit{execute}(u_0(i+1))$

ELSE $t_0(i+1) = u_0(i+1)$

From (18), (19), (26), and (27), we know (d.1.a.1) and (d.1.a.2).

To show (d.2.a.3), we take arbitrary $i \in \mathbb{N}_k$ and show

(d.2.a.3.a.1) $\neg\mathit{breaks}(\mathit{control}(u_0(i)))$

(d.2.a.3.a.2) $\mathit{executes}(\mathit{control}(t_0(i)))$

(d.2.a.3.a.3) $[\![E]\!]_\perp(t_0(i)) \neq \perp$
(d.2.a.3.a.4) $[\![E]\!]_\perp(t_0(i)) = \text{TRUE}$
(d.2.a.3.a.5) $[\![C]\!]_\perp(t_0(i), u_0(i+1))$
IF $\mathit{continues}(\mathit{control}(u_0(i+1))) \vee \mathit{breaks}(\mathit{control}(u_0(i+1)))$
(d.2.a.3.a.6) THEN $t_0(i+1) = \mathit{execute}(u_0(i+1))$
ELSE $t_0(i+1) = u_0(i+1)$

From (20), (26), and (27), we know (d.2.a.3.a.1) and (d.2.a.3.a.2).

From (20), we know

(70) $[\![E]\!]_\perp(t(i)) = \text{TRUE}$

From (29) and (70), we know (d.2.a.3.a.3) and (d.2.a.3.a.4).

To show (d.2.a.3.a.5), we assume

(71) $\neg[\![C]\!]_\perp(t_0(i), u_0(i+1))$
(72) $\forall j \in \mathbb{N}_i : [\![C]\!]_\perp(t_0(j), u_0(j+1))$

and show a contradiction. From (3), (23), (29), (d.2.a.3.a.2), (71), and the induction hypothesis, we know

(73) $\neg(\mathit{continues}(\mathit{control}(u_0(i+1))) \wedge [\![E]\!]_\perp(u_0(i+1)) = \perp)$

and for some $s'' \in \mathit{State}$

(74) $[\![C]\!]_\perp(t_0(i), s'')$
(75) $u_0(i+1) = \mathit{expthrow}(s'')$
(76) $\mathit{continues}(\mathit{control}(s''))$
(77) $[\![E]\!]_\perp(s'') = \perp$

From (23), (78), and the definition of *expthrow*, we know

(78) $u(i+1) = u_0(i+1)$

From (20), (24), (78), and the definition of *expthrow*, we know

(79) $t(i+1) = u_0(i+1)$

From (20), (75), (79), and the definition of *expthrow*, we know

(80) $i+1 = k$

From (17), (75), (79), and (80), we know

(81) $s' = \mathit{expthrow}(s'')$

From (18), (19), (20), (21), (23), (25), (26), (27), (28), (29), (72), (74), (76) and (77), one can show (we omit the details)

(82) $[\![\texttt{while (}E\texttt{) }C]\!]_\perp(s, s'')$

But (76), (77), (81), and (82), contradict (5).

From (25), we know (d.2.a.3.a.6).

It remains to show (a.3). We assume

(4) $[\![\text{CHECK } E_D\texttt{; while (}E\texttt{) (}C'\texttt{; CHECK } E_D\texttt{)}]\!](s,s')$

and show

(a.3.a) $\neg(\mathit{continues}(\mathit{control}(s')) \wedge [\![E']\!]_\perp(s') = \perp)$

From (1b), (4) and the definition of $[\![\_]\!]$, we can easily derive (we omit the details)

(5) $\neg\mathit{continues}(\mathit{control}(s'))$

which implies (a.3.a). □

Loops with `continue` (More General Alternative)

Rather than prohibiting the catching of exceptions of type *EXP*, we may also determine some exception kind *K* that is not caught by the loop body and, if the loop condition is not well defined at some `continue`, throw this exception. The loop body has now to be surrounded by a handler that catches this exception; execution then proceeds as usual with checking the loop expression.

```
while (E) {
  // exception K is not caught in the loop body
  try {
    ...
    // transformation of continue
    { if (!E_D) throw K VAL; continue }
    ...
  }
  catch(K I) { }
  check E_D;
}
```

We do not formally elaborate this solution further.

5.9.6 Reasoning about Checked Programs

From the soundness claim stated in Figure 5.45 and Lemma "Checked Commands Extended", we see that (due to the `continue` statement), checked programs in

the original expression semantics are not unconditionally equivalent to the original programs in the extended semantics with undefined expressions. In particular, the original program in the extended semantics may give rise to states that are not states of the transformed program in the new semantics. However, there is a close relationship stated by the following lemma.

Lemma (Introducing Checks 1) If a judgement $\text{CHECK}_E(C) = C'$ can be derived from the rules of the checked program calculus, then we have the following relationship between C, C', and E:

$$
\begin{array}{l}
\mathit{DifferentVariables} \wedge \\
C \text{ has no subcommand } \texttt{try}\ C_1\ \texttt{catch}\ (\mathit{EXP}\ I)\ C_2 \Rightarrow \\
\quad \forall s, s' \in \mathit{State} : \mathit{executes}(\mathit{control}(s)) \wedge [\![C]\!]_\perp(s, s') \Rightarrow \\
\qquad \text{IF } \mathit{continues}(\mathit{control}(s')) \wedge [\![E]\!]_\perp(s') = \perp \\
\qquad\quad \text{THEN } [\![C']\!](s, \mathit{expthrow}(s')) \\
\qquad\quad \text{ELSE } [\![C']\!](s, s')
\end{array}
$$

Proof Take commands C, C' and expression E such that

(1) $\text{CHECK}_E(C) = C'$

can be derived from the rules of the checked program calculus and assume

(2) *DifferentVariables*

(3) C has no subcommand $\texttt{try}\ C_1\ \texttt{catch}\ (\mathit{EXP}\ I)\ C_2$

Take arbitrary $s, s' \in \mathit{State}$ and assume

(4) $\mathit{executes}(\mathit{control}(s))$

(5) $[\![C]\!]_\perp(s, s')$

We have to show

(a.1) $\mathit{continues}(\mathit{control}(s')) \wedge [\![E]\!]_\perp(s') = \perp \Rightarrow [\![C']\!](s, \mathit{expthrow}(s'))$

(a.2) $\neg(\mathit{continues}(\mathit{control}(s')) \wedge [\![E]\!]_\perp(s') = \perp) \Rightarrow [\![C']\!](s, s')$

From (1), (2), (3), (4), (5), and finally Lemma "Checked Commands Extended", we know

(6) $[\![C']\!](s, s') \vee$
$(\mathit{continues}(\mathit{control}(s')) \wedge [\![E]\!]_\perp(s') = \perp \wedge [\![C']\!](s, \mathit{expthrow}(s')))$

(7) $[\![C']\!](s,s') \Rightarrow \neg(\mathit{continues}(\mathit{control}(s')) \wedge [\![E]\!]_\perp(s') = \perp)$

To show (a.1), we assume

(8) $\mathit{continues}(\mathit{control}(s'))$

(9) $[\![E]\!]_\perp(s') = \perp$

(10) $\neg[\![C']\!](s, \mathit{expthrow}(s'))$

and show a contradiction.

From (6) and (10), we know

(11) $[\![C']\!](s,s')$

But (8), (9), and (11) contradict (7).

To show (a.2), we assume

(12) $\neg[\![C']\!](s,s')$

and show

(a.2.a.1) $\mathit{continues}(\mathit{control}(s'))$

(a.2.a.2) $[\![E]\!]_\perp(s') = \perp$

From (6) and (12), we have (a.2.a.1) and (a.2.a.2). □

The lemma above may be used to weaken a specification of the transformed program in the original semantics (derived by the original verification calculus) in order to construct a specification of the original program in the extended semantics. The construction of this specification is the core of the following lemma.

Lemma (Introducing Checks 2) If the judgements $\text{CHECK}_E(C) = C'$ and $C' : F$ can be derived, then we have the following relationship between C, C', E and F:

$\mathit{DifferentVariables} \wedge$
C has no subcommand `try` C_1 `catch (`$\mathit{EXP}\ I$`)` $C_2 \wedge$
$E \overset{\text{D}}{\simeq} F_D \wedge$
$\#I_s$ does not occur in $F \Rightarrow$
$\forall e \in \mathit{Environment}, s, s' \in \mathit{State}, I_1, \ldots, I_n \in \text{Identifier}:$
$\mathit{executes}(\mathit{control}(s)) \wedge [\![C]\!]_\perp(s,s') \wedge$
$s = s'$ EXCEPT $I_1, \ldots, I_n \Rightarrow$
$[\![$ `IF next.continues AND`
`NOT` F_D`[next/now][`$I_1'/I_1, \ldots, I_n'/I_n$`]`
`THEN EXSTATE` $\#I_s$`:`
F`[`$\#I_s$`/next] AND` $\#I_s$`.throws` EXP
`ELSE` $F[\![e]\!](s,s')$

Proof Take C, C', E, F such that

(1) $\text{CHECK}_E(C) = C'$

(2) $C' : F$

can be derived. We assume

(3) *DifferentVariables*

(4) C has no subcommand $\texttt{try}\ C_1\ \texttt{catch}\ (\mathit{EXP}\ I)\ C_2$

(5) $E \overset{\text{D}}{\simeq} F_D$

(6) $\#I_s$ does not occur in F

Take arbitrary $e \in \mathit{Environment}, s, s' \in \mathit{State}, I_1 \ldots, I_n \in$ Identifier with

(7) $\mathit{executes}(\mathit{control}(s))$

(8) $[\![C]\!]_\perp(s, s')$

(9) $s = s'$ EXCEPT $I_1, \ldots, I_n$

It suffices to show

(a) $[\![\ \texttt{IF next.continues AND}$
$\quad \texttt{NOT}\ F_D[\texttt{next/now}][I_1\texttt{'}/I_1, \ldots, I_n\texttt{'}/I_n]$
$\quad \texttt{THEN EXSTATE}\ \#I_s\texttt{:}$
$\quad\quad F[\#I_s/\texttt{next}]\ \texttt{AND}\ \#I_s\texttt{.throws}\ \mathit{EXP}$
$\quad \texttt{ELSE}\ F\ ]\!](e)(s, s')$

To show (a), from the definition of $[\![\ \_\ ]\!]$, it suffices to show

(a.1) $\mathit{continues}(\mathit{control})(s') \wedge$
$\neg[\![F_D[\texttt{next/now}][I_1\texttt{'}/I_1, \ldots, I_n\texttt{'}/I_n]]\!](e)(s, s') \Rightarrow$
$\quad \exists c \in \mathit{Control}:$
$\quad\quad [\![F[\#I_s/\texttt{next}]]\!](e[I_s \mapsto c]_c)(s, s') \wedge$
$\quad\quad \mathit{throws}(c) \wedge \mathit{key}(c) = \mathit{EXP}$

(a.2) $(\neg\mathit{continues}(\mathit{control})(s') \vee$
$[\![F_D[\texttt{next/now}][I_1\texttt{'}/I_1, \ldots, I_n\texttt{'}/I_n]]\!](e)(s, s')) \Rightarrow$
$\quad [\![F]\!](e)(s, s')$

We define

(10) $e_0 := e[I_s \mapsto \mathit{control}(\mathit{expthrow}(s'))]_c$

To show (a.1), we assume

(11) $\mathit{continues}(\mathit{control})(s')$

(12) $\neg[\![F_D[\texttt{next/now}][I_1\texttt{'}/I_1,\ldots,I_n\texttt{'}/I_n]]\!](e)(s,s')$

and show

(a.1.1) $[\![F[\#I_s/\texttt{next}]]\!](e_0)(s,s')$

(a.1.2) $\mathit{throws}(\mathit{control}(\mathit{expthrow}(s')))$

(a.1.3) $\mathit{key}(\mathit{control}(\mathit{expthrow}(s'))) = \mathit{EXP}$

From the definition of *expthrow* and (CD1), we know (a.1.2) and (a.1.3).

From (9), (RWE), (WSE), (RVE), and (NEQ), we know

(13) s' EQUALS $\mathit{writes}(s,I_1,\mathit{read}(s',I_1),\ldots,I_n,\mathit{read}(s',I_n))$

From (12), (13), (ESF'), (CD0), (CD4), (PNNF1), and (PPVF1'), we know

(14) $[\![F_D]\!](e)(s',s')$

From (5), (14), and the definition $\overset{\mathrm{D}}{\simeq}$, we know

(15) $[\![E]\!]_\perp(s') = \perp$

From (1), (3), (4), (7), (8), (11), (15), and Lemma "Introducing Checks 1", we know

(16) $[\![C']\!](s,\mathit{expthrow}(s'))$

From (2), (3), (4), (7), (16), and Lemma "Soundness of the Verification Calculus", we know

(17) $[\![F]\!](e)(s,\mathit{expthrow}(s'))$

From (6), (10), (17), and (CNEF2), we know

(18) $[\![F[\#I_s/\texttt{next}]]\!](e_0)(s,\mathit{expthrow}(s'))$

From (18), the definition of *expthrow*, (CD2), (CD3), (CNEF0), and (PVFNE), we know (a.1.1).

To show (a.2), we assume

(19) $\neg continues(control(s')) \vee$
$[\![F_D[\texttt{next/now}][I_1\texttt{'}/I_1,\ldots,I_n\texttt{'}/I_n]]\!](e)(s,s')$

and show

(a.2.a) $[\![F]\!](e)(s,s')$

Assume we can show

(a.2.b) $[\![C']\!](s,s')$

From (2), (3), (7), and (a.2.b), we have (a.2.a).

It remains to show (a.2.b). According to (19), we may proceed with two cases.

In the first case, we assume

(20) $\neg continues(control(s'))$

From (3), (4), (7), (8), (20), and Lemma "Introducing Checks 1", we know (a.2.b).

In the second case, we assume

(21) $[\![F_D[\texttt{next/now}][I_1\texttt{'}/I_1,\ldots,I_n\texttt{'}/I_n]]\!](e)(s,s')$

From (9), (RWE), (WSE), (RVE), and (NEQ), we know

(22) s' EQUALS $writes(s,I_1,read(s',I_1),\ldots,I_n,read(s',I_n))$

From (21), (22), (ESF'), (CD0), (CD4), (PNNF1), and (PPVF1'), we know

(23) $[\![F_D]\!](e)(s',s')$

From (5), (23), and the definition $\overset{\text{D}}{\simeq}$, we know

(24) $[\![E]\!]_\perp(s') \neq \perp$

From (3), (4), (7), (8), (24), and finally Lemma "Introducing Checks 1", we know (a.2.b). □

5.9.7 Handling Undefined Expressions

We will now summarize our conclusions about handling undefined expressions.

We have constructed an alternative command semantics $[\![C]\!]_\perp$; under certain restrictions (expression evaluation exceptions are not caught), every poststate of the alternative semantics is also a poststate of the original semantics $[\![C]\!]$ or otherwise a poststate that throws an "evaluation exception expression" (Lemma "Commands with Partial Expressions"). A specification F where $C : F$ is valid in the original semantics is thus in general not valid in the alternative semantics. While we can derive a specification

F `OR next.throws` *EXP*

that is also valid in the alternative semantics, this new specification is apparently much weaker than the original one. For overcoming this problem, we have presented two solutions.

The first solution is to introduce a calculus for "avoiding undefined expressions". By this calculus, we can rule out that states with undefined expressions may occur in the execution of a program such that the $C : F$ is also valid in the alternative semantics. While this solution is simple (and thus probably preferred in most cases), it also prohibits certain "real world" programs that actually make use of the "exception semantics" of undefined expressions (by catching and handling such exceptions).

As an alternative, we have therefore described a program translation where

- every expression evaluation is preceded by a "checking command" that verifies the definedness of the expression and raises an "evaluation exception", if the expression is not well-defined in the current state, and
- every occurrence of `continue` is preceded by a checking command that verifies the definedness of the expression of the enclosing loop.

By translating command C to a "checked" version C', the hope was that any specification F where $C' : F$ is valid in the original semantics makes also $C : F$ valid in the alternative semantics. Unfortunately, this is not fully the case: in the alternative semantics, commands may yield a "continuing" state with an undefined loop expression (which will later lead to a "throwing" state when the loop expression is evaluated) but the transformed command immediately raises an evaluation exception without passing through the continuing state. Only if we can rule out continuing states with undefined loop expressions (in particular, if the command

cannot trigger a "continuing" state as can be easily found out from the verification calculus), $C : F$ is also valid in the alternative semantics.

In general, however, as described by Lemma "Introducing Checks 2" we have to transform F into another specification F'

```
IF next.continues AND
     NOT F_D[next/now][I_1'/I_1,...,I_n'/I_n]
  THEN EXSTATE #I_s:
     F[#I_s/next] AND #I_s.throws EXP
  ELSE F
```

(where $I_1, \ldots, I_n$ is the frame of variables modified by C and F_D denotes the definedness condition of the expression of the enclosing loop) such that $C : F'$ holds according to the extended semantics with undefined expressions. If C does not occur inside a loop, then F' is identical to F.

The restriction "C has no subcommand `try` C_1 `catch (`*EXP I*`)` C_2" (that appears in the correctness claim of the transformed program) was only introduced to simplify the elaboration. Actually, as shown in Subsection 5.9.5 (without proof), we can abandon this restriction at the price of a slightly more complicated transformation of loop bodies to cover those cases where the loop expression is undefined when a `continue` statement is executed inside a protected code block which catches the "evaluation exception" *EXP*.

Based on above results, a full calculus for deriving specifications from "checked programs" is now rather straight-forward to elaborate.

5.9.8 Well-Defined Expressions

We conclude this section by introducing a small expression language that concretizes the framework elaborated in the previous subsections. For this purpose, we assume that the domain *Value* of expressible values includes the domain $\mathbb{B}$ of logical values TRUE and FALSE, a domain *INT* of bounded integers (modeling machine numbers) including 0 and 1 and the domain INT^* of finite sequences of such integers (modeling arrays of machine numbers). Figure 5.46 gives formal definition of the domains and of the associated operations.

The definitions follow the usual semantics of programming languages, e.g. the domain *Int* contains $2M$ numbers $\{-M, \ldots, -1, 0, 1, \ldots, M-1\}$ and the arithmetic operations $\ominus$, $\oplus$, $\oslash$ model bounded integer arithmetic on this domain (we omit the proof that the result is indeed in *Int*). Furthermore, *null*, *new*, *length*, *get*, and *put* model the "null" array (identified with an array of length 0), creating an array of a

certain length, determining the length of an array, reading the element of an array at a certain index and updating an array element at a certain index by a new value.

The functions $\oslash$, *get*, and *put* are made total by returning arbitrary results for division by zero and array access outside the legal index range (we omit the proof that the resulting functions are indeed total). According to the mathematical definition, *new* and *length* are already total because, e.g., $new(-1) = null$ and $length(null) = 0$. However, in the following, we will pretend that *new* is undefined on negative arguments and *length* is undefined on *null* as it is usual in most programming languages.

Figure 5.47 introduces a couple of predicates and functions for the formula language; their interpretation depends on the previously introduced operations.

Figure 5.48 introduces the expression language, and gives it a "total" semantics $[\![\_]\!]$ based on the semantic domains introduced above. On this basis, Figure 5.49 defines the "partial" semantics $[\![\_]\!]_\bot$ that returns $\bot$ in those cases where the result of $[\![\_]\!]$ is to be ignored. The construction makes use of the function $[\![\_]\!]_{\mathrm{D}}$ that determines the "definedness" predicate for a given expression. Here we assume that the expressions have been statically type-checked in the usual way such that expressions yielding Boolean values, integers, and arrays are kept apart and there is no need to deal with typing issues in the definedness predicate (for dynamically typed programming languages this however would be necessary).

For verifying that a program must not encounter states where an expression is undefined, Figure 5.51 introduces a function $[\![\_]\!]_{\mathrm{DF}}$ that constructs from every expression a "definedness formula". This constructions makes use of the translation of expressions to terms described in Figure 5.50 (expressions not denoting terms but formulas are translated to the constant `error`, but such applications of the translation function do not occur in well-typed programs). Please note how programming language operations are mapped to predicates and functions of the formula language with adequate interpretations.

The core claim is then stated as follows.

Lemma (Definedness Formulas) The translation $[\![\_]\!]_{\mathrm{DF}}$ gives valid definedness formulas:

$$\forall E \in \text{Expression} : E \overset{\mathrm{D}}{\simeq} [\![E]\!]_{\mathrm{DF}}$$

Proof Take arbitrary $E \in \text{Expression}$ and $s \in \mathit{State}$. It suffices to prove

(a) $[\![E]\!]_{\mathrm{D}}(s) \Leftrightarrow [\![[\![E]\!]_{\mathrm{DF}}]\!](s)$

Semantic Domains and Operations

$trunc : \mathbb{Q} \to \mathbb{Z}$
$trunc(r) =$
  IF $r < 0$
    THEN MIN $i \in \mathbb{Z} : r \leq i$
    ELSE MAX $i \in \mathbb{Z} : i \leq r$

$M :=$ SUCH $m \in \mathbb{N} : m > 1$

$INT := \mathbb{Z}_M$

$\ominus : INT \to INT$
$\ominus(x) =$ IF $x = -M$ THEN $-M$ ELSE $-x$

$\oplus : INT \times INT \to INT$
$\oplus(x,y) =$
  LET $s = x + y$ IN
  IF $s < -M$ THEN
    $s + M$
  ELSE IF $s > M - 1$ THEN
    $s - M$
  ELSE s

$\oslash : INT \times INT \to INT$
$\oslash(x,y) =$
  IF $y = 0$ THEN
    0
  ELSE
    LET $s = trunc(x/y)$ IN
    IF $s = M$ THEN $-M$ ELSE s

$null := \emptyset$

$new : INT \to INT^*$
$new(n) = \{\langle i, 0\rangle : i \in \mathrm{N}_n\}$

$length : INT^* \to INT$
$length(a) =$ LENGTH(a)

$get : INT^* \times INT \to INT$
$get(a,i) =$ IF $a \neq null \wedge 0 \leq i \wedge i < length(a)$ THEN $a(i)$ ELSE 0

$put : INT^* \times INT \times INT \to INT^*$
$put(a,i,x) =$
  IF $a \neq null \wedge 0 \leq i \wedge i < length(a)$ THEN $a[i \mapsto x]$ ELSE a

Figure 5.46: Semantic Algebras

Formulas: Predicates and Constants

$[\![\_]\!]$: **Predicate** $\rightarrow$ ***Predicate***

$[\![\texttt{eq}]\!](v_1, v_2) \Leftrightarrow v_1 = v_2$
$[\![\texttt{lt}]\!](v_1, v_2) \Leftrightarrow v_1 < v_2$
$[\![\texttt{le}]\!](v_1, v_2) \Leftrightarrow v_1 \leq v_2$

$[\![\_]\!]$: **Function** $\rightarrow$ ***Function***

$[\![\texttt{error}]\!] = 0$
$[\![\texttt{zero}]\!] = 0$
$[\![\texttt{one}]\!] = 1$
$[\![\texttt{neg}]\!](v) = \ominus v$
$[\![\texttt{add}]\!](v_1, v_2) = v_1 \oplus v_2$
$[\![\texttt{div}]\!](v_1, v_2) = v_1 \oslash v_2$
$[\![\texttt{null}]\!] = \mathit{null}$
$[\![\texttt{new}]\!](v) = \mathit{new}(v)$
$[\![\texttt{length}]\!](v) = \mathit{length}(v)$
$[\![\texttt{put}]\!](v_1, v_2, v_3) = \mathit{put}(v_1, v_2, v_3)$
$[\![\texttt{get}]\!](v_1, v_2) = \mathit{get}(v_1, v_2)$

Figure 5.47: Predicates and Constants of the Formula Language

Syntax

$E \in$ Expression.

$E ::=$

$I \mid$
`0` $\mid$ `1` $\mid -E \mid E_1$`+`$E_2 \mid E_1 / E_2 \mid$
`true` $\mid$ `false` $\mid$ `!`$E \mid E_1$`&&`$E_2 \mid E_1$`||`$E_2 \mid$
E_1`==`$E_2 \mid E_1$`<`$E_2 \mid E_1$`<=`$E_2 \mid$
`null` $\mid$ `new` $E \mid E_1$`.length` $\mid E_1$`[`E_2`]` $\mid E_1$`[`E_2`|->`E_3`]`

Total Semantics

$[\![\_]\!]$: **Expression** $\rightarrow$ ***StateFunction***

$[\![I]\!](s) = read(s, I)$
$[\![\texttt{0}]\!](s) = 0$
$[\![\texttt{1}]\!](s) = 1$
$[\![-E]\!](s) = \ominus [\![E]\!](s)$
$[\![E_1 \texttt{+} E_2]\!](s) = [\![E_1]\!](s) \oplus [\![E_2]\!](s)$
$[\![E_1 / E_2]\!](s) = [\![E_1]\!](s) \oslash [\![E_2]\!](s)$
$[\![\texttt{true}]\!](s) =$ TRUE
$[\![\texttt{false}]\!](s) =$ FALSE
$[\![\texttt{!} E]\!](s) =$ IF $[\![E]\!](s) =$ TRUE THEN FALSE ELSE TRUE
$[\![E_1 \texttt{\&\&} E_2]\!](s) =$ IF $[\![E_1]\!](s) =$ TRUE THEN $[\![E_2]\!](s)$ ELSE FALSE
$[\![E_1 \texttt{||} E_2]\!](s) =$ IF $[\![E_1]\!](s) =$ TRUE THEN TRUE ELSE $[\![E_2]\!](s)$
$[\![E_1 \texttt{==} E_2]\!](s) =$ IF $[\![E_1]\!](s) = [\![E_2]\!](s)$ THEN TRUE ELSE FALSE
$[\![E_1 \texttt{<} E_2]\!](s) =$ IF $[\![E_1]\!](s) < [\![E_2]\!](s)$ THEN TRUE ELSE FALSE
$[\![E_1 \texttt{<=} E_2]\!](s) =$ IF $[\![E_1]\!](s) \leq [\![E_2]\!](s)$ THEN TRUE ELSE FALSE
$[\![\texttt{null}]\!](s) = null$
$[\![\texttt{new} E]\!](s) = new([\![E]\!](s))$
$[\![E \texttt{.length}]\!](s) = length([\![E]\!](s))$
$[\![E_1 \texttt{[} E_2 \texttt{]}]\!](s) = get([\![E_1]\!](s), [\![E_2]\!](s))$
$[\![E_1 \texttt{[} E_2 \texttt{|->} E_3 \texttt{]}]\!](s) = put([\![E_1]\!](s), [\![E_2]\!](s), [\![E_3]\!](s))$

Figure 5.48: An Expression Language (Part 1/2)

Partial Semantics

$[\![\_]\!]_\perp$: **Expression** $\rightarrow$ ***StateFunction***$_\perp$
$[\![E]\!]_\perp(s) =$ IF $[\![E]\!]_D(s)$ THEN $[\![E]\!](s)$ ELSE $\perp$

$[\![\_]\!]_D$: **Expression** $\rightarrow$ ***StatePredicate***
$[\![I]\!]_D(s) \Leftrightarrow$ TRUE
$[\![\texttt{0}]\!]_D(s) \Leftrightarrow$ TRUE
$[\![\texttt{1}]\!]_D(s) \Leftrightarrow$ TRUE
$[\![-E]\!]_D(s) \Leftrightarrow [\![E]\!]_D(s)$
$[\![E_1 \texttt{+} E_2]\!]_D(s) \Leftrightarrow [\![E_1]\!]_D(s) \wedge [\![E_2]\!]_D(s)$
$[\![E_1 \texttt{/} E_2]\!]_D(s) \Leftrightarrow [\![E_1]\!]_D(s) \wedge [\![E_2]\!]_D(s) \wedge [\![E_2]\!](s) \neq 0$
$[\![\texttt{true}]\!]_D(s) \Leftrightarrow$ TRUE
$[\![\texttt{false}]\!]_D(s) \Leftrightarrow$ TRUE
$[\![\texttt{!} E]\!]_D(s) \Leftrightarrow [\![E]\!]_D(s)$
$[\![E_1 \texttt{\&\&} E_2]\!]_D(s) \Leftrightarrow [\![E_1]\!]_D(s) \wedge [\![E_2]\!]_D(s)$
$[\![E_1 \texttt{||} E_2]\!]_D(s) \Leftrightarrow [\![E_1]\!]_D(s) \wedge [\![E_2]\!]_D(s)$
$[\![E_1 \texttt{==} E_2]\!]_D(s) \Leftrightarrow [\![E_1]\!]_D(s) \wedge [\![E_2]\!]_D(s)$
$[\![E_1 \texttt{<} E_2]\!]_D(s) \Leftrightarrow [\![E_1]\!]_D(s) \wedge [\![E_2]\!]_D(s)$
$[\![E_1 \texttt{<=} E_2]\!]_D(s) \Leftrightarrow [\![E_1]\!]_D(s) \wedge [\![E_2]\!]_D(s)$
$[\![\texttt{null}]\!]_D(s) \Leftrightarrow$ TRUE
$[\![\texttt{new}\ E]\!]_D(s) \Leftrightarrow [\![E]\!]_D(s) \wedge 0 \leq [\![E]\!]$
$[\![E\texttt{.length}]\!]_D(s) \Leftrightarrow [\![E]\!]_D(s) \wedge [\![E]\!](s) \neq \mathit{null}$
$[\![E_1 \texttt{[} E_2 \texttt{]}]\!]_D(s) \Leftrightarrow [\![E_1]\!]_D(s) \wedge [\![E_2]\!]_D(s) \wedge$
$\quad [\![E_1]\!](s) \neq \mathit{null} \wedge 0 \leq [\![E_2]\!](s) \wedge [\![E_2]\!](s) < \mathit{length}([\![E_1]\!](s))$
$[\![E_1 \texttt{[} E_2 \texttt{|->} E_3 \texttt{]}]\!]_D(s) \Leftrightarrow [\![E_1]\!]_D(s) \wedge [\![E_2]\!]_D(s) \wedge [\![E_3]\!]_D(s) \wedge$
$\quad [\![E_1]\!](s) \neq \mathit{null} \wedge 0 \leq [\![E_2]\!](s) \wedge [\![E_2]\!](s) < \mathit{length}([\![E_1]\!](s))$

Figure 5.49: An Expression Language (Part 2/2)

Term Translations

$[\![\_]\!]_{\text{TERM}}$: **Expression** $\rightarrow$ **Term**

$[\![I]\!]_{\text{TERM}} = I$

$[\![\texttt{0}]\!]_{\text{TERM}} = \texttt{0}$

$[\![\texttt{1}]\!]_{\text{TERM}} = \texttt{1}$

$[\![\texttt{-}E]\!]_{\text{TERM}} = \texttt{neg(}[\![E]\!]_{\text{TERM}}\texttt{)}$

$[\![E_1\texttt{+}E_2]\!]_{\text{TERM}} = \texttt{add(}[\![E_1]\!]_{\text{TERM}}\texttt{,}[\![E_2]\!]_{\text{TERM}}\texttt{)}$

$[\![E_1\texttt{/}E_2]\!]_{\text{TERM}} = \texttt{div(}[\![E_1]\!]_{\text{TERM}}\texttt{,}[\![E_2]\!]_{\text{TERM}}\texttt{)}$

$[\![\texttt{true}]\!]_{\text{TERM}} = \texttt{error}$

$[\![\texttt{false}]\!]_{\text{TERM}} = \texttt{error}$

$[\![\texttt{!}E]\!]_{\text{TERM}} = \texttt{error}$

$[\![E_1\texttt{\&\&}E_2]\!]_{\text{TERM}} = \texttt{error}$

$[\![E_1\texttt{||}E_2]\!]_{\text{TERM}} = \texttt{error}$

$[\![E_1\texttt{==}E_2]\!]_{\text{TERM}} = \texttt{error}$

$[\![E_1\texttt{<}E_2]\!]_{\text{TERM}} = \texttt{error}$

$[\![E_1\texttt{<=}E_2]\!]_{\text{TERM}} = \texttt{error}$

$[\![\texttt{null}]\!]_{\text{TERM}} = \texttt{null}$

$[\![\texttt{new}E]\!]_{\text{TERM}} = \texttt{new(}[\![E]\!]_{\text{TERM}}\texttt{)}$

$[\![E\texttt{.length}]\!]_{\text{TERM}} = \texttt{length(}[\![E]\!]_{\text{TERM}}\texttt{)}$

$[\![E_1\texttt{[}[\![E_2]\!]_{\text{TERM}}\texttt{]}]\!]_{\text{TERM}} = \texttt{get(}[\![E_1]\!]_{\text{TERM}}\texttt{,}[\![E_2]\!]_{\text{TERM}}\texttt{)}$

$[\![E_1\texttt{[}[\![E_2]\!]_{\text{TERM}}\texttt{|->}E_3\texttt{]}]\!]_{\text{TERM}} =$
$\quad \texttt{put(}[\![E_1]\!]_{\text{TERM}}\texttt{,}[\![E_2]\!]_{\text{TERM}}\texttt{,}[\![E_3]\!]_{\text{TERM}}\texttt{)}$

Figure 5.50: Term Translations

Definedness Formulas

$[\![\_]\!]_{\mathrm{DF}}$: **Expression** → **Formula**

$[\![I]\!]_{\mathrm{DF}} = \texttt{TRUE}$
$[\![\texttt{0}]\!]_{\mathrm{DF}} = \texttt{TRUE}$
$[\![\texttt{1}]\!]_{\mathrm{DF}} = \texttt{TRUE}$
$[\![\texttt{-}E]\!]_{\mathrm{DF}} = [\![E]\!]_{\mathrm{DF}}$
$[\![E_1\texttt{+}E_2]\!]_{\mathrm{DF}} = [\![E_1]\!]_{\mathrm{DF}}$ AND $[\![E_2]\!]_{\mathrm{DF}}$
$[\![E_1\texttt{/}E_2]\!]_{\mathrm{DF}} = [\![E_1]\!]_{\mathrm{DF}}$ AND $[\![E_2]\!]_{\mathrm{DF}}$ `&& !eq(`$[\![E_2]\!]_{\mathrm{TERM}}$`, zero)`
$[\![\texttt{true}]\!]_{\mathrm{DF}} = \texttt{TRUE}$
$[\![\texttt{false}]\!]_{\mathrm{DF}} = \texttt{TRUE}$
$[\![\texttt{!}E]\!]_{\mathrm{DF}} = [\![E]\!]_{\mathrm{DF}}$
$[\![E_1\texttt{\&\&}E_2]\!]_{\mathrm{DF}} = [\![E_1]\!]_{\mathrm{DF}}$ AND $[\![E_2]\!]_{\mathrm{DF}}$
$[\![E_1\texttt{||}E_2]\!]_{\mathrm{DF}} = [\![E_1]\!]_{\mathrm{DF}}$ AND $[\![E_2]\!]_{\mathrm{DF}}$
$[\![E_1\texttt{==}E_2]\!]_{\mathrm{DF}} = [\![E_1]\!]_{\mathrm{DF}}$ AND $[\![E_2]\!]_{\mathrm{DF}}$
$[\![E_1\texttt{<}E_2]\!]_{\mathrm{DF}} = [\![E_1]\!]_{\mathrm{DF}}$ AND $[\![E_2]\!]_{\mathrm{DF}}$
$[\![E_1\texttt{<=}E_2]\!]_{\mathrm{DF}} = [\![E_1]\!]_{\mathrm{DF}}$ AND $[\![E_2]\!]_{\mathrm{DF}}$
$[\![\texttt{null}]\!]_{\mathrm{DF}} = \texttt{TRUE}$
$[\![\texttt{new}\ E]\!]_{\mathrm{DF}} = [\![E]\!]_{\mathrm{DF}}$ AND `le(zero,` $[\![E]\!]_{\mathrm{TERM}}$`)`
$[\![E\texttt{.length}]\!]_{\mathrm{DF}} = [\![E]\!]_{\mathrm{DF}}$ AND `!eq(`$[\![E]\!]_{\mathrm{TERM}}$`, null)`
$[\![E_1\texttt{[}E_2\texttt{]}]\!]_{\mathrm{DF}} = [\![E_1]\!]_{\mathrm{DF}}$ AND $[\![E_2]\!]_{\mathrm{DF}}$ AND

```
    !eq(⟦E1⟧TERM, null) AND
    le(zero,⟦E2⟧TERM) AND
    lt(⟦E2⟧TERM,length(⟦E1⟧TERM))
```

$[\![E_1\texttt{[}E_2\texttt{|->}E_3\texttt{]}]\!]_{\mathrm{DF}} = [\![E_1]\!]_{\mathrm{DF}}$ AND $[\![E_2]\!]_{\mathrm{DF}}$ AND $[\![E_3]\!]_{\mathrm{DF}}$ AND

```
    !eq(⟦E1⟧TERM, null) AND
    le(zero,⟦E2⟧TERM) AND
    lt(⟦E2⟧TERM,length(⟦E1⟧TERM))
```

Figure 5.51: Definedness Formulas

Definedness Expressions

$[\![\_]\!]_{\mathrm{DE}}$: **Expression** → **Expression**

$[\![I]\!]_{\mathrm{DE}} = \texttt{true}$
$[\![\texttt{0}]\!]_{\mathrm{DE}} = \texttt{true}$
$[\![\texttt{1}]\!]_{\mathrm{DE}} = \texttt{true}$
$[\![\texttt{-}E]\!]_{\mathrm{DE}} = [\![E]\!]_{\mathrm{DE}}$
$[\![E_1\texttt{+}E_2]\!]_{\mathrm{DE}} = [\![E_1]\!]_{\mathrm{DE}}$ `&&` $[\![E_2]\!]_{\mathrm{DE}}$
$[\![E_1\texttt{/}E_2]\!]_{\mathrm{DE}} = [\![E_1]\!]_{\mathrm{DE}}$ `&&` $[\![E_2]\!]_{\mathrm{DE}}$ `&& !(`E_2 `== 0)`
$[\![\texttt{true}]\!]_{\mathrm{DE}} = \texttt{true}$
$[\![\texttt{false}]\!]_{\mathrm{DE}} = \texttt{true}$
$[\![\texttt{!}E]\!]_{\mathrm{DE}} = [\![E]\!]_{\mathrm{DE}}$
$[\![E_1\texttt{\&\&}E_2]\!]_{\mathrm{DE}} = [\![E_1]\!]_{\mathrm{DE}}$ `&&` $[\![E_2]\!]_{\mathrm{DE}}$
$[\![E_1\texttt{||}E_2]\!]_{\mathrm{DE}} = [\![E_1]\!]_{\mathrm{DE}}$ `&&` $[\![E_2]\!]_{\mathrm{DE}}$
$[\![E_1\texttt{==}E_2]\!]_{\mathrm{DE}} = [\![E_1]\!]_{\mathrm{DE}}$ `&&` $[\![E_2]\!]_{\mathrm{DE}}$
$[\![E_1\texttt{<}E_2]\!]_{\mathrm{DE}} = [\![E_1]\!]_{\mathrm{DE}}$ `&&` $[\![E_2]\!]_{\mathrm{DE}}$
$[\![E_1\texttt{<=}E_2]\!]_{\mathrm{DE}} = [\![E_1]\!]_{\mathrm{DE}}$ `&&` $[\![E_2]\!]_{\mathrm{DE}}$
$[\![\texttt{null}]\!]_{\mathrm{DE}} = \texttt{true}$
$[\![\texttt{new}\ E]\!]_{\mathrm{DE}} = [\![E]\!]_{\mathrm{DE}}$ `&& 0 <=` E
$[\![E\texttt{.length}]\!]_{\mathrm{DE}} = [\![E]\!]_{\mathrm{DE}}$`&& !(`$E$ `== null)`
$[\![E_1\texttt{[}E_2\texttt{]}]\!]_{\mathrm{DE}} = [\![E_1]\!]_{\mathrm{DE}}$ `&&` $[\![E_2]\!]_{\mathrm{DE}}$ `&&`
`!(`E_1 `== null) && 0 <=` E_2 `&&` E_2 `<` E_1`.length`
$[\![E_1\texttt{[}E_2\texttt{|->}E_3\texttt{]}]\!]_{\mathrm{DE}} = [\![E_1]\!]_{\mathrm{DE}}$ `&&` $[\![E_2]\!]_{\mathrm{DE}}$ `&&` $[\![E_3]\!]_{\mathrm{DE}}$ `&&`
`!(`E_1 `== null) && 0 <=` E_2 `&&` E_2 `<` E_1`.length`

Figure 5.52: Definedness Expressions

The proof proceeds by induction on the structure of E. We omit the details. □

For inserting runtime checks into the program that capture those situations where an expression is undefined, Figure 5.52 introduces a function $[\![\_]\!]_{\mathrm{DE}}$ that constructs from every expression a checkable "definedness expression". The core claim is then stated as follows.

Lemma (Definedness Expressions) The translation $[\![\_]\!]_{\mathrm{DE}}$ gives valid definedness expressions:

$$\forall E \in \text{Expression} : E \overset{\mathrm{D}}{\simeq} [\![E]\!]_{\mathrm{DE}}$$

Proof Take arbitrary $E \in \text{Expression}$ and $s \in \mathit{State}$. It suffices to prove

(a) $[\![E]\!]_{\mathrm{D}}(s) \Leftrightarrow [\![[\![E]\!]_{\mathrm{DE}}]\!](s) = \mathrm{TRUE}$

The proof proceeds by induction on the structure of E. We omit the details. □

Chapter 6

Methods

In this chapter we extend the command language with interruptions by the concept of *methods* (also called "procedures" or "functions" in various programming languages)[1]. For this purpose, we first introduce the notion of *contexts* that assign variables to identifiers such that different methods can operate with different sets of local variables. This generalizes the previous treatment of variable declarations; the semantics of the command language is adapted correspondingly.

Next, we introduce *method declarations* and, as a new kind of commands, *method calls*, give them a semantics, and formulate a corresponding reasoning framework. Finally, we generalize the method language to allow also *recursive method calls* and extend the reasoning framework in order to take into account the problem of non-termination due to recursion.

6.1 Programs with Contexts

We understand by the "context" of a command the set of variables (locations in the store) that may be used by the command. Because the only way by which a command may refer to the store is by identifiers mapped to variables, its context is determined by this mapping. Up to now this mapping has been fixed by the valuation function $[\![\_]\!] : \text{Identifier} \to \mathit{Variable}$, thus all commands have operated in the same context. Nevertheless, for command blocks with local variable declarations and definitions, we have simulated "context switches" by updating the variable of the identifier introduced by the declaration/definition before the command body is executed and restoring the variable to its original value afterwards, for instance:

[1] For the moment, we restrict our consideration to the "static" methods of imperative languages i.e. we do not consider the "dynamic" or "virtual" methods of object-oriented languages.

$$
\begin{array}{l}
[\![\texttt{var } I\texttt{=}E\texttt{; } C]\!](s,s') \Leftrightarrow \\
\quad \exists s_0, s_1 \in \mathit{State} : \\
\qquad s_0 = \mathit{write}(s, I, [\![E]\!](s)) \wedge \\
\qquad [\![C]\!](s_0, s_1) \wedge \\
\qquad s' = \mathit{write}(s_1, I, \mathit{read}(s, I))
\end{array}
$$

This simple strategy works smoothly because the context of a command only changes step by step by every declaration of a variable. However, if we also introduce method calls

$$I_r = I_m\,(E_1, \ldots, E_p)$$

the body of method I_m is executed in a context that is substantially different from the current one: to simulate this context change by store updates, we have to remember the current values of all variables denoted by locally declared identifiers and restore these values to the values they had outside the local declarations (which have to be captured in time). To switch back to the original context, the values of the locally introduced identifiers have to be restored to the remembered values. All in all, this shows that the strategy of simulating context changes by store updates does not scale well beyond declarations of local variables.

Before we extend our language by methods, we thus introduce an explicit notion of contexts to simplify context switches. A *context* then consists of two parts:

1. a *view* that represents the mapping of identifiers to variables and
2. a *space* that represents an infinite pool of addresses,

such that the view does not map different identifiers to the same variable and the view also does not map any identifier to an element of the space. The view thus satisfies the constraint denoted by the predicate *DifferentVariables* and the space serves as a pool of unassigned variables. By the *range* of a context we understand those variables that are accessible by the context (either through the view or through the space).

Figure 6.1 gives the formal definition of domain *Context* and introduces a function $\mathit{push}(c, I)$ that takes a context c and an identifier I and returns (with the use of an auxiliary function *take*) a new context that differs from c only in that I is mapped to a previously unassigned variable. The term $\mathit{push}(c, I_1, \ldots, I_n)$ abbreviates the repeated application of *push* such that the resulting context maps $I_1, \ldots, I_n$ to unassigned variables. As we will see later, new contexts are constructed from given ones by applications of *push* only.

Contexts: Core Definitions

$View = \text{Identifier} \to Variable$
$Space = \mathbb{P}(Variable)^{\infty}$
$Context = View \times Space$

$context : View \times Space \to Context, context(v,s) = \langle v,s \rangle$
$view : Context \to View, view(v,s) = v$
$space : Context \to Space, space(v,s) = s$

$range : Context \to Space \to \mathbb{P}(Variable)$
$range(c) = \text{range}(view(c)) \cup space(c)$

$take : Space \to (Variable \times Space)$
$take(s) = \text{LET } x = \text{SUCH } x : x \in s \text{ IN } \langle x, s \backslash \{x\} \rangle$

$push : Context \times \text{Identifier} \to Context$
$push(c,I) = \text{LET } \langle v,s \rangle = c, \langle x,s' \rangle = take(s) \text{ IN } \langle v[I \mapsto x], s' \rangle$

$push(c,I_1,\ldots,I_n) \equiv push(\ldots push(c,I_1)\ldots,I_n)$

Figure 6.1: Contexts: Core Definitions

Contexts and Identifiers

$[\![\_]\!] : \text{Identifier} \to \mathit{Context} \to \mathit{Variable}$
$[\![I]\!]^c = \mathit{view}(c)(I)$

$\mathit{DifferentVariables} \subseteq \mathit{Context}$
$\mathit{DifferentVariables}(c) \Leftrightarrow$
$\quad \forall I_1, I_2 \in \text{Identifier} : I_1 \neq I_2 \Rightarrow [\![I_1]\!]^c \neq [\![I_2]\!]^c \wedge$
$\quad \forall I \in \text{Identifier} : [\![I]\!]^c \notin \mathit{space}(c)$

$c_0 = c_1 \text{ EXCEPT } I_1, \ldots, I_n :\Leftrightarrow$
$\quad \forall I \in \text{Identifier} : I \neq I_1 \wedge \ldots \wedge I \neq I_n \Rightarrow [\![I]\!]^{c_0} = [\![I]\!]^{c_1}$

$c_0 = c_1 \text{ AT } I_1, \ldots, I_n :\equiv$
$\quad \forall I \in \text{Identifier} : I = I_1 \vee \ldots \vee I = I_n \Rightarrow [\![I]\!]^{c_0} = [\![I]\!]^{c_1}$

$c_0 \text{ EQUALS } c_1 :\equiv$
$\quad \forall I \in \text{Identifier} : [\![I]\!]^{c_0} = [\![I]\!]^{c_1}$

Figure 6.2: Contexts and Identifiers

In Figure 6.2 we see that the meaning of an identifier now becomes relative to the context: the valuation function of identifiers is generalized such that $[\![I]\!]^c$ denotes the variable assigned to identifier I in context c. The soundness condition $\mathit{DifferentVariables}(c)$ constrains the view and the space of a context c as described above. The condition $c_0 = c_1 \text{ EXCEPT } I_1, \ldots, I_n$ states that contexts c_0 and c_1 assign identical variables to all identifiers except $I_1, \ldots, I_n$. Likewise, $c_0 = c_1 \text{ AT } I_1, \ldots, I_n$ states that contexts c_0 and c_1 assign identical variables to the identifiers $I_1, \ldots, I_n$.

Moreover, reading and writing stores becomes dependent on the context; the corresponding operations are generalized as shown in Figure 6.3 by the definitions of $\mathit{read}(s,I)^c$ and $\mathit{write}(s,I,v)^c$. The formulas $\_ \text{ EQUALS}^c \_$ and $\_ = \_ \text{ EXCEPT}^c I_1, \ldots, I_n$ are corresponding generalizations of their original forms. The formulas $\_ \text{ EQUALS}_c^{c'} \_$ and $\_ = \_ \text{ EXCEPT}_c^{c'} I_1, \ldots, I_n$ describe corresponding relations between stores in different contexts.

The function *push* preserves this constraint as stated by the following lemma.

Lemma (Soundness of Contexts) *push* preserves the soundness of contexts:

Contexts and States

$read : State \times \text{Identifier} \times \text{Context} \to Value$
$read(s,I)^c = store(s)([\![I]\!]^c)$

$write : State \times \text{Identifier} \times Value \times \text{Context} \to State$
$write(s,I,v)^c = \langle store(s)[[\![I]\!]^c \mapsto v], control(s)\rangle$

$writes(s,I_1,v_1,\ldots,I_n,v_n)^c \equiv write(\ldots write(s,I_1,v_1)^c \ldots,I_n,v_n)^c$

$s_0 \text{ EQUALS}^c\ s_1\ \equiv$
$\quad \forall I \in \text{Identifier} : read(s_0,I)^c = read(s_1,I)^c$

$s_0 = s_1 \text{ EXCEPT}^c\ I_1,\ldots,I_n\ \equiv$
$\quad \forall I \in \text{Identifier} : I \neq I_1 \wedge \ldots \wedge I \neq I_n\ \Rightarrow$
$\qquad read(s_0,I)^c = read(s_1,I)^c$

$s_0 \text{ EQUALS}_c^{c'}\ s_1\ \equiv$
$\quad \forall I \in \text{Identifier} : read(s_0,I)^c = read(s_1,I)^{c'}$

$s_0 = s_1 \text{ EXCEPT}_c^{c'}\ I_1,\ldots,I_n\ \equiv$
$\quad \forall I \in \text{Identifier} : I \neq I_1 \wedge \ldots \wedge I \neq I_n\ \Rightarrow$
$\qquad read(s_0,I)^c = read(s_1,I)^{c'}$

$s_0 = s_1 \text{ EXCEPT } V \equiv$
$\quad \forall v \in Variable : store(s_0)(v) \neq store(s_1)(v) \Rightarrow v \in V$

Figure 6.3: Contexts and States

$$\forall c \in \mathit{Context}, I \in \text{Identifier}:\\ \mathit{DifferentVariables}(c) \Rightarrow \mathit{DifferentVariables}(\mathit{push}(c,I))$$

Proof Immediate from the definitions of *DifferentVariables* and *push*. □

Furthermore, the function *push* narrows the range of a context as stated by the following lemma.

Lemma (Range of Contexts) *push* removes from the range the variable currently denoted by the variable pushed:

$$\forall c \in \mathit{Context}, I \in \text{Identifier}:\\ \mathit{range}(\mathit{push}(c,I)) = \mathit{range}(c) \setminus \{[\![I]\!]^c\}$$

Proof Immediate from the definitions of *range* and *push*. □

The context created by *push* creates a new identifier assignment as stated by the following lemma.

Lemma (Identifiers and Declarations 1) The variable assigned to an identifier by a local declaration is different from the variable assigned to any identifier outside the declaration:

$$\forall c \in \mathit{Context}, I, J \in \text{Identifier}:\\ \mathit{DifferentVariables}(c) \Rightarrow [\![I]\!]^{\mathit{push}(c,I)} \neq [\![J]\!]^c$$

Proof Take arbitrary $c \in \mathit{Context}, I, J \in \text{Identifier}$. We assume

(1) $\mathit{DifferentVariables}(c)$

and show

(a) $[\![I]\!]^{\mathit{push}(c,I)} \neq [\![J]\!]^c$

From the definition of *Context*, we know for some $v \in \mathit{View}$ and $s \in \mathit{Space}$

(2) $c = \langle v, s \rangle$

From the definition of *push*, we know for some $x \in \mathit{Variable}$

(3) $x \in s$

(4) $push(c,I) = \langle v[I \mapsto x], s \backslash \{x\} \rangle$

From (4) and the definition of $[\![\_]\!]$, we know

(5) $[\![I]\!]^{push(c,I)} = x$

From (1), (2), and the definition of *DifferentVariables*, we know

(6) $[\![J]\!]^c \notin s$

From (3), (5), and (6), we know (a). □

Lemma (Identifiers and Declarations 2) The variable assigned to an identifier by a local declaration is different from the variable assigned to any identifier outside the declaration:

$$\forall c \in Context, I, J \in \text{Identifier}: \\ DifferentVariables(c) \wedge I \neq J \Rightarrow [\![J]\!]^{push(c,I)} = [\![J]\!]^c$$

Proof Take arbitrary $c \in Context, I, J \in$ Identifier and assume

(1) $DifferentVariables(c)$

(2) $I \neq J$

We have to show

(a) $[\![J]\!]^{push(c,I)} = [\![J]\!]^c$

From the definition of *take*, there exists $x \in Variable, s \in Space$ such that

(3) $\langle x, s \rangle := take(space(c))$

From (3) and the definitions of $[\![\_]\!]$ and *push*, to show (a), it suffices to show

(b) $view(c)[I \mapsto x](J) = view(c)(J)$

which by (2) is true. □

In a local declaration, a virtually store with identical values for the variables assigned to identifiers can be created as demonstrated by the following lemma.

Lemma (Writing and Declarations) Writing into a locally declared variable that value that the variable had outside the declaration yields indistinguishable stores:

$$\begin{array}{l}\forall c \in \mathit{Context}, s \in \mathit{State}, I \in \text{Identifier}: \\ \quad \mathit{DifferentVariables}(c) \Rightarrow \\ \quad\quad \text{LET } v = \mathit{read}(s,I)^c, c' = \mathit{push}(c,I), s' = \mathit{write}(s,I,v)^{c'} \text{ IN} \\ \quad\quad s \text{ EQUALS}_c^{c'} \ s'\end{array}$$

Proof Take arbitrary $c \in \mathit{Context}, s \in \mathit{State}, I \in$ Identifier and assume

(1) $\mathit{DifferentVariables}(c)$

We define

(2) $v := \mathit{read}(s,I)^c$

(3) $c' := \mathit{push}(c,I)$

(4) $s' := \mathit{write}(s,I,v)^{c'}$

We take arbitrary $J \in \mathit{Identifier}$ and show

(a) $\mathit{read}(s,J)^c = \mathit{read}(s',J)^{c'}$

We proceed by case distinction.

In the first case, we assume

(5) $I = J$

By (2), (3), (4), (5) and the definition of *read*, to show (a), it suffices to show

(b) $s([\![I]\!]^c) = s[[\![I]\!]^{\mathit{push}(c,I)} \mapsto s([\![I]\!]^c)]([\![I]\!]^{\mathit{push}(c,I)})$

which is obviously true.

In the second case, we assume

(6) $I \neq J$

By (2), (3), (4), and the definition of *read*, to show (a), it suffices to show

(b) $s([\![J]\!]^c) = s[[\![I]\!]^{\mathit{push}(c,I)} \mapsto s([\![I]\!]^c)]([\![J]\!]^{\mathit{push}(c,I)})$

From (1), (6) and Lemma "Identifiers and Declarations 2", we have

(7) $[\![J]\!]^c = [\![J]\!]^{push(c,I)}$

From (1), (6), Lemma "Soundness of Contexts", and finally the definition of *DifferentVariables*, we know

(8) $[\![I]\!]^{push(c,I)} \neq [\![J]\!]^{push(c,I)}$

From (7), and (8), we know (b). □

Changes made with respect to a new identifier assignment are forgotten when the assignment is forgotten, as stated by the following lemma.

Lemma (States and Declarations 0) Changing the value of a locally declared variable has no visible effect outside the declaration:

$$\begin{array}{l}\forall c \in \mathit{Context}, s \in \mathit{State}, I \in \text{Identifier}, v \in \text{Value}: \\ \quad \mathit{DifferentVariables}(c) \Rightarrow \\ \qquad s \text{ EQUALS}^c \ \mathit{write}(s, I, v)^{push(c,I)}\end{array}$$

Proof Take arbitrary $c \in \mathit{Context}, s \in \mathit{State}, I \in \text{Identifier}, v \in \text{Value}$ and assume

(1) $\mathit{DifferentVariables}(c)$

and show

(a) $s \text{ EQUALS}^c \ \mathit{write}(s, I, v)^{push(c,I)}$

To show (2), by the definitions of EQUALS and *read*, it suffices to show

(b) $\forall J \in \text{Identifier}: s([\![J]\!]^c) = s[[\![I]\!]^{push(c,I)} \mapsto v]([\![J]\!]^c)$

We take arbitrary $J \in \text{Identifier}$ and show

(c) $s([\![J]\!]^c) = s[[\![I]\!]^{push(c,I)} \mapsto v]([\![J]\!]^c)$

This is a consequence of (1) and Lemma "Identifiers and Declarations". □

Furthermore, the effect of a variable declaration on the view is restricted as stated by the following three lemmas.

Lemma (States and Declarations 1) By a local declaration, only the view to this variable is changed:

$$\begin{array}{l}\forall c \in \mathit{Context}, s \in \mathit{State}, I \in \text{Identifier}: \\ \quad \mathit{DifferentVariables}(c) \Rightarrow \\ \quad\quad s = s \text{ EXCEPT}_c^{\mathit{push}(c,I)} \ I\end{array}$$

Proof We take arbitrary $c \in \mathit{Context}, s \in \mathit{State}, I \in \text{Identifier}$ and assume

(1) $\mathit{DifferentVariables}(c)$

We show

(a) $s = s \text{ EXCEPT}_c^{\mathit{push}(c,I)} \ I$

By the definition of EXCEPT, it suffices to show

(b) $\forall J \in \text{Identifier}: J \neq I \Rightarrow \mathit{read}(s,J)^c = \mathit{read}(s,J)^{\mathit{push}(c,I)}$

From (1), Lemma "Identifiers and Declarations 2", and the definition of *read*, we know (b). □

Lemma (States and Declarations 2) A local declaration does not change the views on the identifiers not affected by the declaration:

$$\begin{array}{l}\forall c \in \mathit{Context}, s_0, s_1 \in \mathit{State}, I \in \text{Identifier}: \\ \quad \mathit{DifferentVariables}(c) \Rightarrow \\ \quad\quad s_0 = s_1 \text{ EXCEPT}^c \ I \Rightarrow s_0 = s_1 \text{ EXCEPT}^{\mathit{push}(c,I)} \ I\end{array}$$

Proof Take arbitrary $c \in \mathit{Context}, s_0, s_1 \in \mathit{State}, I \in \text{Identifier}$. We assume

(1) $\mathit{DifferentVariables}(c)$

(2) $s_0 = s_1 \text{ EXCEPT}^c \ I$

and show

(a) $s_0 = s_1 \text{ EXCEPT}^{\mathit{push}(c,I)} \ I$

From (1) and Lemma "Soundness of Contexts", we know

(3) $\mathit{DifferentVariables}(\mathit{push}(c,I))$

To show (a), it suffices to show

(b) $\forall J \in \text{Identifier} : J \neq I \Rightarrow$
$\quad read(s_0, J)^{push(c,I)} = read(s_1, J)^{push(c,I)}$

Take arbitrary $J \in$ Identifier and assume

(4) $J \neq I$

From the definition of *read*, to show (b), it suffices to show

(c) $s_0([\![J]\!]^{push(c,I)}) = s_1([\![J]\!]^{push(c,I)})$

From the definition of $[\![\_]\!]$ and *push*, it suffices to show for arbitrary $x \in space(c)$

(d) $s_0(view(c)[I \mapsto x](J)) = s_1(view(c)[I \mapsto x](J))$

From (2), we know

(5) $s_0(view(c)(J)) = s_1(view(c)(J))$

From (4) and (5), we know (d). □

Lemma (States and Declarations 3) When leaving the scope of a declaration, the view on the identifiers is unchanged except for the locally declared variable:

$\forall c \in Context, s_0, s_1 \in State, I \in \text{Identifier} :$
$\quad DifferentVariables(c) \Rightarrow$
$\quad\quad s_0 = s_1 \text{ EXCEPT}^{push(c,I)} \; I \Rightarrow s_0 = s_1 \text{ EXCEPT}^{c} \; I$

Proof Take arbitrary $c \in Context, s_0, s_1 \in State, I \in$ Identifier. We assume

(1) $DifferentVariables(c)$

(2) $s_0 = s_1 \text{ EXCEPT}^{push(c,I)} \; I$

and show

(a) $s_0 = s_1 \text{ EXCEPT}^{c} \; I$

From (1) and Lemma "Soundness of Contexts", we know

(3) $DifferentVariables(push(c, I))$

To show (a), it suffices to show

(b) $\forall J \in \text{Identifier} : J \neq I \Rightarrow read(s_0, J)^c = read(s_1, J)^c$

Take arbitrary $J \in$ Identifier and assume

(4) $J \neq I$

From the definition of *read*, to show (b), it suffices to show

(c) $s_0([\![J]\!]^c) = s_1([\![J]\!]^c)$

From the definition of $[\![\_]\!]$, it suffices to show

(d) $s_0(view(c)(J)) = s_1(view(c)(J))$

From (2), (4), and the definition of *push*, we know for some $x \in space(c)$

(5) $s_0(view(c)[I \mapsto x](J)) = s_1(view(c)[I \mapsto x](J))$

From (4) and (5), we know (d). □

With these provisions, we can modify the semantics of the command language as shown in Figure 6.4. The context c becomes an additional argument to all valuation functions and is forwarded from one valuation function to the next. Syntactically this means in most cases just the transformation of applications $[\![\_]\!]$ to $[\![\_]\!]^c$; we omit a repetition of these.

The only case where the context actually plays a role are (apart from the meaning of identifiers in expressions already given in Figure 6.2), the definitions in Figure 6.4 that determine the semantics of variable assignments, local variable declarations and definitions, and exception handlers. The valuation of local variable declarations/definitions demonstrates the core purpose of contexts: from the current context c, a new context c' is generated in which the body of the command is evaluated; the poststore of the body itself is the poststore of the block *without* restoring any variable value.

Figure 6.5 generalizes the valuation functions of formulas and terms correspondingly; here the only rules of relevance are those for the prestate/poststate values of variables. The semantics of formulas does not change the meaning of formulas as stated by the following lemma.

Programs with Contexts: Valuation Functions

$[\![\_]\!] : \textbf{Program} \to \textit{Context} \to \textit{StateRelation}$
$[\![C]\!]^c = [\![C]\!]^c$

$[\![\_]\!] : \textbf{Command} \to \textit{Context} \to \textit{StateRelation}$
$[\![I = E]\!]^c(s,s') \Leftrightarrow$
  $s' = write(s, I, [\![E]\!]^c(s))^c$
$[\![\texttt{var } I\texttt{; } C]\!]^c(s,s') \Leftrightarrow$
  LET $c' = push(c,I)$ IN $[\![C]\!]^{c'}(s,s')$
$[\![\texttt{var } I\texttt{=}E\texttt{; } C]\!]^c(s,s') \Leftrightarrow$
  LET $c' = push(c,I), s_0 = write(s, I, [\![E]\!]^c(s))^{c'}$ IN $[\![C]\!]^{c'}(s_0,s')$
$[\![\texttt{try } C_1 \texttt{ catch (}I_k\ I_v\texttt{) } C_2]\!]^c(s,s') \Leftrightarrow$
  $\exists s_0, s_1 \in \textit{State}:$
    $[\![C_1]\!]^c(s,s_0) \wedge$
    IF $throws(control(s_0)) \wedge key(control(s_0)) = I_k$ THEN
      LET $c' = push(c, I_v)$ IN
      $s_1 = write(execute(s_0), I_v, value(control(s_0)))^{c'} \wedge$
      $[\![C_2]\!]^{c'}(s_1,s')$
    ELSE $s' = s_0$
$[\![\ldots]\!]^c(s,s') \Leftrightarrow \ldots$

$[\![\_]\!] : \textbf{Expression} \to \textit{Context} \to \textit{StateFunction}$
$[\![\ldots]\!]^c(s) = \ldots$

Programs with Contexts: Termination

$[\![\_]\!]_{\mathrm{T}} : \textbf{Command} \to \textit{Context} \to \textit{StateCondition}$
$[\![\ldots]\!]^c_{\mathrm{T}}(s) \Leftrightarrow \ldots$

Figure 6.4: A Context Language: Valuation Functions

Formulas and Terms with Contexts: Valuation Functions

$[\![\_]\!] : \textbf{Formula} \to \textit{Context} \to \textit{Environment} \to \textit{StateRelation}$
$[\![\ldots]\!]^c(e)(s,s') \Leftrightarrow \ldots$

$[\![\_]\!] : \textbf{Term} \to \textit{Context} \to \textit{Environment} \to \textit{BinaryStateFunction}$
$[\![I]\!]^c(e)(s,s') = \mathit{read}(s,I)^c$
$[\![I']\!]^c(e)(s,s') = \mathit{read}(s',I)^c$
$[\![\ldots]\!]^c(e)(s,s') = \ldots$

Figure 6.5: A Context Language: Formulas and Terms

Lemma (Formulas with Contexts) For the same mapping of identifiers to variables, the semantics of formulas with contexts is equivalent to the original one.

$$\begin{array}{l}\forall c \in \textit{Context} : (\forall I \in \text{Identifier} : [\![I]\!] = [\![I]\!]^c) \Rightarrow \\ \quad \forall F \in \text{Formula}, e \in \textit{Environment}, s, s' \in \textit{State} : \\ \quad\quad [\![F]\!](e)(s,s') \Leftrightarrow [\![F]\!]^c(e)(s,s')\end{array}$$

Proof A consequence of the semantics of identifiers with contexts. □

Lemma (Equal Stores/ESF') A formula preserves in different contexts and different stores its value provided that the variables denoted by identifiers and the control data remain the same:

$$\begin{array}{l}\forall F \in \text{Formula}, e \in \textit{Environment}, s_0, s_0', s_1, s_1' \in \textit{State}, c \in \textit{Context} : \\ \quad s_0 \ \text{EQUALS}_c^{c'} \ s_1 \wedge s_0' \ \text{EQUALS}_c^{c'} \ s_1' \wedge \\ \quad \mathit{control}(s_0) = \mathit{control}(s_1) \wedge \mathit{control}(s_0') = \mathit{control}(s_1') \Rightarrow \\ \quad\quad [\![F]\!]^c(e)(s_0,s_0') \Leftrightarrow [\![F]\!]^{c'}(e)(s_1,s_1')\end{array}$$

Proof A consequence of the semantics of identifiers with contexts. □

However, the semantics of a program with contexts is not necessarily equivalent to the original semantics, because in the former the program may modify variables in the space of the context that were not modified (even not accessible) in the original semantics. Nevertheless, their semantices are very closely related; for elaborating this relationship, we need a couple of results that are introduced below.

First, as well in the original semantics as in the semantics with contexts, commands cannot distinguish states apart from their control data and the contents of variables denoted by identifiers.

Lemma (Commands and States 1) Commands cannot distinguish between states that hold the same control data and the same values in all variables denoted by identifiers:

$$\begin{array}{l}\forall C \in \text{Command}, s_0, s_0', s_1, s_1' \in \mathit{State} : \\ \quad \mathit{control}(s_0) = \mathit{control}(s_1) \wedge s_0 \text{ EQUALS } s_1 \wedge \\ \quad \mathit{control}(s_0') = \mathit{control}(s_1') \wedge s_0' \text{ EQUALS } s_1' \Rightarrow \\ \quad\quad [\![C]\!](s_0, s_0') \Leftrightarrow [\![C]\!](s_1, s_1')\end{array}$$

Proof By induction on the structure of C. No command accesses other variables than those referenced by identifiers. □

Lemma (Commands and States 2) Commands cannot distinguish between states that hold the same control data and the same values in all variables denoted by identifiers in the view of the context:

$$\begin{array}{l}\forall C \in \text{Command}, s_0, s_0', s_1, s_1' \in \mathit{State}, c \in \mathit{Context} : \\ \quad \mathit{DifferentVariables}(c) \wedge \\ \quad \mathit{control}(s_0) = \mathit{control}(s_1) \wedge s_0 \text{ EQUALS}^c \ s_1 \wedge \\ \quad \mathit{control}(s_0') = \mathit{control}(s_1') \wedge s_0' \text{ EQUALS}^c \ s_1' \Rightarrow \\ \quad\quad [\![C]\!]^c(s_0, s_0') \Leftrightarrow [\![C]\!]^c(s_1, s_1')\end{array}$$

Proof By induction on the structure of C. No command accesses other variables than those referenced by identifiers. □

Second, the context of a command restricts the modifications that the command may make on the store as stated by the following lemma.

Lemma (Commands and Contexts) A command can only change variables that are in the range of its context:

$$\begin{array}{l}\forall C \in \text{Command}, c \in \mathit{Context}, s, s' \in \mathit{State} : \\ \quad \mathit{DifferentVariables}(c) \wedge [\![C]\!]^c(s, s') \Rightarrow s = s' \text{ EXCEPT } \mathit{range}(c)\end{array}$$

Proof From the fact that the only variables referenced in the command are the results of the application of some view and that all views in the command are derived from the current view by applications of *push* which, by Lemma "Range of Contexts" does not widen the range. □

Next, changes made by a command to a locally declared variable are not visible to the outer context as stated by the following lemma.

Lemma (Commands and Declarations 1) If a command changes a variable inside a local declaration, this change is not visible outside the declaration:

$$\begin{array}{l}\forall C \in \text{Command}, c \in \mathit{Context}, I \in \text{Identifier}, s, s' \in \mathit{State}: \\ \quad \mathit{DifferentVariables}(c) \wedge [\![C]\!]^{\mathit{push}(c,I)}(s,s') \Rightarrow \\ \quad\quad \mathit{read}(s,I)^c = \mathit{read}(s',I)^c\end{array}$$

Proof Take arbitrary $C \in \text{Command}, c \in \mathit{Context}, J \in \text{Identifier}, s, s' \in \mathit{State}$ and assume

(1) $\mathit{DifferentVariables}(c)$

(2) $[\![C_0]\!]^{\mathit{push}(c,J)}(s,s')$

From (1) and Lemma "Soundness of Contexts", we also know

(3) $\mathit{DifferentVariables}(\mathit{push}(c,J))$

We show

(a) $\mathit{read}(s,J)^c = \mathit{read}(s',J)^c$

From the definition of *read*, to show (a), it suffices to show

(b) $\mathit{store}(s)([\![J]\!]^c) = \mathit{store}(s')([\![J]\!]^c)$

The proof proceeds by induction on the structure of C_0. We focus on the relevant cases of assignments, local variable definitions/declarations, and exception handlers.

- Case $C_0 = I{=}E$:

  From (2) and the definition of $[\![\_]\!]$, we know

  (4) $s' = \mathit{write}(s, I, [\![E]\!]^{\mathit{push}(c,J)}(s))^{\mathit{push}(c,J)}$

From (4) and the definition of *write*, we know

(6) $store(s') = store(s)[[\![I]\!]^{push(c,J)} \mapsto [\![E]\!]^{push(c,J)}(s)]$

From (1) and Lemma "Identifiers and Declarations 1", we know

(7) $[\![I]\!]^{push(c,I)} \neq [\![J]\!]^{c}$

From (6) and (7), we know (b).

- Case $C_0 = \texttt{var}\ I\texttt{;}\ C$: From (2) and the definition of $[\![\_]\!]$, we know

  (4) $[\![C]\!]^{push(push(c,J),I)}(s,s')$

  From (3) and Lemma "Soundness of Contexts", we also know

  (5) $DifferentVariables(push(push(c,J),I))$

  From (4), (5) and Lemma "Commands and Contexts", to show (b), it suffices to show

  (c) $[\![J]\!]^{c} \notin range(push(push(c,J),I))$

  From Lemma "Range of Contexts", we know

  (6) $range(push(push(c,J),I)) = range(c) \setminus \{[\![J]\!]^{c}, [\![I]\!]^{push(c,J)}\}$

  From (6), we know (c).

- Case $C_0 = \texttt{var}\ I\texttt{=}E\texttt{;}\ C$: we define

  (4) $s_0 := write(s,I,[\![E]\!]^{c}(s))^{push(c,I)}$

  From (2), (4), and the definition of $[\![\_]\!]$, we know

  (4) $[\![C]\!]^{push(push(c,J),I)}(s_0,s')$

  From (3) and Lemma "Soundness of Contexts", we also know

  (5) $DifferentVariables(push(push(c,J),I))$

  From Lemma "Range of Contexts", we know

  (6) $range(push(push(c,J),I)) = range(c) \setminus \{[\![J]\!]^{c}, [\![I]\!]^{push(c,J)}\}$

  From (6), we know

  (7) $[\![J]\!]^{c} \notin range(push(push(c,J),I))$

  From (4), (5), (7), and Lemma "Commands and Contexts", we know

  (8) $store(s_0)([\![J]\!]^{c}) = store(s')([\![J]\!]^{c})$

  From (4) and the definition of *write*, we know

(9) $store(s_0) = store(s)[[\![I]\!]^{push(c,I)} \mapsto [\![E]\!]^c(s)]$

From (1) and Lemma "Identifiers and Declarations 1", we know

(10) $[\![J]\!]^c \neq [\![I]\!]^{push(c,I)}$

From (8), (9), and (10), we know (b).

- Case $C_0 = \texttt{try}\ C_1\ \texttt{catch}\,(I_k\ I_v)\ C_2$: from (2) and the definition of $[\![\_]\!]$, we know for some $s_0, s_1 \in State$:

(4) $[\![C_1]\!]^{push(c,J)}(s, s_0)$
IF $throws(control(s_0)) \wedge key(control(s_0)) = I_k$ THEN
LET $c' = push(push(c,J), I_v)$ IN
(5) $s_1 = write(execute(s_0), I_v, value(control(s_0)))^{c'} \wedge$
$[\![C_2]\!]^{c'}(s_1, s')$
ELSE $s' = s_0$

From (3), (4), the induction hypothesis, and the definition of *read*, we know

(6) $store(s)([\![J]\!]^c) = store(s_0)([\![J]\!]^c)$

If $\neg(throws(control(s_0)) \wedge key(control(s_0)) = I_k)$, (5) and (6) imply (a).

We may thus proceed with the assumptions

(7) $throws(control(s_0))$
(8) $key(control(s_0)) = I_k$
(9) $s_1 = write(execute(s_0), I_v, value(control(s_0)))^{push(push(c,J),I_v)}$
(10) $[\![C_2]\!]^{push(push(c,J),I_v)}(s_1, s')$

From (3) and Lemma "Soundness of Contexts", we know

(11) $DifferentVariables(push(push(c,J), I_v))$

From Lemma "Range of Contexts", we know

(12) $range(push(push(c,J), I_v)) = range(c) \setminus \{[\![J]\!]^c, [\![I]\!]^{push(c,J)}\}$

From (12), we know

(13) $[\![J]\!]^c \notin range(push(push(c,J), I_v))$

From (10), (11), (13), and Lemma "Commands and Contexts", we know

(14) $store(s_1)([\![J]\!]^c) = store(s')([\![J]\!]^c)$

From (9) and the definitions of *write* and *execute*, we know

(15) $store(s_1) = store(s_0)[[\![I_v]\!]^{push(c,I_v)} \mapsto value(control(s_0))]$

Simulation of State Relations

$$\begin{array}{l}
\llcorner \sim^{\llcorner} \llcorner \subseteq \mathit{StateRelation} \times \mathit{Context} \times \mathit{StateRelation} \\
R \sim^{c} S \Leftrightarrow \\
\quad \forall s, s' \in \mathit{State} : \mathit{executes}(\mathit{control}(s)) \wedge R(s, s') \Rightarrow \\
\quad\quad \exists s_0, s_1 \in \mathit{State} : S(s_0, s_1) \wedge \\
\quad\quad\quad s_0 \text{ EQUALS}^{c}\ s \wedge \mathit{control}(s_0) = \mathit{control}(s) \wedge \\
\quad\quad\quad s_1 \text{ EQUALS}^{c}\ s' \wedge \mathit{control}(s_1) = \mathit{control}(s')
\end{array}$$

Figure 6.6: Simulation of State Relations

From (1) and Lemma "Identifiers and Declarations 1", we know

(16) $[\![J]\!]^{c} \neq [\![I_v]\!]^{\mathit{push}(c, I_v)}$

From (6), (14), (15), and (16), we know (b). □

The following lemma shows how may create in a local declaration context stores that let a command behave like outside the declaration.

Lemma (Commands and Declarations 2) In a local declaration, a command behaves like it does outside declaration, if the stores in the declaration is updated such that the locally declared variable has the same value as the variable had outside the declaration.

$$\begin{array}{l}
\forall c \in \mathit{Context}, I \in \mathit{Identifier}, C \in \mathit{Command}, s', s' \in \mathit{Store} : \\
\quad \mathit{DifferentVariables}(c) \Rightarrow \\
\quad\quad \text{LET } c' = \mathit{push}(c, I) \text{ IN} \\
\quad\quad\quad [\![C]\!]^{c}(s, s') \Leftrightarrow \\
\quad\quad\quad [\![C]\!]^{c'}(\mathit{write}(s, I, \mathit{read}(s, I)^{c})^{c'}, \mathit{write}(s', I, \mathit{read}(s', I)^{c})^{c'})
\end{array}$$

Proof The proof proceeds by induction on the structure of C; it is essentially a consequence of Lemma "Writing and Declarations". □

Figure 6.6 introduces (for every context c) a relation $\sim^{c}$ between state relations that is weaker than complete equivalence: $R \sim^{c} S$ holds, if $R(s, s')$ implies $S(s_0, s_1)$ where state s_0 is identical to s and state s_1 is identical to s_1 with respect to their control states and the view of c (the states may differ in the storage contents of the variables outside the view). Based on this definition, we can state the actual relationship of the semantics of programs with contexts to the original semantics.

Lemma (Programs with Contexts) For the same mapping of identifiers to variables, the semantics of programs with contexts simulates the basic semantics and vice versa:

$$\begin{array}{l}\forall C \in \text{Command}, c \in \mathit{Context}: \\ \quad \mathit{DifferentVariables}(c) \wedge (\forall I \in \text{Identifier}: [\![I]\!] = [\![I]\!]^c) \Rightarrow \\ \quad\quad [\![C]\!] \sim^c [\![C]\!]^c \wedge [\![C]\!]^c \sim^c [\![C]\!]\end{array}$$

Proof Take arbitrary $C_0 \in \text{Command}, c \in \mathit{Context}$ and assume

(1) $\mathit{DifferentVariables}(c)$

(2) $\forall I \in \text{Identifier}: [\![I]\!] = [\![I]\!]^c$

From (1) and (2), we also have

(3) $\mathit{DifferentVariables}$

We have to show for arbitrary $s, s' \in \mathit{State}$

(a.1) $[\![C_0]\!](s,s') \sim^c [\![C_0]\!]^c(s,s')$

(a.2) $[\![C_0]\!]^c(s,s') \sim [\![C_0]\!](s,s')$

The proof proceeds by induction on the structure of C_0. The correctness is obvious for all the cases without structural change of the definition of the valuation function. The cases for assignments, identifiers in expressions, and pre/post-state variables in terms follow directly from the definitions. In the following, we restrict our consideration to local variable declarations; the cases of local variable definitions and exception handlers are handled in a similar way.

Case $C_0 = \texttt{var}\ I\texttt{;}\ C$ To show (a.1), we assume for some $s_0, s_1 \in \mathit{State}$

(4) $\mathit{executes}(\mathit{control}(s))$

(5) $s_0 = s$ EXCEPT I

(6) $\mathit{control}(s_0) = \mathit{control}(s)$

(7) $[\![C]\!](s_0, s_1)$

(8) $s' = \mathit{write}(s_1, I, \mathit{read}(s, I))$

and show

(a.1.a)
$$\begin{array}{l}\exists s_2, s_3 \in \mathit{State}: \\ \quad \text{LET } c' = \mathit{push}(c, I) \text{ IN } [\![C]\!]^{c'}(s_2, s_3) \wedge \\ s_2 \text{ EQUALS}^c\ s \wedge \mathit{control}(s_2) = \mathit{control}(s) \wedge \\ s_3 \text{ EQUALS}^c\ s' \wedge \mathit{control}(s_3) = \mathit{control}(s')\end{array}$$

From (4), (7) and the induction hypothesis, we know for some $s_4, s_5 \in \mathit{State}$

(9) $[\![C]\!]^c(s_4, s_5)$

(10) $s_4 \text{ EQUALS}^c\ s_0$

(11) $\mathit{control}(s_4) = \mathit{control}(s_0)$

(12) $s_5 \text{ EQUALS}^c\ s_1$

(13) $\mathit{control}(s_5) = \mathit{control}(s_1)$

From (2), (9), (12), and the definition of EQUALS, we know

(14) $s_4 \text{ EQUALS } s_0$

(15) $s_5 \text{ EQUALS } s_1$

We define

(16) $c' := \mathit{push}(c, I)$

(17) $s_2 := \mathit{write}(s, I, \mathit{read}(s_0, I)^c)^{c'}$

(18) $s_3 := \mathit{write}(s', I, \mathit{read}(s_1, I)^c)^{c'}$

From (1), (16), and Lemma "Soundness of Contexts", we know

(19) $\mathit{DifferentVariables}(c')$

To show (a.1.a), it suffices to show

(a.1.b.1) $[\![C]\!]^{c'}(s_2, s_3)$

(a.1.b.2) $s_2 \text{ EQUALS}^c\ s$

(a.1.b.3) $\mathit{control}(s_2) = \mathit{control}(s)$

(a.1.b.4) $s_3 \text{ EQUALS}^c\ s'$

(a.1.b.5) $\mathit{control}(s_3) = \mathit{control}(s')$

From (1), (17), and Lemma "States and Declarations 0", we know (a.1.b.2).

From (17) and (CW), we know (a.1.b.3).

From (1), (18), and Lemma "States and Declarations 0", we know (a.1.b.4).

From (18) and (CW), we know (a.1.b.5).

From (1), (9) and Lemma "Commands and Declarations 2", we know

(20) $[\![C]\!]^{c'}(write(s_4,I,read(s_4,I)^c)^{c'},write(s_5,I,read(s_5,I)^c)^{c'})$

From (19), (20), and Lemma "Commands and States 2", to show (a.1.b.1), it suffices to show

(a.1.b.1.a.1) $control(s_2) = control(write(s_4,I,read(s_4,I)^c)^{c'})$

(a.1.b.1.a.2) s_2 EQUALS$^{c'}$ $write(s_4,I,read(s_4,I)^c)^{c'}$

(a.1.b.1.a.3) $control(s_3) = control(write(s_5,I,read(s_5,I)^c)^{c'})$

(a.1.b.1.a.4) s_3 EQUALS$^{c'}$ $write(s_5,I,read(s_5,I)^c)^{c'}$

From (6), (11), (17), and (CW), we know (a.1.b.1.a.1).

To show (a.1.b.1.a.2), from (RVE) and (NEQ), it suffices to show

(a.1.b.1.a.2.a.1) $s_2 = write(s_4,I,read(s_4,I)^c)^{c'}$ EXCEPT$^{c'}$ I

(a.1.b.1.a.2.a.2) $read(s_2,I)^{c'} = read(write(s_4,I,read(s_4,I)^c)^{c'},I)^{c'}$

To show (a.1.b.1.a.2.a.1), from (19), (WS), (TRE), it suffices to show

(a.1.b.1.a.2.a.1.a) $s_2 = s_4$ EXCEPT$^{c'}$ I

To show (a.1.b.1.a.2.a.1.a), from (16), (19), and Lemma "States and Declarations 2", it suffices to show

(a.1.b.1.a.2.a.1.b) $s_2 = s_4$ EXCEPT^{c} I

From (2) and (5), we know

(21) $s_0 = s$ EXCEPT^{c} I

From (10), (21), (a.1.b.2), and (TRE), we know (a.1.b.1.a.2.a.1.b).

To show (a.1.b.1.a.2.a.2), from (17) and (RW1), it suffices to show

(a.1.b.1.a.2.a.2.a) $read(s_0,I)^c = read(s_4,I)^c$

From (10), (RSE), and (NEQ), we know (a.1.b.1.a.2.a.2.a).

From (8), (13), (18), and (CW), we know (a.1.b.1.a.3).

To show (a.1.b.1.a.4), from (RVE) and (NEQ), it suffices to show

(a.1.b.1.a.4.a.1) $s_3 = write(s_5, I, read(s_5, I)^c)^{c'}$ EXCEPT$^{c'}$ I

(a.1.b.1.a.4.a.2) $read(s_3, I)^{c'} = read(write(s_5, I, read(s_5, I)^c)^{c'}, I)^{c'}$

To show (a.1.b.1.a.4.a.1), from (19), (WS), (TRE), it suffices to show

(a.1.b.1.a.4.a.1.a) $s_3 = s_5$ EXCEPT$^{c'}$ I

To show (a.1.b.1.a.4.a.1.a), from (16), (19), and Lemma "States and Declarations 2", it suffices to show

(a.1.b.1.a.4.a.1.b) $s_3 = s_5$ EXCEPT^{c} I

From (1), (2), (8), and (WS), we know

(22) $s_1 = s'$ EXCEPT^{c} I

From (12), (NEQ), and (AVE), we know

(23) $s_5 = s_1$ EXCEPT^{c} I

From (18) and (RW1), we know

(24) $s_3 = s'$ EXCEPT$^{c'}$ I

From (16), (19), (24), and Lemma "States and Declarations 3", we know

(25) $s_3 = s'$ EXCEPT^{c} I

From (22), (23), (25), and (TRE), we know (a.1.b.1.a.4.a.1.b).

To show (a.1.b.1.a.4.a.2), from (18) and (RW1), it suffices to show

(a.1.b.1.a.4.a.2.a) $read(s_1, I)^c = read(s_5, I)^c$

From (12), (RSE), and (NEQ), we know (a.1.b.1.a.4.a.2.a).

To show (a.2), we define

(4) $c' = push(c, I)$

and assume

(5) $\mathit{executes}(\mathit{control}(s))$

(6) $[\![C]\!]^{c'}(s,s')$

From (1), (4), and Lemma "Soundness of Contexts", we know

(7) $\mathit{DifferentVariables}(c')$

We have to show

(a.2.a)
$$\begin{array}{l}\exists s_0,s_1,s_2,s_3 \in \mathit{State}: \\ \quad s_2 = s_0 \text{ EXCEPT } I \wedge \mathit{control}(s_2) = \mathit{control}(s_0) \wedge \\ \quad [\![C]\!](s_2,s_3) \wedge \\ \quad s_1 = \mathit{write}(s_3,I,\mathit{read}(s_0,I)) \wedge \\ \quad s_0 \text{ EQUALS}^c\ s \wedge \mathit{control}(s_0) = \mathit{control}(s) \wedge \\ \quad s_1 \text{ EQUALS}^c\ s' \wedge \mathit{control}(s_1) = \mathit{control}(s')\end{array}$$

From (1), (2), (5), (6), and the induction hypothesis, we know for some $s_2,s_3 \in \mathit{State}$

(8) $[\![C]\!](s_2,s_3)$

(9) $s_2 \text{ EQUALS}^c\ s$

(10) $\mathit{control}(s_2) = \mathit{control}(s)$

(11) $s_3 \text{ EQUALS}^c\ s'$

(12) $\mathit{control}(s_3) = \mathit{control}(s')$

We define

(13) $s_0 := s_2$

(14) $s_1 := \mathit{write}(s_3,I,\mathit{read}(s_0,I))$

To show (a.2.a), from (8), it suffices to show

(a.2.b.1) $s_2 = s_0 \text{ EXCEPT } I$

(a.2.b.2) $\mathit{control}(s_2) = \mathit{control}(s_0)$

(a.2.b.3) $s_0 \text{ EQUALS}^c\ s$

(a.2.b.4) $\mathit{control}(s_0) = \mathit{control}(s)$

(a.2.b.5) $s_1 \text{ EQUALS}^c\ s'$

(a.2.b.6) $control(s_1) = control(s')$

From (13) and (RE), we know (a.2.b.1).

From (13) we know (a.2.b.2).

From (9) and (13), we know (a.2.b.3).

From (10) and (13), we know (a.2.b.4).

From (2), (14), and the definition of EQUALS, we know

(15) $read(s_1, I)^c = read(s_0, I)^c$

(16) $s_1 = s_3$ EXCEPT^c I

From (11), (16), and the definition of EQUALS, we know

(17) $s_1 = s'$ EXCEPT^c I

From (15) and (17), to show (a.2.b.5), it suffices to show

(a.2.b.5.a) $read(s_0, I)^c = read(s', I)^c$

From (1), (4), (6), and Lemma "Commands and Declarations 1", we know finally (a.2.b.5.a).

From (12), (14), and (CW), we know (a.2.b.6). □

Based on this result, we get the following crucial relationship between commands and specifications in both semantices.

Lemma (Specifications of Programs with Contexts) For the same mapping of identifiers to variables, programs with contexts satisfy the same specifications as the programs in the basic semantics:

$$
\begin{array}{l}
\forall C \in \text{Command}, F \in \text{Formula}, c \in \mathit{Context}: \\
\quad \mathit{DifferentVariables}(c) \wedge (\forall I \in \text{Identifier}: [\![I]\!] = [\![I]\!]^c) \Rightarrow \\
\qquad (\forall s, s' \in \mathit{State}, e \in \mathit{Environment}: \\
\qquad\quad \mathit{executes}(\mathit{control}(s)) \wedge [\![C]\!](s, s') \Rightarrow [\![F]\!](e)(s, s')) \Leftrightarrow \\
\qquad (\forall s, s' \in \mathit{State}, e \in \mathit{Environment}: \\
\qquad\quad \mathit{executes}(\mathit{control}(s)) \wedge [\![C]\!]^c(s, s') \Rightarrow [\![F]\!]^c(e)(s, s'))
\end{array}
$$

Proof Take arbitrary $C \in \text{Command}, F \in \text{Formula}, c \in \mathit{Context}$ and assume

(1) $\mathit{DifferentVariables}(c)$

(2) $\forall I \in \text{Identifier} : [\![I]\!] = [\![I]\!]^c$

From (1), (2), and Lemma "Programs with Contexts", we know

(3) $[\![C]\!] \sim^c [\![C]\!]^c$

(4) $[\![C]\!]^c \sim^c [\![C]\!]$

From (3), (4), and the definition of $\_ \sim^{\_} \_$, we know

(5)
$$\begin{array}{l}\forall s, s' \in \mathit{State} : \mathit{executes}(\mathit{control}(s)) \wedge [\![C]\!](s, s') \Rightarrow \\ \quad \exists s_0, s_1 \in \mathit{State} : [\![C]\!]^c(s_0, s_1) \wedge \\ \qquad s_0 \text{ EQUALS}^c\ s \wedge \mathit{control}(s_0) = \mathit{control}(s) \wedge \\ \qquad s_1 \text{ EQUALS}^c\ s' \wedge \mathit{control}(s_1) = \mathit{control}(s')\end{array}$$

(6)
$$\begin{array}{l}\forall s, s' \in \mathit{State} : \mathit{executes}(\mathit{control}(s)) \wedge [\![C]\!]^c(s, s') \Rightarrow \\ \quad \exists s'' \in \mathit{State} : [\![C]\!](s, s'') \wedge \\ \qquad s'' \text{ EQUALS}^c\ s' \wedge \mathit{control}(s'') = \mathit{control}(s')\end{array}$$

We have to show

(a)
$$\begin{array}{l}(\forall s, s' \in \mathit{State}, e \in \mathit{Environment} : \\ \quad \mathit{executes}(\mathit{control}(s)) \wedge [\![C]\!](s, s') \Rightarrow [\![F]\!](e)(s, s')) \Leftrightarrow \\ (\forall s, s' \in \mathit{State}, e \in \mathit{Environment} : \\ \quad \mathit{executes}(\mathit{control}(s)) \wedge [\![C]\!]^c(s, s') \Rightarrow [\![F]\!]^c(e)(s, s'))\end{array}$$

To show the $\Rightarrow$ direction of (a), we assume

(7)
$$\begin{array}{l}\forall s, s' \in \mathit{State}, e \in \mathit{Environment} : \\ \quad \mathit{executes}(\mathit{control}(s)) \wedge [\![C]\!](s, s') \Rightarrow [\![F]\!](e)(s, s')\end{array}$$

and show

(b)
$$\begin{array}{l}\forall s, s' \in \mathit{State}, e \in \mathit{Environment} : \\ \quad \mathit{executes}(\mathit{control}(s)) \wedge [\![C]\!]^c(s, s') \Rightarrow [\![F]\!]^c(e)(s, s')\end{array}$$

To show (b), we take arbitrary $s, s' \in \mathit{State}, e \in \mathit{Environment}$ and assume

(8) $\mathit{executes}(\mathit{control}(s)$

(9) $[\![C]\!]^c(s, s')$

and show

(c) $[\![F]\!]^c(e)(s,s')$

From (6), (8), and (9), we know for some $s_0, s_1 \in State$

(10) $[\![C]\!](s_0,s_1)$
(11) s_0 EQUALS^c s
(12) $control(s_0) = control(s)$
(13) s_1 EQUALS^c s'
(14) $control(s_1) = control(s')$

From (7), (8), and (10), we know

(15) $[\![F]\!](e)(s,s'')$

From (2), (11), (13), and the definition of EQUALS, we know

(16) s_0 EQUALS s
(17) s_1 EQUALS s'

From (12), (14), (15), (16), (17), and (ESF'), we know

(18) $[\![F]\!](e)(s,s')$

From (1), (2), (18), and Lemma "Formulas with Contexts", we know (c).
To show the $\Leftarrow$ direction of (a), we assume

(19) $\forall s,s' \in State, e \in Environment$:
$executes(control(s)) \wedge [\![C]\!]^c(s,s') \Rightarrow [\![F]\!]^c(e)(s,s')$

and show

(b) $\forall s,s' \in State, e \in Environment$:
$executes(control(s)) \wedge [\![C]\!](s,s') \Rightarrow [\![F]\!](e)(s,s')$

To show (b), we take arbitrary $s,s' \in State, e \in Environment$ and assume

(20) $executes(control(s)$
(21) $[\![C]\!](s,s')$

and show

(c) $[\![F]\!](e)(s,s')$

From (5), (20), and (21), we know for some $s_0, s_1 \in \mathit{State}$

(22) $[\![C]\!]^c(s_0, s_1)$

(23) s_0 EQUALS^c s

(24) $\mathit{control}(s_0) = \mathit{control}(s)$

(25) s_1 EQUALS^c s'

(26) $\mathit{control}(s_1) = \mathit{control}(s')$

From (7), (8), and (22), we know

(27) $[\![F]\!]^c(e)(s_0, s_1)$

From (1), (2), (27), and Lemma "Formulas with Contexts", we know

(28) $[\![F]\!](e)(s_0, s_1)$

From (2), (23), (25), and the definition of EQUALS, we know

(29) s_0 EQUALS s

(30) s_1 EQUALS s'

From (24), (26), (28), (29), (30), and (ESF'), we know (c). □

Based on the generalized definition of the semantics of formulas that takes contexts into account, Figure 6.7 gives the modified soundness claim for the three core judgements for commands. The correctness of the soundness claim for $C : F$ is a consequence of the correctness of the corresponding claim for the language without contexts and Lemma "Specifications of Programs with Contexts". In a similar fashion, also the correctness of the other claims can be derived.

In the following sections, we elaborate the semantics of programs with methods on the basis of the semantics of programs with contexts. For establishing the corresponding verification calculus, we take the soundness of the core judgements for commands with contexts as granted.

6.2 Method Declarations and Method Calls

Figure 6.8 extends the command language by the concept of *methods*. In this extended "method language", a program consists of a sequence of *method declarations Ms* and a *program body* (i.e. command) C which is executed in the environment established by the declarations. Every method declaration M consists of

Definitions

$[\![\_]\!] : \text{Formula} \rightarrow \mathit{Context} \rightarrow \mathit{StateCondition}$
$[\![F]\!]^c(s) \Leftrightarrow \forall e \in \text{Environment} : [\![F]\!]^c(e)(s,s)$

$[\![\_]\!] : \text{Formula} \rightarrow \mathit{StateCondition}$
$[\![F]\!](s) \Leftrightarrow \forall c \in \text{Context} : \mathit{DifferentVariables}(c) \Rightarrow [\![F]\!]^c(s)$

Judgements

$C \checkmark F \Leftrightarrow$
$\quad \forall c \in \mathit{Context} : \mathit{DifferentVariables}(c) \Rightarrow$
$\quad\quad \forall s, s' \in \mathit{State} : \mathit{executes}(\mathit{control}(s)) \wedge [\![F]\!]^c(s) \Rightarrow$
$\quad\quad\quad ([\![C]\!]^c(s,s') \Leftrightarrow [\![C]\!]^c_\perp(s,s'))$

$C : F \Leftrightarrow$
$\quad \forall c \in \mathit{Context} : \mathit{DifferentVariables}(c) \Rightarrow$
$\quad\quad \forall s, s' \in \mathit{State}, e \in \mathit{Environment} :$
$\quad\quad\quad \mathit{executes}(\mathit{control}(s)) \wedge [\![C]\!]^c(s,s') \Rightarrow [\![F]\!]^c(e)(s,s')$

$C \downarrow F \Leftrightarrow$
$\quad F$ has no free (mathematical or state) variables $\wedge$
$\quad F$ does not depend on the poststate $\Rightarrow$
$\quad\quad \forall c \in \mathit{Context} : \mathit{DifferentVariables}(c) \Rightarrow$
$\quad\quad\quad \forall s \in \mathit{State} :$
$\quad\quad\quad\quad \mathit{executes}(\mathit{control}(s)) \wedge [\![F]\!]^c(s) \Rightarrow [\![C]\!]^c_{\mathrm{T}}(s)$

Figure 6.7: A Context Language: Judgements

a *method header* $I_m(J_1,\ldots,J_p)$ and a *method body* (i.e. command) C. The method header introduces the name I_m of the method and a sequence of *formal parameters* $J_1,\ldots,J_p$ which denote those variables that hold the values of the actual arguments of a method call.

The program body as well as the method bodies are prefixed by a *specification* S that describes the effect of the respective command by

- a *frame* of identifiers $I_1,\ldots,I_n,?J_1,\ldots,?J_o$ that denotes the set of those variables whose values may be altered by the command (the identifier form $?J_i$ must be chosen if a parameter name J_i shadows the name of a variable),
- the exceptions $K_1,\ldots,K_m$ that may by thrown by the command,
- a formula F_C that denotes a state condition, and
- a formula F_R that denotes a state relation,
- a term T that denotes a measure for limiting the recursion depth,

where both F_C and F_R constrain the behavior of the command. Specifications will not be used for describing the semantics of programs but for reasoning about them, as will be explained later.

A command may now also be a *method call* $I_r = I_m\,(E_1,\ldots,E_p)$ which assigns to the formal parameters $J_1,\ldots,J_p$ of I_m of the method declarations the values of the actual arguments $E_1,\ldots,E_p$ of the method call, then executes the body of I_m, and finally delivers the return value generated by the command into the variable denoted by I_r. In the following, we assume that the number of arguments in method calls always equals the number of parameters in the declarations of the corresponding methods (this constraint can be established by a simple static checking mechanism).

The domain *MethodEnv* defined in Figure 6.8 denotes the set of environments established by method declarations. Such a "method environment" maps a method name to a view (the global variables visible by the method) and an element of domain *Behavior*; each such "method behavior" maps a context and a sequence of values (the arguments of a method call) to a pair of a state relation and a state condition (the transition relation and the termination condition of the method body executed with these arguments).

As shown by the valuation function for programs in Figure 6.9, a program is executed in a pre-established method environment which is updated by the method declarations in the program; the resulting environment is the one in which the

Method Language: Abstract Syntax

$Ms \in$ Methods
$M \in$ Method
$S \in$ Specification
$P \in$ Program
$C \in$ Command
$E \in$ Expression
$I, J, K \in$ Identifier
$F \in$ Formula

$P ::= Ms\ S\ \{C\}.$

$Ms ::= \llcorner \mid Ms\ M.$

$M ::= \texttt{method}\ I_m\ (J_1, \ldots, J_p)\ \ S\ \{C\}.$

$S ::= \texttt{writesonly}\ I_1, \ldots, I_n, \texttt{?}J_1, \ldots, \texttt{?}J_o$
$\texttt{throwsonly}\ K_1, \ldots, K_m\ \texttt{assumes}\ F_C\ \texttt{implements}\ F_R$
$\texttt{decreases}\ T$

$C ::= \ldots \mid I_r = I_m\ (E_1, \ldots, E_p)\ .$

Definitions

$Behavior := Context \times Value^* \to (StateRelation \times StateCondition)$
$MethodEnv :=$ Identifier $\to (View \times Behavior)$

$view : View \times Behavior \to View$
$view(v, b) = v$

$call : View \times Context \to Context$
$call(v, c') = context(v, space(c'))$

Figure 6.8: A Method Language (1/3)

Method Language: Valuation Functions

$$[\![\_]\!] : \textbf{Program} \to \textit{Context} \to \textit{MethodEnv} \to (\textit{StateRelation} \times \textit{StateCondition})$$
$$[\![Ms\ S\ \{C\}]\!]^c(me) =$$
$$\quad \text{LET } me' = [\![Ms]\!]^{view(c)}(me) \text{ IN } \langle [\![C]\!]^{c,me'}, [\![C]\!]_{\mathrm{T}}^{c,me'} \rangle$$

$$[\![\_]\!] : \textbf{Methods} \to \textit{View} \to \textit{MethodEnv} \to \textit{MethodEnv}$$
$$[\![\_]\!]^v(me) = me$$
$$[\![Ms\ M]\!]^v(me) = [\![M]\!]^v([\![Ms]\!]^v(me))$$

$$[\![\_]\!] : \textbf{Method} \to \textit{View} \to \textit{MethodEnv} \to \textit{MethodEnv}$$
$$[\![\texttt{method}\ I_m\ (J_1,\ldots,J_p)\ \ S\ \{C\}]\!]^v(me) =$$
$$\quad \text{LET}$$
$$\qquad b = [\![\texttt{method}\ I_m\ (J_1,\ldots,J_p)\ \ S\ \{C\}]\!](me)$$
$$\quad \text{IN } me[I_m \mapsto \langle v, b \rangle]$$

$$[\![\_]\!] : \textbf{Method} \to \textit{MethodEnv} \to \textit{Behavior}$$
$$[\![\texttt{method}\ I_m\ (J_1,\ldots,J_p)\ \ S\ \{C\}]\!](me) =$$
$$\quad \text{LET}$$
$$\qquad b : \textit{Behavior}$$
$$\qquad b^c(v_1,\ldots,v_p) =$$
$$\qquad\quad \text{LET}$$
$$\qquad\qquad c' = \textit{push}(c, J_1, \ldots, J_p)$$
$$\qquad\qquad r \in \textit{StateRelation}$$
$$\qquad\qquad r(s,s') \Leftrightarrow$$
$$\qquad\qquad\quad \exists s_0 : \textit{State} :$$
$$\qquad\qquad\qquad [\![C]\!]^{c',me}(\textit{writes}(s, J_1, v_1, \ldots, J_p, v_p)^{c'}, s_0) \wedge$$
$$\qquad\qquad\qquad s' = \text{IF } \textit{throws}(\textit{control}(s_0))$$
$$\qquad\qquad\qquad\qquad \text{THEN } s_0 \text{ ELSE } \textit{executes}(s_0)$$
$$\qquad\qquad t \in \textit{StateCondition}$$
$$\qquad\qquad t(s) \Leftrightarrow [\![C]\!]_{\mathrm{T}}^{c',me}(\textit{writes}(s, J_1, v_1, \ldots, J_p, v_p)^{c'})$$
$$\qquad\quad \text{IN } \langle r, t \rangle$$
$$\quad \text{IN } b$$

Figure 6.9: A Method Language (2/3)

Method Language: Valuation Functions (Contd)

$[\![\_]\!] : \textbf{Command} \to (\textit{Context} \times \textit{MethodEnv}) \to \textit{StateRelation}$
$[\![\ldots]\!]^{c,me}(s,s') \Leftrightarrow \ldots$
$[\![I_r = I_m\ (E_1,\ldots,E_p)\ ]\!]^{c,me}(s,s') \Leftrightarrow$
  LET $\langle v,b\rangle = me(I_m)$ IN
  LET $\langle r,t\rangle = b^{call(v,c)}([\![E_1]\!]^c(s),\ldots,[\![E_p]\!]^c(s))$ IN
  $\exists s_0 \in State : r(s,s_0) \wedge$
    IF $throws(control(s_0))$
      THEN $s' = s_0$
      ELSE $s' = write(s_0, I_r, value(control(s_0)))^c$

$[\![\_]\!]_{\mathrm{T}} : \textbf{Command} \to (\textit{Context} \times \textit{MethodEnv}) \to \textit{StateCondition}$
$[\![\ldots]\!]_{\mathrm{T}}^{c,me}(s) \Leftrightarrow \ldots$
$[\![I_r = I_m\ (E_1,\ldots,E_p)\ ]\!]_{\mathrm{T}}^{c,me}(s) \Leftrightarrow$
  LET $\langle v,b\rangle = me(I_m)$ IN
  LET $\langle r,t\rangle = b^{call(v,c)}([\![E_1]\!]^c(s),\ldots,[\![E_p]\!]^c(s))$ IN
  $t(s)$

Figure 6.10: A Method Language (3/3)

body of the program is executed. As shown by the valuation function for method declaration sequences, every declaration is evaluated in the environment established by the previous ones; the method declaration updates the environment by a new method behavior. Consequently method bodies can only call those methods that have been previously established; recursive or mutually recursive method calls are thus prohibited (this restriction will be relaxed later).

Since the declaration of a method takes place outside the scope of any call of that method, any local declaration in effect at the point of the call should have no effect on the execution of the method body (the principle of *static scoping*); the execution of the body thus must take place in a different context than that one in which the method call is executed. In more detail, the context of the method is constructed from

- the view that is in use at the point of the declaration (indicating the "global variables" of the method) and
- the space of that context that is in use at the point of the call (indicating the space that may be used for allocation of local variables)

The context of the method body is the context of the method updated by the declarations of the parameters of the method. Figures 6.9 and 6.10 give the valuation functions for method declarations and method calls that together establish a corresponding "context switching discipline".

The declaration of a method $I_m\,(J_1,\ldots,J_p)$ receives in the parameter v the view of the current context. It then constructs a method behavior b which receives the method context c and the argument values values $v_1,\ldots,v_p$ from the caller; from c the context c' of the method body is constructed by the declarations of the formal parameters $J_1,\ldots,J_p$. The declaration returns a new method environment that maps the method name I_m to the pair $\langle v,b\rangle$.

The method behavior b returns a pair pair $\langle r,t\rangle$ of a transition relation r and a termination condition t derived from the transition relation and the termination condition of the method body C. This body is executed in that state that is constructed from the prestate s of the method call by writing (in context c') into the variables denoted by identifiers $J_1,\ldots,J_p$ the argument values $v_1,\ldots,v_p$ of the call. If the poststate of C throws an exception, it also represents the poststate of the call; otherwise the poststate is made "executing" (the poststate of C should be "returning", i.e., delivering a result value, which can be ensured by a simple static check but is actually not required by the semantics).

The valuation functions $[\![\,\lrcorner\,]\!]$ and $[\![\,\lrcorner\,]\!]_{\mathrm{T}}$ for commands are generalized to take also the method environment *me* into account. In a method call $I_r = I_m\,(E_1,\ldots,E_p)$,

this environment is looked up to yield a method view and method behavior. From the view of the method declaration and the space of the current context, a new context is created and passed to the behavior, together with the values of the arguments $E_1, \ldots, E_p$.

The application of the behavior yields the pair $\langle r, t \rangle$ of a transition relation r and a termination condition t. In $[\![\_]\!]_T$, t is applied to the prestate s of the call in order to tell whether the call of the method is guaranteed to terminate. In $[\![\_]\!]$, r is applied to the prestate s of the call to yield some state s_0; if s_0 is "throwing", then it immediately represents the poststate of the call. Otherwise the poststate is (according to the semantics of method declarations) "executing"; the control data of the state then holds the result value of the method (deposited there by a `return` statement) and writes it into the variable denoted by I_r.

6.3 Formulas with Global Variables

As a prerequisite for specifying the commands of the method language, Figure 6.12 extends the language of formulas and terms based on the definitions given in Figure 6.11. The extension introduces the concept of *global variables* which are denoted by references of the form `?`I, for some identifier I. These references are evaluated in a "global" context that is in general different from the "local" context of those variables that are denoted by references of the form I (which we call *local variables* from now on). In this way we are able to express statements about the variables that are visible in the global context (where methods are declared) even inside the context of a command (where some of the global variables may be shadowed by local declarations).

For instance, take the method

```
method inc(J)
writesonly I throwsonly ␣ assumes TRUE
implements I'=I+J
{ I=I+J }
```

which increases the value of the global variable `I` visible in the context of the declaration of the method by the value of the parameter `J`.

Now the method may be invoked in a context where the declaration of a local variable `I` shadows the global variable:

```
{ var I=1; inc(I) }
```

Formulas with Global Variables: Definitions

$$
\begin{aligned}
& s \text{ EQUALS}^{c,c'} s' \equiv \\
& \quad \forall x \in \mathit{Variable} : x \in \mathit{space}(c') \vee \mathit{store}(s)(x) = \mathit{store}(s')(x) \\
& s = s' \text{ EXCEPT}^{c,c'} I_1, \ldots, I_n, ?J_1, \ldots, ?J_m \equiv \\
& \quad \forall x \in \mathit{Variable} : x \notin \{[\![I_1]\!]^{c'}, \ldots, [\![I_n]\!]^{c'}, [\![J_1]\!]^{c}, \ldots, [\![J_m]\!]^{c}\} \Rightarrow \\
& \qquad x \in \mathit{space}(c') \vee \mathit{store}(s)(x) = \mathit{store}(s')(x) \\
& s = s' \text{ AT}^{c,c'} I_1, \ldots, I_n, ?J_1, \ldots, ?J_m \equiv \\
& \quad \forall x \in \mathit{Variable} : x \in \{[\![I_1]\!]^{c'}, \ldots, [\![I_n]\!]^{c'}, [\![J_1]\!]^{c}, \ldots, [\![J_m]\!]^{c}\} \Rightarrow \\
& \qquad \mathit{store}(s)(x) = \mathit{store}(s')(x)
\end{aligned}
$$

Figure 6.11: Definitions for Formulas with Global Variables

The effect of the method invocation

```
inc(I)
```

can (in the local context of the invocation) be expressed by the formula

```
I=1 AND ?I'=?I+I AND I'=I
```

which says that the value of the global variable `I` is increased by the value of the local variable `I` and that the value of the local variable `I` remains unchanged.

In the global context, however, the effect of the method invocation is described by the formula

```
EXISTS $I,$J:
  $I=1 AND I'=I+$I AND $J=$I
```

Here the global variable `I` can be directly referenced and the pre/poststate values of the local variable `I` are described by the mathematical values `$I` and `$J`; this formula can be finally simplified to

```
I'=I+1
```

which expresses the effect of the program in a concise way.

Formulas with Global Variables: Abstract Syntax

$F \in$ Formula
$T \in$ Term
$R \in$ Reference
$I \in$ Identifier

$F ::= p\,(T_1,\ldots,T_n)$
$\quad \mid \texttt{readsonly} \mid \texttt{writesonly}\; I_1,\ldots,I_n,?J_1,\ldots,?J_m \mid$
$\quad \mid \texttt{unchanged}\; I_1,\ldots,I_n,?J_1,\ldots,?J_m \mid \ldots$
$T ::= R \mid R' \mid \ldots$
$R ::= I \mid ?I$

Valuation Functions

$[\![\_]\!] : \textbf{Formula} \rightarrow$
$\quad (\textit{Context} \times \textit{Context}) \rightarrow \textit{Environment} \rightarrow \textit{StateRelation}$
$[\![p\,(T_1,\ldots,T_n)]\!]^{c_0,c_1}(e)(s,s') \Leftrightarrow$
$\quad [\![p]\!]([\![T_1]\!]^{c_0,c_1}(e)(s,s'),\ldots,[\![T_n]\!]^{c_0,c_1}(e)(s,s'))$
$[\![\texttt{readsonly}]\!]^{c_0,c_1}(s,s') \Leftrightarrow$
$\quad s \text{ EQUALS}^{c_0,c_1} s'$
$[\![\texttt{writesonly}\; I_1,\ldots,I_n,?J_1,\ldots,?J_m]\!]^{c_0,c_1}(s,s') \Leftrightarrow$
$\quad s = s' \text{ EXCEPT}^{c_0,c_1}\; I_1,\ldots,I_n,?J_1,\ldots,?J_m$
$[\![\texttt{unchanged}\; I_1,\ldots,I_n,?J_1,\ldots,?J_m]\!]^{c_0,c_1}(s,s') \Leftrightarrow$
$\quad s = s' \text{ AT}^{c_0,c_1}\; I_1,\ldots,I_n,?J_1,\ldots,?J_m$
$[\![\ldots]\!]^{c_0,c_1}(e)(s,s') \Leftrightarrow \ldots$

$[\![\_]\!] : \textbf{Term} \rightarrow$
$\quad (\textit{Context} \times \textit{Context}) \rightarrow \textit{Environment} \rightarrow \textit{BinaryStateFunction}$
$[\![I]\!]^{c_0,c_1}(e)(s,s') = read(s,I)^{c_1}$
$[\![?I]\!]^{c_0,c_1}(e)(s,s') = read(s,I)^{c_0}$
$[\![I']\!]^{c_0,c_1}(e)(s,s') = read(s',I)^{c_1}$
$[\![?I']\!]^{c_0,c_1}(e)(s,s') = read(s',I)^{c_0}$
$[\![\ldots]\!]^{c_0,c_1}(e)(s,s') = \ldots$

Figure 6.12: Formulas with Global Variables

The reasoning calculus we are going to devise will in an "inside-out" manner first construct the "local context" description of the method invocation and from this construct the "global context" description.

Figure 6.12 depicts the valuation functions for formulas and terms with global variables. These functions depend on two contexts c_0 and c_1 where c_0 denotes the the global context which is used to resolve store references of form $?I$ respectively $?I'$ and c_1 denotes the local context which is used to resolve store references of form I respectively I'. Since variable references only occur inside terms in the form of such store references, we expand the semantics of variable references within the semantics of terms (i.e. we omit a separate valuation function for variable references). Thus we are able to make use of the generalized form of the *read* operation introduced in a previous section.

6.4 Method Specifications

As a provision for the reasoning calculus of the method language, we introduce in Figure 6.13 the domain *MethodSpec* of "method specifications". Each such specification consists of a sequence of method parameters and a "core specification", i.e. an element of domain *Spec*, which represents the content of the specification *S* by which the body of the method has been annotated (the frame of potentially modified variables, the list of potentially thrown exceptions, the formulas denoting a state relation and a state condition). The domain *SpecEnv* of "specification environments" then maps method names to method specifications.

The predicate *specifies* states whether, in a given context, a method specification adequately describes a method behavior. Based on this predicate, another predicate of the same name is introduced which states whether, in a given context, an environment *se* of method specification adequately models an environment *me* of method implementations.

A method specification is an adequate description of a method implementation, if the following is true for any prestate s of the method call (compare with the formal definition): after creating from the declaration context c a new context c' for the formal parameters $J_1, \ldots, J_p$ and writing into the variables denoted by parameters $J_1, \ldots, J_p$ the actual argument values $v_1, \ldots, v_p$ of the call, the execution of the method body yields a poststate s' which is "executing" or "throwing" such that

- s' is allowed by the formula $F_C \Rightarrow F_R$ in context c' for the prestate of the method body,

Method Specifications: Definitions

$$
\begin{array}{l}
\mathit{Spec} := \text{Identifier}^* \times \text{Identifier}^* \times \text{Formula} \times \text{Formula} \times \text{Term} \\
\mathit{MethodSpec} := \text{Identifier}^* \times \mathit{Spec} \\
\mathit{SpecEnv} := \text{Identifier} \to \mathit{MethodSpec} \\
\\
\mathit{specifies} \subseteq \mathit{MethodSpec} \times \mathit{Behavior} \times \mathit{Context} \\
\mathit{specifies}(\langle (J_1,\ldots,J_p), \langle (I_1,\ldots,I_n), (K_1,\ldots,K_m), F_C, F_R, T\rangle\rangle, \\
\qquad b, c) \Leftrightarrow \\
\quad \{J_1,\ldots,J_p\} \cap \{I_1,\ldots,I_n\} = \emptyset \wedge \\
\quad F_C, F_R \text{ and } T \text{ have no free (mathematical or state) variables} \wedge \\
\quad F_C \text{ and } T \text{do not depend on the poststate} \wedge \\
\quad \forall v_1,\ldots,v_p \in \mathit{Value}, s \in \mathit{State}, e \in \mathit{Environment}, c' \in \mathit{Context} : \\
\qquad \mathit{executes}(\mathit{control}(s)) \wedge \mathit{DifferentVariables}(c') \wedge \\
\qquad c' \text{ EQUALS } c \wedge \ \mathit{space}(c') \subseteq \mathit{space}(c) \ \Rightarrow \\
\qquad \text{LET} \\
\qquad\quad \langle r,t\rangle = b^{c'}(v_1,\ldots,v_p), \\
\qquad\quad c'' = \mathit{push}(c', J_1,\ldots,J_p), \\
\qquad\quad s_1 = \mathit{writes}(s, J_1, v_1, \ldots, J_p, v_p)^{c''} \\
\qquad \text{IN } (\forall s' \in \mathit{State} : r(s,s') \ \Rightarrow \\
\qquad\quad [\![F_C \mathrel{=>} F_R]\!]^{c,c''}(e)(s_1,s') \wedge \\
\qquad\quad s = s' \text{ EXCEPT } \mathit{range}(c') \cup \text{range}(\mathit{view}(c)) \wedge \\
\qquad\quad s = s' \text{ EXCEPT}^{c,c} \ I_1,\ldots,I_n \wedge \\
\qquad\quad (\mathit{executes}(\mathit{control}(s')) \vee \mathit{throws}(\mathit{control}(s'))) \wedge \\
\qquad\quad (\mathit{throws}(\mathit{control}(s')) \ \Rightarrow \\
\qquad\qquad \mathit{key}(\mathit{control}(s')) \in \{K_1,\ldots,K_m\})) \\
\qquad \wedge ([\![F_C]\!]^{c,c''}(e)(s_1,s_1) \Rightarrow t(s)) \\
\\
\mathit{specifies} \subseteq \mathit{SpecEnv} \times \mathit{MethodEnv} \times \mathit{Context} \\
\mathit{specifies}(se, me, c) \Leftrightarrow \\
\quad \forall I \in \text{Identifier} : \\
\qquad \text{LET } \langle v, b\rangle = me(I) \text{ IN} \\
\qquad v = \mathit{view}(c) \wedge \mathit{specifies}(se(I), b, c)
\end{array}
$$

Figure 6.13: Definitions for Method Specifications

- the only variable values in which s' differs from s in context c are among those denoted by $I_1, \ldots, I_n$,
- s' throws no other exceptions than those denoted by $K_1, \ldots, K_m$.

Furthermore, if the formula F_C holds in the context c' on the prestate of the method body, then the execution of the method terminates.

In this formulation, we note the following points:

- the specification formulas F_C and F_R are formulated from the method body's point of view (prestate, poststate, and context),
- except that a state flag that is "returning" (or, if not prohibited by static checks, also "continuing" or "breaking") is cleared to "executing";
- F_C serves two roles: on the one hand, it denotes the termination condition of the command; on the other hand, it is also taken as an additional precondition to the transition formula F_R.

The first item establishes the fundamental mindset for understanding a method specification: variable values and control data are taken from those states that are in effect immediately before respectively after the execution of the method body.

On the contrary, the last two items are pragmatic design decisions (which might be reconsidered): First, we abstract from the fact that method bodies are expected to terminate in a "returning" state to make their execution look like that of a command in the current context (where a normal poststate is "executing").

Second, we restrict the "constraining power" of the transition relation to only those prestates where the method is guaranteed to terminate. This condition is typically inserted anyway, because otherwise specifications have to become much more general as intended (and consequently much more complicated). As an example, take the method (with operations being interpreted over $\mathbb{N}$)

```
method count(i) S {
  s = 0;
  while (i > 0) { s = s+1; i = i-1; }
  return s;
}
```

where the loop has invariant

```
(i >= 0 => i' >= 0) AND
s'+i' = s+i
```

From the rules of the calculus, we may construct for the method body the specification formula

```
(i >= 0 => i' >= 0) AND
s'+i' = i AND
i' <= 0
```

which can be simplified to

```
(i >= 0 => i' = 0) AND
s'+i' = i AND
i' <= 0
```

While this specification formula is correct, it is very general and does not really capture our intention. By adding the termination condition `i >= 0` as a precondition, we can simplify the formula to a weaker but more natural version

```
i >= 0 => (i' = 0 AND s' = i)
```

respectively (if we are not interested in the poststate value of the parameter `i`)

```
i >= 0 => s' = i
```

which actually expresses all that we want to say about the method body.

6.5 Reasoning about Commands

Based on the generalized class of formulas introduced in the previous section, Figure 6.14 introduces some auxiliary definitions. In particular, the predicate $pushes(c, c', \{V_1, \ldots, V_o\})$ states that c and c' are two contexts that differ only in their views of the identifiers $V_1, \ldots, V_o$ and that the variables denoted by these identifiers and the space of c' are from the space of c.

Figure 6.15 now gives updated versions of the three core judgements for the commands of the method language[2]. When reasoning about a command C, we now

[2]The first judgement also uses of a valuation $[\![C]\!]_\perp^{c,me}$ which is a natural generalization of the previously introduced valuation $[\![C]\!]_\perp$ by taking contexts and method environments into account.

Method Language: Definitions

$pushes \subseteq Context \times Context \times \text{Idenfier}^*$
$pushes(c, c', \{V_1, \ldots, V_o\}) :\Leftrightarrow$
$\quad DifferentVariables(c) \wedge DifferentVariables(c') \wedge$
$\quad c = c' \text{ EXCEPT } V_1, \ldots, V_o \wedge$
$\quad \{[\![V_1]\!]^{c'}, \ldots, [\![V_o]\!]^{c'}\} \cup space(c') \subseteq space(c)$

$[\![\_]\!] : \text{Formula} \rightarrow (Context \times Context) \rightarrow StateCondition$
$[\![F]\!]^{c_0,c_1}(s) \Leftrightarrow \forall e \in \text{Environment} : [\![F]\!]^{c_0,c_1}(e)(s,s)$

$[\![\_]\!] : \text{Formula} \rightarrow StateCondition$
$[\![F]\!](s) \Leftrightarrow$
$\quad \forall c_0, c_1 \in \text{Context} :$
$\quad\quad DifferentVariables(c_0) \wedge DifferentVariables(c_1) \Rightarrow$
$\quad\quad\quad [\![F]\!]^{c_0,c_1}(s)$

$\_ \simeq \_ : \mathbb{P}(\text{Expression} \times \text{Term})$
$E \simeq T \Leftrightarrow$
$\quad T$ has no free (mathematical or state) variables $\wedge$
$\quad T$ has no primed program variables $\wedge$
$\quad T$ has no occurrence of `next` $\wedge$
$\quad \forall c_0, c_1 \in Context :$
$\quad\quad DifferentVariables(c_0) \wedge DifferentVariables(c_1) \Rightarrow$
$\quad\quad\quad \forall s, s' \in Store, e \in Environment :$
$\quad\quad\quad\quad [\![E]\!]^{c_1}(s) = [\![T]\!]^{c_0,c_1}(e)(s,s')$

$\_ \overset{\text{D}}{\simeq} \_ : \mathbb{P}(\text{Expression} \times \text{Formula})$
$E \overset{\text{D}}{\simeq} F_D \Leftrightarrow$
$\quad \forall c_0, c_1 \in Context :$
$\quad\quad DifferentVariables(c_0) \wedge DifferentVariables(c_1) \Rightarrow$
$\quad\quad\quad \forall s \in State : [\![E]\!]^{c_1}_{\text{D}}(s) \Leftrightarrow [\![F_D]\!]^{c_0,c_1}(s)$

Figure 6.14: Definitions for the Method Language

Commands: Judgements

$$
\begin{array}{l}
se, \{V_1, \ldots, V_o\} \vdash C \checkmark F \Leftrightarrow \\
\quad F \text{ has no free (mathematical or state) variables} \wedge \\
\quad F \text{ does not depend on the poststate} \Rightarrow \\
\qquad \forall c, c' \in \mathit{Context}, me \in \mathit{MethodEnv} : \\
\qquad\quad \mathit{specifies}(se, me, c) \wedge \mathit{pushes}(c, c', \{V_1, \ldots, V_o\}) \Rightarrow \\
\qquad\qquad \forall s, s' \in \mathit{State} : \mathit{executes}(\mathit{control}(s)) \wedge [\![F]\!]^{c,c'}(s) \Rightarrow \\
\qquad\qquad\quad ([\![C]\!]^{c',me}(s, s') \Leftrightarrow [\![C]\!]_{\perp}^{c',me}(s, s'))
\end{array}
$$

$$
\begin{array}{l}
se, \{V_1, \ldots, V_o\} \vdash C : F \Leftrightarrow \\
\quad F \text{ has no free (mathematical or state) variables} \wedge \\
\quad \forall c, c' \in \mathit{Context}, me \in \mathit{MethodEnv} : \\
\qquad \mathit{specifies}(se, me, c) \wedge \mathit{pushes}(c, c', \{V_1, \ldots, V_o\}) \Rightarrow \\
\qquad\quad \forall s, s' \in \mathit{State}, e \in \mathit{Environment} : \\
\qquad\qquad \mathit{executes}(\mathit{control}(s)) \wedge [\![C]\!]^{c',me}(s, s') \Rightarrow \\
\qquad\qquad\quad [\![F]\!]^{c,c'}(e)(s, s')
\end{array}
$$

$$
\begin{array}{l}
se, \{V_1, \ldots, V_o\} \vdash C \downarrow F \Leftrightarrow \\
\quad F \text{ has no free (mathematical or state) variables} \wedge \\
\quad F \text{ does not depend on the poststate} \Rightarrow \\
\qquad \forall c, c' \in \mathit{Context}, me \in \mathit{MethodEnv} : \\
\qquad\quad \mathit{specifies}(se, me, c) \wedge \mathit{pushes}(c, c', \{V_1, \ldots, V_o\}) \Rightarrow \\
\qquad\qquad \forall s \in \mathit{State} : \mathit{executes}(\mathit{control}(s)) \wedge [\![F]\!]^{c,c'}(s) \Rightarrow \\
\qquad\qquad\quad [\![C]\!]_{\mathrm{T}}^{c',me}(s)
\end{array}
$$

Figure 6.15: Judgements for Commands of the Method Language

take into account a specification environment *se* which we assume to correctly describe all methods that may be invoked by C and a set of identifiers $\{V_1, \ldots, V_o\}$ that denote those locally declared variables in which scope the command C is executed. Then

- $se, \{V_1, \ldots, V_o\} \vdash C \checkmark F$ guarantees the well-definedness of all expressions encountered by the execution of C in any prestate that satisfies F;
- $se, \{V_1, \ldots, V_o\} \vdash C : F$ guarantees that C only performs such state transitions that are allowed by F;
- $se, \{V_1, \ldots, V_o\} \vdash C \downarrow F$ guarantees the termination of of the execution of C in any prestate that satisfies F.

It should be noted that the judgements are based on the *specifications* of the methods that may be invoked by C, not on their implementations. This allows a style of *compositional reasoning* which decouples the correct usage of a method from the accidents of its implementation by placing the method specification into the focus of consideration:

- on the one hand, we may verify the correctness of a program from the specifications of the methods it calls,
- on the other hand, we have to verify that the implementations of the methods satisfy their specifications.

The specification of a method thus becomes the "contract" between the caller of the method and its implementor (*design by contract* principle): any change to the implementation of a method does not affect the overall correctness of the program provided that we can verify that the modified implementation also satisfies the specification (i.e. that the contract is preserved).

Most rules of the command language can be easily generalized to the new forms of judgements by forwarding the specification environment and the set of locally declared variables from the conclusions to to the corresponding hypotheses (it should be noted, however, that the generated proof obligations may now involve global variable references $?V_1, \ldots, ?V_o$ denoting variables different from the local variables $V_1, \ldots, V_o$). Substantial modifications are only necessary in the rules for variable declarations/definitions and exception handlers, because these are the only commands that introduce local variables.

Figures 6.16 and 6.17 give the new well-definedness rules for variable declarations/definitions and exception handlers. They are derived from the original

Variable Declarations/Definitions: Well-Definedness

$$\frac{I \notin \{V_1,\ldots,V_o\} \quad se,\{V_1,\ldots,V_o,I\} \vdash C\checkmark P[?J/I]}{se,\{V_1,\ldots,V_o\} \vdash \texttt{var}\ I\texttt{;}\ C\checkmark P}$$

$$\frac{I \in \{V_1,\ldots,V_o\} \quad \$J \text{ does not occur in } P \quad se,\{V_1,\ldots,V_o\} \vdash C\checkmark \texttt{EXISTS}\ \$J : P[\$J/I]}{se,\{V_1,\ldots,V_o\} \vdash \texttt{var}\ I\texttt{;}\ C\checkmark P}$$

$$\frac{I \notin \{V_1,\ldots,V_o\} \quad E \overset{\text{D}}{\simeq} F_D \quad \forall s \in State : [\![(\texttt{now.executes AND}\ P) \texttt{=>}\ F_D]\!](s) \quad se,\{V_1,\ldots,V_o,I\} \vdash C\checkmark P[?J/I]\ \texttt{AND}\ I = T[?I/I]}{se,\{V_1,\ldots,V_o\} \vdash \texttt{var}\ I\texttt{=}E\texttt{;}\ C\checkmark P}$$

$$\frac{I \in \{V_1,\ldots,V_o\} \quad E \overset{\text{D}}{\simeq} F_D \quad \forall s \in State : [\![(\texttt{now.executes AND}\ P) \texttt{=>}\ F_D]\!](s) \quad \$J \text{ does not occur in } P \text{ and } T \quad se,\{V_1,\ldots,V_o\} \vdash C\checkmark \texttt{EXISTS}\ \$J : P[\$J/I]\ \texttt{AND}\ I = T[\$J/I]}{se,\{V_1,\ldots,V_o\} \vdash \texttt{var}\ I\texttt{=}E\texttt{;}\ C\checkmark P}$$

Figure 6.16: Well-Definedness Rules for Variable Declarations/Definitions

Exception Handlers: Well-Definedness

$$\frac{\begin{array}{l} I_v \notin \{V_1,\ldots,V_o\} \\ \#I_s \text{ does not occur in } Q \\ se,\{V_1,\ldots,V_o\} \vdash \textsc{post}(C_1,P) = Q \\ se,\{V_1,\ldots,V_o\} \vdash C_1 \checkmark P \\ se,\{V_1,\ldots,V_o,I_v\} \vdash C_2 \checkmark \\ \quad \texttt{EXSTATE}\ \#I_s\texttt{:} \\ \qquad Q[\#I_s/\texttt{now}][?I_v/I_v]\ \texttt{AND} \\ \qquad \#I_s\texttt{.throws}\ I_k\ \texttt{AND}\ \#I_s\texttt{.value} = I_v \end{array}}{se,\{V_1,\ldots,V_o\} \vdash \texttt{try}\ C_1\ \texttt{catch(}I_k\ I_v\texttt{)}\ C_2 \checkmark P}$$

$$\frac{\begin{array}{l} I_v \in \{V_1,\ldots,V_o\} \\ \$J \text{ does not occur in } Q \\ \#I_s \text{ does not occur in } Q \\ se,\{V_1,\ldots,V_o\} \vdash \textsc{post}(C_1,P) = Q \\ se,\{V_1,\ldots,V_o\} \vdash C_1 \checkmark P \\ se,\{V_1,\ldots,V_o\} \vdash C_2 \checkmark \\ \quad \texttt{EXISTS}\ \$J\texttt{:}\ \texttt{EXSTATE}\ \#I_s\texttt{:} \\ \qquad Q[\#I_s/\texttt{now}][\$J/I_v]\ \texttt{AND} \\ \qquad \#I_s\texttt{.throws}\ I_k\ \texttt{AND}\ \#I_s\texttt{.value} = I_v \end{array}}{se,\{V_1,\ldots,V_o\} \vdash \texttt{try}\ C_1\ \texttt{catch(}I_k\ I_v\texttt{)}\ C_2 \checkmark P}$$

Figure 6.17: Well-Definedness Rules for Exception Handlers

Variable Declarations: Verification

$$\frac{I \notin \{V_1,\ldots,V_o\} \quad se,\{V_1,\ldots,V_o,I\} \vdash C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n,?I} \quad I_a \neq I_b \quad \$I_a,\$I_b \text{ do not occur in } F}{se,\{V_1,\ldots,V_o\} \vdash \texttt{var}\ I\texttt{;}\ C : \big[\texttt{EXISTS}\ \$I_a,\$I_b\texttt{:}\ F[\$I_a/I,\$I_b/I'][I/?I,I'/?I']\big]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n,I}}$$

$$\frac{I \notin \{V_1,\ldots,V_o\} \quad se,\{V_1,\ldots,V_o,I\} \vdash C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \quad ?I \notin \{I_1,\ldots,I_n\} \quad I_a \neq I_b \quad \$I_a,\$I_b \text{ do not occur in } F}{se,\{V_1,\ldots,V_o\} \vdash \texttt{var}\ I\texttt{;}\ C : \big[\texttt{EXISTS}\ \$I_a,\$I_b\texttt{:}\ F[\$I_a/I,\$I_b/I'][I/?I,I'/?I']\big]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{\{I_1,\ldots,I_n\}\setminus\{I\}}}$$

$$\frac{I \in \{V_1,\ldots,V_o\} \quad se,\{V_1,\ldots,V_o\} \vdash C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{\{I_1,\ldots,I_n\}} \quad I_a \neq I_b \quad \$I_a,\$I_b \text{ do not occur in } F}{se,\{V_1,\ldots,V_o\} \vdash \texttt{var}\ I\texttt{;}\ C : \big[\texttt{EXISTS}\ \$I_a,\$I_b\texttt{:}\ F[\$I_a/I,\$I_b/I']\big]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{\{I_1,\ldots,I_n\}\setminus\{I\}}}$$

Figure 6.18: Verification Rules for Variable Declarations

Variable Definitions: Verification

$$\frac{\begin{array}{l} I \notin \{V_1,\ldots,V_o\} \\ se,\{V_1,\ldots,V_o,I\} \vdash C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n,?I} \\ I_a \neq I_b \\ \$I_a,\$I_b \text{ do not occur in } F \\ E \simeq T \end{array}}{\begin{array}{l} se,\{V_1,\ldots,V_o\} \vdash \texttt{var}\ I\texttt{=}E\texttt{;}\ C : \\ \quad [\texttt{EXISTS}\ \$I_a,\$I_b\texttt{:}\ \$I_a\texttt{=}T\ \texttt{AND} \\ \qquad F[\$I_a/I,\$I_b/I'][I/?I,I'/?I']]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n,I} \end{array}}$$

$$\frac{\begin{array}{l} I \notin \{V_1,\ldots,V_o\} \\ se,\{V_1,\ldots,V_o,I\} \vdash C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \\ ?I \notin \{I_1,\ldots,I_n\} \\ I_a \neq I_b \\ \$I_a,\$I_b \text{ do not occur in } F \\ E \simeq T \end{array}}{\begin{array}{l} se,\{V_1,\ldots,V_o\} \vdash \texttt{var}\ I\texttt{=}E\texttt{;}\ C : \\ \quad [\texttt{EXISTS}\ \$I_a,\$I_b\texttt{:}\ \$I_a\texttt{=}T\ \texttt{AND} \\ \qquad F[\$I_a/I,\$I_b/I'][I/?I,I'/?I']]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{\{I_1,\ldots,I_n\}\setminus\{I\}} \end{array}}$$

$$\frac{\begin{array}{l} I \in \{V_1,\ldots,V_o\} \\ se,\{V_1,\ldots,V_o\} \vdash C : [F]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{\{I_1,\ldots,I_n\}} \\ I_a \neq I_b \\ \$I_a,\$I_b \text{ do not occur in } F \\ E \simeq T \end{array}}{\begin{array}{l} se,\{V_1,\ldots,V_o\} \vdash \texttt{var}\ I\texttt{=}E\texttt{;}\ C : \\ \quad [\texttt{EXISTS}\ \$I_a,\$I_b\texttt{:}\ \$I_a\texttt{=}T\ \texttt{AND} \\ \qquad F[\$I_a/I,\$I_b/I']]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{\{I_1,\ldots,I_n\}\setminus\{I\}} \end{array}}$$

Figure 6.19: Verification Rules for Variable Definitions

Exception Handlers: Verification

$$
\frac{\begin{array}{l}
I_v \notin \{V_1,\ldots,V_o\} \\
se,\{V_1,\ldots,V_o\} \vdash C_1 : [F_1]^{F_{c1},F_{b1},F_{r1},\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \\
se,\{V_1,\ldots,V_o,I_v\} \vdash C_2 : [F_2]^{F_{c2},F_{b2},F_{r2},\{L_1,\ldots,L_o\}}_{I_1,\ldots,I_n} \\
I_a \neq I_b \\
\{I_a,I_b\} \cap \{I_1,\ldots,I_n\} = \emptyset \\
I_s \neq I_t \\
\$I_1,\ldots,\$I_n,\#I_s \text{ do not occur in } F_1 \text{ and } F_2 \\
\$I_a,\$I_b,\#I_t \text{ do not occur in } F_2
\end{array}}{\begin{array}{l}
se,\{V_1,\ldots,V_o\} \vdash \texttt{try}\ C_1\ \texttt{catch(}I_k\ I_v\texttt{)}\ C_2 : \\
\quad [\texttt{EXISTS}\ \$I_1,\ldots,\$I_n\texttt{:}\ \texttt{EXSTATE}\ \#I_s\texttt{:} \\
\qquad F_1[\#I_s/\texttt{next}][\$I_1/I_1\texttt{'},\ldots,\$I_n/I_n\texttt{'}]\ \texttt{AND} \\
\qquad \texttt{IF}\ \#I_s\texttt{.throws}\ I_k\ \texttt{THEN} \\
\qquad\quad \texttt{EXISTS}\ \$I_a,\$I_b\texttt{:}\ \texttt{EXSTATE}\ \#I_t\texttt{:} \\
\qquad\qquad \$I_a = \#I_s\texttt{.value}\ \texttt{AND}\ \#I_t\texttt{.executes}\ \texttt{AND} \\
\qquad\qquad F_2[\#I_t/\texttt{now}][\$I_a/I_v][\$I_1/I_1,\ldots,\$I_n/I_n][\$I_b/I_v\texttt{'}] \\
\qquad\qquad\quad [I_v/\texttt{?}I_v,I_v\texttt{'}/\texttt{?}I_v\texttt{'}] \\
\qquad \texttt{ELSE} \\
\qquad\quad I_1\texttt{'} = \$I_1\ \texttt{AND}\ \ldots\ \texttt{AND}\ I_n\texttt{'} = \$I_n\ \texttt{AND}\ \texttt{next==}\#I_s \\
\quad]^{F_{c1}\ \texttt{OR}\ F_{c2},F_{b1}\ \texttt{OR}\ F_{b2},F_{r1}\ \texttt{OR}\ F_{r2},(\{K_1,\ldots,K_m\}\setminus\{I_k\})\cup\{L_1,\ldots,L_o\}}_{I_1,\ldots,I_n}
\end{array}}
$$

Figure 6.20: Verification Rules for Exception Handlers (1/2)

Exception Handlers: Verification

$$
\frac{\begin{array}{l}
I_v \in \{V_1,\ldots,V_o\} \\
se,\{V_1,\ldots,V_o\} \vdash C_1 : [F_1]^{F_{c1},F_{b1},F_{r1},\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \\
se,\{V_1,\ldots,V_o\} \vdash C_2 : [F_2]^{F_{c2},F_{b2},F_{r2},\{L_1,\ldots,L_o\}}_{I_1,\ldots,I_n} \\
I_a \neq I_b \\
\{I_a,I_b\} \cap \{I_1,\ldots,I_n\} = \emptyset \\
I_s \neq I_t \\
\$I_1,\ldots,\$I_n,\#I_s \text{ do not occur in } F_1 \text{ and } F_2 \\
\$I_a,\$I_b,\#I_t \text{ do not occur in } F_2
\end{array}}{\begin{array}{l}
se,\{V_1,\ldots,V_o\} \vdash \texttt{try}\ C_1\ \texttt{catch(}I_k\ I_v\texttt{)}\ C_2 : \\
\quad [\texttt{EXISTS}\ \$I_1,\ldots,\$I_n\texttt{:}\ \texttt{EXSTATE}\ \#I_s\texttt{:} \\
\qquad F_1[\#I_s/\texttt{next}][\$I_1/I_1\texttt{'},\ldots,\$I_n/I_n\texttt{'}]\ \texttt{AND} \\
\qquad \texttt{IF}\ \#I_s\texttt{.throws}\ I_k\ \texttt{THEN} \\
\qquad\quad \texttt{EXISTS}\ \$I_a,\$I_b\texttt{:}\ \texttt{EXSTATE}\ \#I_t\texttt{:} \\
\qquad\qquad \$I_a = \#I_s\texttt{.value}\ \texttt{AND}\ \#I_t\texttt{.executes}\ \texttt{AND} \\
\qquad\qquad F_2[\#I_t/\texttt{now}][\$I_a/I_v][\$I_1/I_1,\ldots,\$I_n/I_n][\$I_b/I_v\texttt{'}] \\
\qquad \texttt{ELSE} \\
\qquad\quad I_1\texttt{'} = \$I_1\ \texttt{AND}\ \ldots\ \texttt{AND}\ I_n\texttt{'} = \$I_n\ \texttt{AND}\ \texttt{next==}\#I_s \\
\quad]^{F_{c1}\ \texttt{OR}\ F_{c2},F_{b1}\ \texttt{OR}\ F_{b2},F_{r1}\ \texttt{OR}\ F_{r2},(\{K_1,\ldots,K_m\}\setminus\{I_k\})\cup\{L_1,\ldots,L_o\}}_{I_1,\ldots,I_n}
\end{array}}
$$

Figure 6.21: Verification Rules for Exception Handlers (2/2)

Variable Declarations and Definitions: Termination

$$\frac{I \notin \{V_1,\ldots,V_o\} \quad se,\{V_1,\ldots,V_o,I\} \vdash C \downarrow F[?I/I]}{se,\{V_1,\ldots,V_o\} \vdash \texttt{var}\ I\texttt{;}\ C \downarrow F}$$

$$\frac{I \in \{V_1,\ldots,V_o\} \quad \$I \text{ does not occur in } F \quad se,\{V_1,\ldots,V_o\} \vdash C \downarrow \texttt{EXISTS}\ \$I\texttt{:}\ F[\$I/I]}{se,\{V_1,\ldots,V_o\} \vdash \texttt{var}\ I\texttt{;}\ C \downarrow F}$$

$$\frac{I \notin \{V_1,\ldots,V_o\} \quad E \simeq T \quad se,\{V_1,\ldots,V_o,I\} \vdash C \downarrow I\texttt{=}T[?I/I]\ \texttt{AND}\ F[?I/I]}{se,\{V_1,\ldots,V_o\} \vdash \texttt{var}\ I\texttt{=}E\texttt{;}\ C \downarrow F}$$

$$\frac{I \in \{V_1,\ldots,V_o\} \quad E \simeq T \quad \$I \text{ does not occur in } T \text{ and } F \quad se,\{V_1,\ldots,V_o\} \vdash C \downarrow \texttt{EXISTS}\ \$I\texttt{:}\ I = T[\$I/I]\ \texttt{AND}\ F[\$I/I]}{se,\{V_1,\ldots,V_o\} \vdash \texttt{var}\ I\texttt{=}E\texttt{;}\ C \downarrow F}$$

Figure 6.22: Termination Rules for Variable Declarations and Definitions

Exception Handlers: Termination

$$\frac{\begin{array}{l} I_v \notin \{V_1,\ldots,V_o\} \\ se,\{V_1,\ldots,V_o\} \vdash C_1 \downarrow F \\ se,\{V_1,\ldots,V_o\} \vdash C_1 : [S]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \\ se,\{V_1,\ldots,V_o,I_v\} \vdash C_2 \downarrow \texttt{now.executes AND} \\ \quad \texttt{EXISTS } \$I_1,\ldots,\$I_n\texttt{: EXSTATE } \#I_s,\#I_t\texttt{:} \\ \quad\quad \#I_s\texttt{.executes AND} \\ \quad\quad \#I_t\texttt{.throws } I_k \texttt{ AND } I_v = \#I_t\texttt{.value AND} \\ \quad\quad F[\#I_s/\texttt{now}][\$I_1/I_1,\ldots,\$I_n/I_n][?I_v/I_v] \texttt{ AND} \\ \quad\quad S[\#I_s/\texttt{now},\#I_t/\texttt{next}][\$I_1/I_1,\ldots,\$I_n/I_n] \\ \quad\quad\quad [?I_v/I_v,?I_v\texttt{'}/I_v\texttt{'}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}] \\ I_s \neq I_t \\ \$I_1,\ldots,\$I_n,\#I_s,\#I_t \text{ do not occur in } F \text{ and } S \end{array}}{se,\{V_1,\ldots,V_o\} \vdash \texttt{try } C_1 \texttt{ catch(}I_k\ I_v\texttt{) } C_2 \downarrow F}$$

$$\frac{\begin{array}{l} I_v \in \{V_1,\ldots,V_o\} \\ se,\{V_1,\ldots,V_o\} \vdash C_1 \downarrow F \\ se,\{V_1,\ldots,V_o\} \vdash C_1 : [S]^{F_c,F_b,F_r,\{K_1,\ldots,K_m\}}_{I_1,\ldots,I_n} \\ se,\{V_1,\ldots,V_o\} \vdash C_2 \downarrow \texttt{now.executes AND} \\ \quad \texttt{EXISTS } \$I_1,\ldots,\$I_n,\$I_a,\$I_b\texttt{: EXSTATE } \#I_s,\#I_t\texttt{:} \\ \quad\quad \#I_s\texttt{.executes AND} \\ \quad\quad \#I_t\texttt{.throws } I_k \texttt{ AND } I_v = \#I_t\texttt{.value AND} \\ \quad\quad F[\#I_s/\texttt{now}][\$I_1/I_1,\ldots,\$I_n/I_n][\$I_a/I_v] \texttt{ AND} \\ \quad\quad S[\#I_s/\texttt{now},\#I_t/\texttt{next}][\$I_1/I_1,\ldots,\$I_n/I_n] \\ \quad\quad\quad [\$I_a/I_v,\$I_b/I_v\texttt{'}][I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}] \\ I_a \neq I_b \wedge \{I_a,I_b\} \cap \{I_1,\ldots,I_n\} = \emptyset \wedge I_s \neq I_t \\ \$I_1,\ldots,\$I_n,\$I_a,\$I_b,\#I_s,\#I_t \text{ do not occur in } F \text{ and } S \end{array}}{se,\{V_1,\ldots,V_o\} \vdash \texttt{try } C_1 \texttt{ catch(}I_k\ I_v\texttt{) } C_2 \downarrow F}$$

Figure 6.23: Termination Rules for Exception Handlers

rules but take into account whether the declaration of a variable I for the command C shadows a global variable ($I \notin \{V_1, \ldots, V_o\}$) or just another local variable ($I \in \{V_1, \ldots, V_o\}$). In the first case, any reference I in the formula P characterizing the current state is replaced by $?I$ in the formula characterizing the state of the command C; since the declaration shadows the global variable, for checking the well-definedness of C the variable set has to be extended to $\{V_1, \ldots, V_o, I\}$. In the second case, I is replaced by a fresh mathematical variable $\$J$; since I already denotes a local variable, the variable set $\{V_1, \ldots, V_o\}$ needs not be extended.

Correspondingly, Figure 6.18 gives the new verification rules for variable declarations. The first two cover the case $I \notin \{V_1, \ldots, V_o\}$ where a locally declared variable I shadows the global variable of the same name; the third rule covers the case $I \in \{V_1, \ldots, V_o\}$ where the local variable shadows another local variable. The third rule is essentially the same as the original rule for variable declarations.

The first two rules differ in whether the command changes the shadowed global variable (which is possible if C invokes a method). The first rule applies, if this is indeed the case as indicated by the reference $?I$ in the list of modified variables; this reference can be renamed to I in the list of variables modified by the declaration (since I and $?I$ denote the same variable in the context of the declaration). The second rule applies, if the global variable has not been modified ($?I \notin \{I_1, \ldots, I_n\}$); any potential occurrence of I in the list of variables modified by C may thus be removed from the list of variables modified by the declaration. In both cases, any reference $?I$ respectively $?I'$ in the specification of F must be renamed to I respectively I' in the specification of the variable declaration.

Figure 6.19 gives the corresponding rules for variable definitions. The rules in Figures 6.20 and 6.21 are straight-forward generalizations of the rule for exception handlers; one is applicable if the exception parameter I_v shadows a global variable of the same name, the other one is applicable, if this is not the case.

The correspondingly generalized rules for the termination calculus are shown in Figures 6.22 and 6.23; also they can be derived from the original rules by taking care of the appropriate renaming of a global variable whenever it is shadowed by a local declaration.

We omit the proofs of the rules discussed above but rather focus on the corresponding rules for the new command "method call" shown in Figures 6.24, 6.25, 6.26, and 6.27. While there is one rule for deriving the well-definedness respectively termination of method calls, the verification of method calls is covered by two rules, because this judgement depends on the status of the variable I_r that receives the result of the method calls:

- The rule in Figure 6.25 is applicable if I_r is locally declared or is not in the frame of the method call ($I_r \in \{V_1, \ldots, V_o\} \vee I_r \notin \{I_1, \ldots, I_n\}$). In this

Method Calls: Well-Definedness

$$\frac{\begin{array}{l} se(I_m) = \langle (J_1,\ldots,J_p), \langle (I_1,\ldots,I_n), (K_1,\ldots,K_m), F_C, F_R, T\rangle\rangle \\ E_1 \overset{\mathrm{D}}{\simeq} F_1, \ldots, E_P \overset{\mathrm{D}}{\simeq} F_p \\ \forall s \in State: \\ \quad [\![(\texttt{now.executes AND } F) \texttt{ => } (F_1 \texttt{ AND } \ldots \texttt{ AND } F_p)]\!](s) \end{array}}{se, \{V_1,\ldots,V_o\} \vdash I_r = I_m\,(E_1,\ldots,E_p) \checkmark F}$$

Figure 6.24: Well-Definedness of Method Calls

case, if the method throws an exception, the poststate value of the variable remains unchanged.

- The rule in Figure 6.26 is applicable if I_r is a global variable that also appears in the frame of the method ($I_r \in\in \{I_1,\ldots,I_n\}\backslash\{V_1,\ldots,V_o\}$). In this case, if the method throws an exception, the poststate value of I_r is the value that the variable has received by the method call.

The soundness of these rules is shown in the following subsections.

6.5.1 Well-Definedness of Method Calls

$$\frac{\begin{array}{l} se(I_m) = \langle (J_1,\ldots,J_p), \langle (I_1,\ldots,I_n), (K_1,\ldots,K_m), F_C, F_R, T\rangle\rangle \\ E_1 \overset{\mathrm{D}}{\simeq} F_1, \ldots, E_P \overset{\mathrm{D}}{\simeq} F_p \\ \forall s \in State: \\ \quad [\![(\texttt{now.executes AND } F) \texttt{ => } (F_1 \texttt{ AND } \ldots \texttt{ AND } F_p)]\!](s) \end{array}}{se, \{V_1,\ldots,V_o\} \vdash I_r = I_m\,(E_1,\ldots,E_p) \checkmark F}$$

For proving the soundness of this rule, we first have to give method calls a valuation in the semantics with undefined expressions:

Method Calls: Verification

$se(I_m) = \langle (J_1,\ldots,J_p), \langle (I_1,\ldots,I_n), (K_1,\ldots,K_m), F_C, F_R, T\rangle\rangle$
$E_1 \simeq T_1, \ldots, E_p \simeq T_p$
$L_1,\ldots,L_p$ is a renaming of $J_1,\ldots,J_p$
$\{J_1,\ldots,J_p\} \cap \{L_1,\ldots,L_p\} = \emptyset$
$\{J_1,\ldots,J_p,L_1,\ldots,L_p\} \cap \{R\} = \emptyset$
$I_r \in \{V_1,\ldots,V_o\} \vee I_r \notin \{I_1,\ldots,I_n\}$
$\$J_1,\ldots,\$J_p,\$L_1,\ldots,\$L_p,\$R$ do not occur in F_C, F_R

$se, \{V_1,\ldots,V_o\} \vdash I_r = I_m\,(E_1,\ldots,E_p)$:

$$
\begin{array}{l}
\big[\texttt{EXISTS}\ \$J_1,\ldots,\$J_p,\$L_1,\ldots,\$L_p,\$R\texttt{:} \\
\quad \$J_1{=}T_1\ \texttt{AND}\ \ldots\ \texttt{AND}\ \$J_p{=}T_p\ \texttt{AND} \\
\quad (F_C \texttt{ => } F_R) \\
\qquad [\$J_1/J_1,\ldots,\$J_p/J_p,\$L_1/J_1\texttt{'},\ldots,\$L_p/J_p\texttt{'}][\$R/I_r\texttt{'}] \\
\qquad [?V_1/V_1,\ldots,?V_o/V_o,?V_1\texttt{'}/V_1\texttt{'},\ldots,?V_o\texttt{'}/V_o\texttt{'}]\ \texttt{AND} \\
\quad \texttt{IF next.executes} \\
\qquad \texttt{THEN}\ I_r\texttt{'} = \texttt{next.value} \\
\qquad \texttt{ELSE}\ I_r\texttt{'} = I_r \\
\big]^{\texttt{FALSE,FALSE,FALSE,}\{K_1,\ldots,K_m\}}_{(\{I_1,\ldots,I_n\}\setminus\{V_1,\ldots,V_o\})\cup\{?I{:}I\in\{I_1,\ldots,I_n\}\cap\{V_1,\ldots,V_o\}\}\cup\{I_r\}}
\end{array}
$$

Figure 6.25: Verification of Method Calls (1/2)

Method Calls: Verification

$se(I_m) = \langle (J_1,\ldots,J_p), \langle (I_1,\ldots,I_n), (K_1,\ldots,K_m), F_C, F_R, T\rangle\rangle$
$E_1 \simeq T_1, \ldots, E_p \simeq T_p$
$L_1,\ldots,L_p$ is a renaming of $J_1,\ldots,J_p$
$\{J_1,\ldots,J_p\} \cap \{L_1,\ldots,L_p\} = \emptyset$
$\{J_1,\ldots,J_p,L_1,\ldots,L_p\} \cap \{R\} = \emptyset$
$I_r \in \{I_1,\ldots,I_n\} \backslash \{V_1,\ldots,V_o\}$
$\$J_1,\ldots,\$J_p,\$L_1,\ldots,\$L_p,\$R$ do not occur in F_C, F_R

$se, \{V_1,\ldots,V_o\} \vdash I_r = I_m\,(E_1,\ldots,E_p)$:
[EXISTS $\$J_1,\ldots,\$J_p,\$L_1,\ldots,\$L_p,\$R$:
$\$J_1$=$T_1$ AND ... AND $\$J_p$=$T_p$ AND
$(F_C$ => $F_R)$
$[\$J_1/J_1,\ldots,\$J_p/J_p,\$L_1/J_1\text{'},\ldots,\$L_p/J_p\text{'}][\$R/I_r\text{'}]$
$[?V_1/V_1,\ldots,?V_o/V_o,?V_1\text{'}/V_1\text{'},\ldots,?V_o\text{'}/V_o\text{'}]$ AND
IF next.executes
THEN $I_r\text{'}$ = next.value
ELSE $I_r\text{'}$ = $\$R$
$]^{\text{FALSE,FALSE,FALSE,}\{K_1,\ldots,K_m\}}_{(\{I_1,\ldots,I_n\}\backslash\{V_1,\ldots,V_o\})\cup\{?I:I\in\{I_1,\ldots,I_n\}\cap\{V_1,\ldots,V_o\}\}}$

Figure 6.26: Verification of Method Calls (2/2)

Method Calls: Termination

$se(I_m) = \langle (J_1,\ldots,J_p), \langle (I_1,\ldots,I_n), (K_1,\ldots,K_m), F_C, F_R, T\rangle\rangle$
$E_1 \simeq T_1, \ldots, E_p \simeq T_p$
$L_1,\ldots,L_p$ is a renaming of $J_1,\ldots,J_p$
$\$L_1,\ldots,\$L_p$ do not occur in F_C
$\forall s \in State$:
$[\![$now.executes AND F =>
FORALL $\$L_1,\ldots,\$L_p$:
$\$L_1$=$T_1$ AND ... AND $\$L_p$=$T_p$ =>
$F_C[\$L_1/J_1,\ldots,\$L_p/J_p,\$L_1/J_1\text{'},\ldots,\$L_p/J_p\text{'}]$
$[?V_1/V_1,\ldots,?V_o/V_o,?V_1\text{'}/V_1\text{'},\ldots,?V_o\text{'}/V_o\text{'}]]\!](s)$

$se, \{V_1,\ldots,V_o\} \vdash I_r = I_m\,(E_1,\ldots,E_p) \downarrow F$

Figure 6.27: Termination of Method Calls

$$
\begin{array}{l}
[\![I_r = I_m\ (E_1, \ldots, E_p)\ ]\!]^{c,me}_{\perp}(s,s') \Leftrightarrow \\
\quad \text{LET } v_1 = [\![E_1]\!]^{c}_{\perp}(s) \text{ IN} \\
\quad \ldots \\
\quad \text{LET } v_p = [\![E_p]\!]^{c}_{\perp}(s) \text{ IN} \\
\quad \text{IF } v_1 = \perp \vee \ldots \vee v_p = \perp \text{ THEN} \\
\qquad s' = \mathit{expthrow}(s) \\
\quad \text{ELSE} \\
\qquad \text{LET } \langle v, b \rangle = \mathit{me}(I_m) \text{ IN} \\
\qquad \text{LET } \langle r, t \rangle = b^{\mathit{call}(v,c)}(v_1, \ldots, v_p) \text{ IN} \\
\qquad \exists s_0 \in \mathit{State} : r(s, s_0) \wedge \\
\qquad\quad \text{IF } \mathit{throws}(\mathit{control}(s_0)) \\
\qquad\qquad \text{THEN } s' = s_0 \\
\qquad\qquad \text{ELSE } s' = \mathit{write}(s_0, I_r, \mathit{value}(\mathit{control}(s_0)))^c
\end{array}
$$

Based on this definition, the soundness of the rule is shown below.

Proof Take $c, c' \in \mathit{Context}, \mathit{me} \in \mathit{MethodEnv}, s, s' \in \mathit{State}, e \in \mathit{Environmet}$ and assume

(2) F has no free (mathematical or state) variables

(3) F does not depend on the poststate

(4) $\mathit{specifies}(se, \mathit{me}, c)$

(5) $\mathit{pushes}(c, c', \{V_1, \ldots, V_o\})$

(6) $\mathit{executes}(\mathit{control}(s))$

(7) $[\![F]\!]^{c,c'}(s)$

We have to show

(a.1) $[\![I_r = I_m\ (E_1, \ldots, E_p)\ ]\!]^{c',me}(s,s') \Rightarrow [\![I_r = I_m\ (E_1, \ldots, E_p)\ ]\!]^{c',me}_{\perp}(s,s')$

(a.2) $[\![I_r = I_m\ (E_1, \ldots, E_p)\ ]\!]^{c',me}_{\perp}(s,s') \Rightarrow [\![I_r = I_m\ (E_1, \ldots, E_p)\ ]\!]^{c',me}(s,s')$

From the hypotheses, we know

(8) $\mathit{se}(I_m) = \langle (J_1, \ldots, J_p), \langle (I_1, \ldots, I_n), (K_1, \ldots, K_m), F_C, F_R, T \rangle \rangle$

(9) $E_1 \overset{\mathrm{D}}{\simeq} F_1, \ldots, E_P \overset{\mathrm{D}}{\simeq} F_p$

(10) $\forall s \in \mathit{State}$:
$[\![(\texttt{now.executes AND } F) \texttt{ => } (F_1 \texttt{ AND } \ldots \texttt{ AND } F_p)]\!](s)$

From (5) and the definition of *pushes*, we know

(11) $\mathit{DifferentVariables}(c)$

(12) $\mathit{DifferentVariables}(c')$

From (6), (10), (11), (12) and the definition of $[\![\_]\!]$, we know

(13) $[\![F]\!]^{c,c'}(s) \Rightarrow [\![F_1]\!]^{c,c'}(s) \wedge \ldots \wedge [\![F_p]\!]^{c,c'}(s)$

From (7) and (13), we know

(14) $[\![F_1]\!]^{c,c'}(s) \wedge \ldots \wedge [\![F_p]\!]^{c,c'}(s)$

From (9), (11), (12), and the definition of $\overset{\mathrm{D}}{\simeq}$, we know

(15) $([\![E_1]\!]^{c'}_{\mathrm{D}}(s) \Leftrightarrow [\![F_1]\!]^{c,c'}(s)) \wedge \ldots ([\![E_p]\!]^{c'}_{\mathrm{D}}(s) \Leftrightarrow [\![F_p]\!]^{c,c'}(s))$

From (11), (14) and (15), we know

(16) $[\![E_1]\!]^{c'}_{\mathrm{D}}(s) \wedge \ldots \wedge [\![E_p]\!]^{c'}_{\mathrm{D}}(s)$

From (16) and the definition of $[\![\_]\!]_{\perp}$, we know

(17) $[\![E_1]\!]^{c'}_{\perp}(s) = [\![E_1]\!]^{c'}(s) \wedge \ldots \wedge [\![E_p]\!]^{c'}_{\perp}(s) = [\![E_p]\!]^{c'}(s)$

To show (a.1), we take $v \in \mathit{View}, b \in \mathit{Behavior}, r \in \mathit{StateRelation}$ as well as $t \in \mathit{StateCondition}$ and $s_0 \in \mathit{State}$ such that

(18) $\langle v, b \rangle = \mathit{me}(I_m)$

(19) $\langle r, t \rangle = b^{\mathit{call}(v,c')}([\![E_1]\!]^{c'}(s), \ldots, [\![E_p]\!]^{c'}(s))$

(20) $r(s, s_0)$

(21)
IF $\mathit{throws}(\mathit{control}(s_0))$
THEN $s' = s_0$
ELSE $s' = \mathit{write}(s_0, I_r, \mathit{value}(\mathit{control}(s_0)))^{c'}$

We also define $v_1, \ldots, v_p \in \mathit{Value}$ such that

(22) $v_1 = [\![E_1]\!]^{c'}_{\perp}(s) \wedge \ldots \wedge v_p = [\![E_p]\!]^{c'}_{\perp}(s)$

and take $r_0 \in \mathit{StateRelation}, t_0 \in \mathit{StateCondition}$ such that

(23) $\langle r_0, t_0 \rangle = b^{\mathit{call}(v,c')}(v_1, \ldots, v_p)$

It suffices to show

(a.1.a.1) $r_0(s, s_0)$

IF $throws(control(s_0))$

(a.1.a.2) THEN $s' = s_0$

ELSE $s' = write(s_0, I_r, value(control(s_0)))^{c'}$

From (17), (19), (20), (22), and (23), we know (a.1.a.1).

From (21) we know (a.1.a.2).

To show (a.2), we define $v_1, \ldots, v_p \in Value$ such that

(24) $v_1 = [\![E_1]\!]^{c'}_{\perp}(s) \wedge \ldots \wedge v_p = [\![E_p]\!]^{c'}_{\perp}(s)$

IF $v_1 = \perp \vee \ldots \vee v_p = \perp$ THEN

$s' = expthrow(s)$

ELSE

LET $\langle v, b \rangle = me(I_m)$ IN

(25) LET $\langle r, t \rangle = b^{call(v,c')}(v_1, \ldots, v_p)$ IN

$\exists s_0 \in State : r(s, s_0) \wedge$

IF $throws(control(s_0))$

THEN $s' = s_0$

ELSE $s' = write(s_0, I_r, value(control(s_0)))^{c'}$

and take $r \in StateRelation, t \in StateCondition$ such that

(26) $\langle r, t \rangle = b^{call(v,c')}([\![E_1]\!]^{c'}(s), \ldots, [\![E_p]\!]^{c'}(s))$

It suffices to show

(a.2.a.1) $r(s, s_0)$

IF $throws(control(s_0))$

(a.2.a.2) THEN $s' = s_0$

ELSE $s' = write(s_0, I_r, value(control(s_0)))^{c'}$

From (17) and (24), we know

(27) $v_1 \neq \perp \wedge \ldots \wedge v_p \neq \perp$

From (18), (25), and (27), we have some $r_0 \in StateRelation$, $t_0 \in StateCondition$, and $s_0 \in State$ with

(28) $\langle r_0, t_0 \rangle = b^{call(v,c')}(v_1, \ldots, v_p)$

(29) $r_0(s, s_0)$

IF $throws(control(s_0))$

(30) THEN $s' = s_0$

ELSE $s' = write(s_0, I_r, value(control(s_0)))^{c'}$

From (17), (24), (26), (28), and (29), we know (a.2.a.1).

From (25), we know (a.2.a.2). □

6.5.2 Verification of Method Calls (Rule 1)

$$
\begin{array}{l}
se(I_m) = \langle (J_1,\ldots,J_p), \langle (I_1,\ldots,I_n), (K_1,\ldots,K_m), F_C, F_R, T\rangle\rangle \\
E_1 \simeq T_1, \ldots, E_p \simeq T_p \\
L_1,\ldots,L_p \text{ is a renaming of } J_1,\ldots,J_p \\
\{J_1,\ldots,J_p\} \cap \{L_1,\ldots,L_p\} = \emptyset \\
\{J_1,\ldots,J_p,L_1,\ldots,L_p\} \cap \{R\} = \emptyset \\
I_r \in \{V_1,\ldots,V_o\} \vee I_r \notin \{I_1,\ldots,I_n\} \\
\$J_1,\ldots,\$J_p,\$L_1,\ldots,\$L_p,\$R \text{ do not occur in } F_C, F_R \\
\hline
se, \{V_1,\ldots,V_o\} \vdash I_r = I_m\,(E_1,\ldots,E_p) : \\
\quad [\texttt{EXISTS } \$J_1,\ldots,\$J_p,\$L_1,\ldots,\$L_p,\$R\texttt{:} \\
\quad\quad \$J_1{=}T_1 \texttt{ AND } \ldots \texttt{ AND } \$J_p{=}T_p \texttt{ AND} \\
\quad\quad (F_C \texttt{ => } F_R) \\
\quad\quad\quad [\$J_1/J_1,\ldots,\$J_p/J_p,\$L_1/J_1\texttt{'},\ldots,\$L_p/J_p\texttt{'}][\$R/I_r\texttt{'}] \\
\quad\quad\quad [?V_1/V_1,\ldots,?V_o/V_o,?V_1\texttt{'}/V_1\texttt{'},\ldots,?V_o\texttt{'}/V_o\texttt{'}] \texttt{ AND} \\
\quad\quad \texttt{IF next.executes} \\
\quad\quad\quad \texttt{THEN } I_r\texttt{'} = \texttt{next.value} \\
\quad\quad\quad \texttt{ELSE } I_r\texttt{'} = I_r \\
\quad]^{\texttt{FALSE,FALSE,FALSE,}\{K_1,\ldots,K_m\}}_{(\{I_1,\ldots,I_n\}\setminus\{V_1,\ldots,V_o\})\cup\{?I{:}I\in\{I_1,\ldots,I_n\}\cap\{V_1,\ldots,V_o\}\}\cup\{I_r\}}
\end{array}
$$

The proof of the soundness of the rule is given below.

Proof Take arbitrary $c, c' \in Context$, $me \in MethodEnv$, $s, s' \in State$, as well as $e \in Environment$ and assume

(1) $specifies(se, me, c)$

(2) $pushes(c, c', \{V_1,\ldots,V_o\})$

(3) $executes(control(s))$

(4) $[\![I_r = I_m\,(E_1,\ldots,E_p)]\!]^{c',me}(s, s')$

We define

$$
\begin{array}{ll}
 & F := \\
 & \quad (\texttt{EXISTS } \$J_1,\ldots,\$J_p,\$L_1,\ldots,\$L_p,\$R\texttt{:} \\
 & \qquad \$J_1{=}T_1 \texttt{ AND } \ldots \texttt{ AND } \$J_p{=}T_p \texttt{ AND} \\
 & \qquad (F_C \texttt{ => } F_R) \\
 & \qquad\quad [\$J_1/J_1,\ldots,\$J_p/J_p,\$L_1/J_1\texttt{'},\ldots,\$L_p/J_p\texttt{'}][\$R/I_r\texttt{'}] \\
 & \qquad\quad [?V_1/V_1,\ldots,?V_o/V_o,?V_1\texttt{'}/V_1\texttt{'},\ldots,?V_o\texttt{'}/V_o\texttt{'}] \texttt{ AND} \\
 & \qquad \texttt{IF next.executes} \\
(5) & \qquad\quad \texttt{THEN } I_r\texttt{'} = \texttt{next.value} \\
 & \qquad\quad \texttt{ELSE } I_r\texttt{'} = I_r) \texttt{ AND} \\
 & \quad \texttt{writesonly } (\{I_1,\ldots,I_n\}\backslash\{V_1,\ldots,V_o\}) \cup \\
 & \qquad \{?I : I \in \{I_1,\ldots,I_n\} \cap \{V_1,\ldots,V_o\}\} \cup \{I_r\} \texttt{ AND} \\
 & \quad (\texttt{next.continues => FALSE}) \texttt{ AND} \\
 & \quad (\texttt{next.breaks => FALSE}) \texttt{ AND} \\
 & \quad (\texttt{next.returns => FALSE}) \texttt{ AND} \\
 & \quad (\texttt{next.throws =>} \\
 & \qquad (\texttt{next.throws } K_1 \texttt{ OR } \ldots \texttt{ OR next.throws } K_n))
\end{array}
$$

We have to show

(a.1) F has no free (mathematical or state) variables

(a.2) $[\![F]\!]^{c,c'}(e)(s,s')$

From the hypotheses, we know

(6) $se(I_m) = \langle (J_1,\ldots,J_p), \langle (I_1,\ldots,I_n),(K_1,\ldots,K_m),F_C,F_R,T\rangle\rangle$

(7) $E_1 \simeq T_1,\ldots,E_p \simeq T_p$

(8) $L_1,\ldots,L_p$ is a renaming of $J_1,\ldots,J_p$

(9) $\{J_1,\ldots,J_p\} \cap \{L_1,\ldots,L_p\} = \emptyset$

(9a) $\{J_1,\ldots,J_p,L_1,\ldots,L_p\} \cap \{R\} = \emptyset$

(10) $I_r \in \{V_1,\ldots,V_o\} \vee I_r \notin \{I_1,\ldots,I_n\}$

(11) $\$J_1,\ldots,\$J_p,\$L_1,\ldots,\$L_p,\$R$ do not occur in F_C,F_R

We define

(12) $c'' := call(view(c),c')$

(13) $v_1 := [\![E_1]\!]^{c'}(s),\ldots,v_p := [\![E_p]\!]^{c'}(s)$

From (4), (12), (13), and the definition of $[\![\_]\!]$, we know for some $v \in View, b \in Behavior, r \in StateRelation, t \in StateCondition, s_0 \in State$

(14) $\langle v, b \rangle = me(I_m)$

(15) $\langle r, t \rangle = b^{call(v,c')}(v_1, \ldots, v_p)$

(16) $r(s, s_0)$

(17) IF $throws(control(s_0))$
THEN $s' = s_0$
ELSE $s' = write(s_0, I_r, value(control(s_0)))^{c'}$

From (2) and the definition of *pushes*, we know

(18) $DifferentVariables(c)$

(19) $DifferentVariables(c')$

(21) $c = c'$ EXCEPT $V_1, \ldots, V_o$

(21a) $\{[\![V_1]\!]^{c'}, \ldots, [\![V_o]\!]^{c'}\} \subseteq space(c)$

From (2), (12), and (COC), we know

(20) $DifferentVariables(c'')$

From (1), (14) and the definition of *specifies*, we know

(22) $v = view(c)$

(23) $specifies(se(I_m), b, c)$

From (12), the definitions of *call* and *view*, and (COV), we know

(23a) c EQUALS c''

From (2), (12), and the definitions of *pushes*, *call*, and *view*, we know

(23b) $space(c'') \subseteq space(c)$

We define

(24) $c''' := push(c'', J_1, \ldots, J_p)$

(25) $s_1 := writes(s, J_1, v_1, \ldots, J_p, v_p)^{c'''}$

From (3), (12), (15), (16), (20), (22), (23), (23a), (23b), (24), (25), and finally the definition of *specifies*, we know

(26) $\{J_1,\ldots,J_p\} \cap \{I_1,\ldots,I_n\} = \emptyset$

(27) F_C, F_R and T have no free (mathematical or state) variables

(28) F_C and T do not depend on the poststate

(29) $[\![F_C \Rightarrow F_R]\!]^{c,c'''}(e)(s_1,s_0)$

(29a) $s = s_0$ EXCEPT $range(c'') \cup \text{range}(view(c))$

(30) $s = s_0$ EXCEPT^{c,c} $I_1,\ldots,I_n$

(31) $executes(control(s_0)) \vee throws(control(s_0))$

(32) $throws(control(s_0)) \Rightarrow key(control(s_0)) \in \{K_1,\ldots,K_m\}$

From (5) and (27), we know (a.1).

From (5), to show (a.2), it suffices to show

(a.2.1)
$[\![$ EXISTS $\$J_1,\ldots,\$J_p,\$L_1,\ldots,\$L_p,\$R$:
  $\$J_1$=$T_1$ AND ... AND $\$J_p$=$T_p$ AND
  $(F_C \Rightarrow F_R)$
    $[\$J_1/J_1,\ldots,\$J_p/J_p,\$L_1/J_1\text{'},\ldots,\$L_p/J_p\text{'}][\$R/L_r\text{'}]$
    $[?V_1/V_1,\ldots,?V_o/V_o,?V_1\text{'}/V_1\text{'},\ldots,?V_o\text{'}/V_o\text{'}]$ AND
  IF next.executes
    THEN $I_r\text{'}$ = next.value
    ELSE $I_r\text{'} = I_r]\!]^{c,c'}(e)(s,s')$

(a.2.2) $s = s'$ EXCEPT$^{c,c'}(\{I_1,\ldots,I_n\}\setminus\{V_1,\ldots,V_o\}) \cup$
  $\{?I : I \in \{I_1,\ldots,I_n\} \cap \{V_1,\ldots,V_o\}\} \cup \{I_r\}$

(a.2.3) $\neg continues(control(s'))$

(a.2.4) $\neg breaks(control(s'))$

(a.2.5) $\neg returns(control(s'))$

(a.2.6) $throws(control(s')) \Rightarrow key(control(s')) \in \{K_1,\ldots,K_m\}$

From (17), (31), (32), (CW), and Lemma "State Control Predicates", we know (a.2.3), (a.2.4), (a.2.5), and (a.2.6).

We define

(33) $w_1 := read(s_0,J_1)^{c'''},\ldots w_p := read(s_0,J_p)^{c'''}, u := read(s_0,I_r)^{c'''}$

(34) $e_0 = e[J_1 \mapsto v_1,\ldots,J_p \mapsto v_p, L_1 \mapsto w_1,\ldots,L_p \mapsto w_p, R \mapsto u]$

To show (a.2.1), from (34) and the definition of $[\![\_]\!]$, it suffices to show

(a.2.1.1) $v_1 = [\![T_1]\!]^{c,c'}(e_0)(s,s') \wedge \ldots \wedge v_p = [\![T_p]\!]^{c,c'}(e_0)(s,s')$

$$
\begin{array}{ll}
 & [\![(F_C \Rightarrow F_R) \\
\text{(a.2.1.2)} & \quad [\$J_1/J_1, \ldots, \$J_p/J_p, \$L_1/J_1\,', \ldots, \$L_p/J_p\,'\,][\$R/I_r\,'\,] \\
 & \quad [?V_1/V_1, \ldots, ?V_o/V_o, ?V_1\,'/V_1\,', \ldots, ?V_o\,'/V_o\,'\,]]\!]^{c,c'}(e_0)(s,s') \\
 & \text{IF } \mathit{executes}(\mathit{control}(s')) \\
\text{(a.2.1.3)} & \quad \text{THEN } \mathit{read}(s',I_r)^{c'} = \mathit{value}(\mathit{control}(s')) \\
 & \quad \text{ELSE } \mathit{read}(s',I_r)^{c'} = \mathit{read}(s,I_r)^{c'}
\end{array}
$$

From (7), (13), (18), (19), and the definition of $\simeq$, we know (a.2.1.1).

From (25) and (WSE), we know

$$(35) \quad s_1 = s \text{ EXCEPT}^{c,c'''} J_1, \ldots, J_p$$

From (25) and (RWE), we know

$$(36) \quad v_1 = \mathit{read}(s_1, J_1)^{c'''} \wedge \ldots \wedge v_p = \mathit{read}(s_1, J_p)^{c'''}$$

From (8), (11), (29), (33), (35), (36), (REE), (PMVF1"), and finally (PMVF2"), we know

$$
\begin{array}{ll}
 & [\![(F_C \Rightarrow F_R) \\
(37) & \quad [\$J_1/J_1, \ldots, \$J_p/J_p, \$L_1/J_1\,', \ldots, \$L_p/J_p\,'\,] \\
 & \quad [\$R/I_r\,'\,]]\!]^{c,c'''}(e_0)(s,s_0)
\end{array}
$$

From (20), (24), (COP), and the definition of *pushes*, we know

$$(38) \quad c'' = c''' \text{ EXCEPT } J_1, \ldots, J_p$$

From (37), (38), (MPVF0'), (MPVF1'), and (COF1), we know

$$
\begin{array}{ll}
 & [\![(F_C \Rightarrow F_R) \\
(39) & \quad [\$J_1/J_1, \ldots, \$J_p/J_p, \$L_1/J_1\,', \ldots, \$L_p/J_p\,'\,] \\
 & \quad [\$R/I_r\,'\,]]\!]^{c,c''}(e_0)(s,s_0)
\end{array}
$$

From (12), the definitions of *call* and *view*, and (COV), we know

$$(40) \quad c \text{ EQUALS } c''$$

From (40) and the definitions of EQUALS and AT, we know

$$(41) \quad c = c'' \text{ AT } V_1, \ldots, V_o$$

From (21), (40), and the definitions of EQUALS and EXCEPT, we know

(42) $c' = c''$ EXCEPT $V_1, \ldots, V_o$

From (39), (41), (42), and (PMGF), we know

(43) $[\![(F_C \Rightarrow F_R)$
$[\$J_1/J_1, \ldots, \$J_p/J_p, \$L_1/J_1\prime, \ldots, \$L_p/J_p\prime\,][\$R/I_r\prime\,]$
$[?V_1/V_1, \ldots, ?V_o/V_o, ?V_1\prime/V_1\prime, \ldots, ?V_o\prime/V_o\prime\,]]^{c,c'}(e_0)(s, s_0)$

From (17) and (WS), we know

(44) $s' = s_0$ EXCEPT$^{c'}$ I_r

From (17) and (CW), we know

(45) $control(s') = control(s_0)$

From (43), (44), (45), (MPVF1'), and (PVF4'), we know (a.2.1.2).

To show (a.2.1.3), we show

(a.2.1.3.1) $executes(control(s')) \Rightarrow read(s', I_r)^{c'} = value(control(s'))$

(a.2.1.3.2) $\neg executes(control(s')) \Rightarrow read(s', I_r)^{c'} = read(s, I_r)^{c'}$

To show (a.2.1.3.1), we assume

(46) $executes(control(s'))$

and show

(a.2.1.3.1.a) $read(s', I_r)^{c'} = value(control(s'))$

From (17), (45), (46), (RW1), and Lemma "State Control Predicates", we know (a.2.1.3.1.a).

To show (a.2.1.3.2), we assume

(47) $\neg executes(control(s'))$

and show

(a.2.1.3.2.a) $read(s', I_r)^{c'} = read(s, I_r)^{c'}$

From (47), (a.2.4), (a.2.5), (a.2.6) (all shown above), and Lemma "State Control Predicates", we know

(48) $\mathit{throws}(\mathit{control}(s'))$

From (17), (45), and (48), we know

(49) $s' = s_0$

We proceed by case distinction.

In the first case, we assume

(50) $I_r \in \{V_1, \ldots, V_o\}$

From (29a), (49), and the definitions of EXCEPT and *read*, it suffices to show

(a.2.1.3.2.b) $[\![I_r]\!]^{c'} \notin \mathit{range}(c'') \cup \mathrm{range}(\mathit{view}(c))$

From (19) and the definition of *DifferentVariables*, we know

(51) $[\![I_r]\!]^{c'} \notin \mathit{space}(c')$

From (12), (51), and the definitions of *call* and *space*, it suffices to show

(a.2.1.3.2.c) $[\![I_r]\!]^{c'} \notin \mathrm{range}(\mathit{view}(c))$

From (18) and the definition of *DifferentVariables*, it suffices to show

(a.2.1.3.2.d) $[\![I_r]\!]^{c'} \in \mathit{space}(c)$

From (21a) and (50), we know (a.2.1.3.2.d).

In the second case, we assume

(52) $I_r \notin \{V_1, \ldots, V_o\}$

From (10) and (52), we know

(53) $I_r \notin \{I_1, \ldots, I_n\}$

From (30), (53), and (RSE), we know

(54) $\mathit{read}(s, I_r)^c = \mathit{read}(s_0, I_r)^c$

From (21), (52), and the definition of EXCEPT, we know

(55) $[\![I_r]\!]^c = [\![I_r]\!]^{c'}$

From (49), (54), (55), and the definition of *read*, we know (a.2.1.3.2.a).

To show (a.2.2), from the definition of EXCEPT, it suffices to take arbitrary $x \in$ Variable with

(56) $x \notin \{[\![I]\!]^{c'} : I \in \{I_1, \ldots, I_n\} \backslash \{V_1, \ldots, V_o\}\}$

(57) $x \notin \{[\![I]\!]^{c} : I \in \{I_1, \ldots, I_n\} \cap \{V_1, \ldots, V_o\}\}$

(58) $x \neq [\![I_r]\!]^{c'}$

(59) $store(s)(x) \neq store(s')(x)$

and show a contradiction.

From (56) and (57), we know

(60) $x \notin \{[\![I_1]\!]^{c'}, \ldots, [\![I_n]\!]^{c'}\} \vee x \in \{[\![V_1]\!]^{c'}, \ldots, [\![V_o]\!]^{c'}\}$

(61) $x \notin \{[\![I_1]\!]^{c}, \ldots, [\![I_n]\!]^{c}\} \vee x \notin \{[\![V_1]\!]^{c}, \ldots, [\![V_o]\!]^{c}\}$

From (17), (58), and the definition of *write*, we know

(62) $store(s')(x) = store(s_0)(x)$

From (60), we have two cases.

In the first case, we assume

(63) $x \in \{[\![V_1]\!]^{c'}, \ldots, [\![V_o]\!]^{c'}\}$

From (19), (63), and the definition of *DifferentVariables*, we know

(64) $x \notin space(c')$

From (2), (63) and the definition of *pushes*, we know

(65) $x \in space(c)$

From (18), (65), and the definition of *DifferentVariables*, we know

(66) $x \notin \text{range}(view(c))$

From (12), (64), (66), and the definitions of *call* and *range*, we know

(67) $x \notin range(c'')$

From (29a), (66), (67), and the definition of EXCEPT, we know

(68) $store(s)(x) = store(s_0)(x)$

But (62) and (68) contradict (59).

In the second case, we assume

(69) $x \notin \{[\![V_1]\!]^{c'}, \ldots, [\![V_o]\!]^{c'}\}$

(70) $x \notin \{[\![I_1]\!]^{c'}, \ldots, [\![I_n]\!]^{c'}\}$

From (61), we have two subcases.

In the first subcase, we assume

(71) $x \notin \{[\![I_1]\!]^{c}, \ldots, [\![I_n]\!]^{c}\}$

From (30), (71), and the definition of EXCEPT, we know

(72) $store(s)(x) = store(s_0)(x)$

But (62) and (72) contradict (59).

In the second subcase, we assume

(73) $x \in \{[\![I_1]\!]^{c}, \ldots, [\![I_n]\!]^{c}\}$

(74) $x \notin \{[\![V_1]\!]^{c}, \ldots, [\![V_o]\!]^{c}\}$

From (73) and (74), we have some $I \in$ Identifier such that

(75) $x = [\![I]\!]^{c}$

(76) $I = I_1 \vee \ldots \vee I = I_n$

(77) $I \neq V_1 \wedge \ldots \wedge I \neq V_o$

From (21), (75), (77), and the definition of EXCEPT, we know

(78) $x = [\![I]\!]^{c'}$

But (76) and (78) contradict (69). □

6.5.3 Verification of Method Calls (Rule 2)

$$
\frac{\begin{array}{l} se(I_m) = \langle (J_1,\ldots,J_p), \langle (I_1,\ldots,I_n), (K_1,\ldots,K_m), F_C, F_R, T\rangle\rangle \\ E_1 \simeq T_1, \ldots, E_p \simeq T_p \\ L_1,\ldots,L_p \text{ is a renaming of } J_1,\ldots,J_p \\ \{J_1,\ldots,J_p\} \cap \{L_1,\ldots,L_p\} = \emptyset \\ \{J_1,\ldots,J_p,L_1,\ldots,L_p\} \cap \{R\} = \emptyset \\ I_r \in \{I_1,\ldots,I_n\} \backslash \{V_1,\ldots,V_o\} \\ \$J_1,\ldots,\$J_p,\$L_1,\ldots,\$L_p,\$R \text{ do not occur in } F_C, F_R \end{array}}{\begin{array}{l} se, \{V_1,\ldots,V_o\} \vdash I_r = I_m\,(E_1,\ldots,E_p) : \\ \quad [\texttt{EXISTS } \$J_1,\ldots,\$J_p,\$L_1,\ldots,\$L_p,\$R\texttt{:} \\ \quad\quad \$J_1\texttt{=}T_1 \texttt{ AND } \ldots \texttt{ AND } \$J_p\texttt{=}T_p \texttt{ AND} \\ \quad\quad (F_C \texttt{ => } F_R) \\ \quad\quad\quad [\$J_1/J_1,\ldots,\$J_p/J_p,\$L_1/J_1\texttt{'},\ldots,\$L_p/J_p\texttt{'}][\$R/I_r\texttt{'}] \\ \quad\quad\quad [?V_1/V_1,\ldots,?V_o/V_o,?V_1\texttt{'}/V_1\texttt{'},\ldots,?V_o\texttt{'}/V_o\texttt{'}] \texttt{ AND} \\ \quad\quad \texttt{IF next.executes} \\ \quad\quad\quad \texttt{THEN } I_r\texttt{'} = \texttt{next.value} \\ \quad\quad\quad \texttt{ELSE } I_r\texttt{'} = \$R \\ \quad]^{\texttt{FALSE,FALSE,FALSE,}\{K_1,\ldots,K_m\}}_{(\{I_1,\ldots,I_n\}\backslash\{V_1,\ldots,V_o\})\cup\{?I:I\in\{I_1,\ldots,I_n\}\cap\{V_1,\ldots,V_o\}\}} \end{array}}
$$

The proof of the soundness of the rule is given below.

Proof Take arbitrary $c, c' \in \mathit{Context}$, $me \in \mathit{MethodEnv}$, $s, s' \in \mathit{State}$, as well as $e \in \mathit{Environment}$ and assume

(1) $\mathit{specifies}(se, me, c)$

(2) $\mathit{pushes}(c, c', \{V_1,\ldots,V_o\})$

(3) $\mathit{executes}(\mathit{control}(s))$

(4) $[\![I_r = I_m\,(E_1,\ldots,E_p)]\!]^{c',me}(s,s')$

We define

$$
\begin{array}{l}
F := \\
\quad (\texttt{EXISTS } \$J_1,\ldots,\$J_p,\$L_1,\ldots,\$L_p,\$R\texttt{:} \\
\qquad \$J_1{=}T_1 \texttt{ AND } \ldots \texttt{ AND } \$J_p{=}T_p \texttt{ AND} \\
\qquad (F_C \texttt{ => } F_R) \\
\qquad\quad [\$J_1/J_1,\ldots,\$J_p/J_p,\$L_1/J_1\texttt{'},\ldots,\$L_p/J_p\texttt{'}][\$R/I_r\texttt{'}] \\
\qquad\quad [?V_1/V_1,\ldots,?V_o/V_o,?V_1\texttt{'}/V_1\texttt{'},\ldots,?V_o\texttt{'}/V_o\texttt{'}] \texttt{ AND} \\
\qquad \texttt{IF next.executes} \\
\qquad\qquad \texttt{THEN } I_r\texttt{'} = \texttt{next.value} \\
\qquad\qquad \texttt{ELSE } I_r\texttt{'} = \$R) \texttt{ AND} \\
\quad \texttt{writesonly } (\{I_1,\ldots,I_n\}\backslash\{V_1,\ldots,V_o\}) \cup \\
\qquad \{?I : I \in \{I_1,\ldots,I_n\} \cap \{V_1,\ldots,V_o\}\} \texttt{ AND} \\
\quad (\texttt{next.continues => FALSE}) \texttt{ AND} \\
\quad (\texttt{next.breaks => FALSE}) \texttt{ AND} \\
\quad (\texttt{next.returns => FALSE}) \texttt{ AND} \\
\quad (\texttt{next.throws =>} \\
\qquad (\texttt{next.throws } K_1 \texttt{ OR } \ldots \texttt{ OR next.throws } K_n))
\end{array}
\tag{5}
$$

We have to show

(a.1) F has no free (mathematical or state) variables

(a.2) $[\![F]\!]^{c,c'}(e)(s,s')$

From the hypotheses, we know

(6) $se(I_m) = \langle (J_1,\ldots,J_p), \langle (I_1,\ldots,I_n),(K_1,\ldots,K_m),F_C,F_R,T\rangle\rangle$

(7) $E_1 \simeq T_1,\ldots,E_p \simeq T_p$

(8) $L_1,\ldots,L_p$ is a renaming of $J_1,\ldots,J_p$

(9) $\{J_1,\ldots,J_p\} \cap \{L_1,\ldots,L_p\} = \emptyset$

(9a) $\{J_1,\ldots,J_p,L_1,\ldots,L_p\} \cap \{R\} = \emptyset$

(10) $I_r \in \{I_1,\ldots,I_n\}\backslash\{V_1,\ldots,V_o\}$

(11) $\$J_1,\ldots,\$J_p,\$L_1,\ldots,\$L_p,\$R$ do not occur in F_C,F_R

We define

(12) $c'' := call(view(c),c')$

(13) $v_1 := [\![E_1]\!]^{c'}(s),\ldots,v_p := [\![E_p]\!]^{c'}(s)$

From (4), (12), (13), and the definition of $[\![\_]\!]$, we know for some $v \in View, b \in Behavior, r \in StateRelation, t \in StateCondition, s_0 \in State$

(14) $\langle v, b \rangle = me(I_m)$

(15) $\langle r, t \rangle = b^{call(v,c')}(v_1, \ldots, v_p)$

(16) $r(s, s_0)$

(17) IF $throws(control(s_0))$
THEN $s' = s_0$
ELSE $s' = write(s_0, I_r, value(control(s_0)))^{c'}$

From (2), (12), and the definition of *pushes*, we know

(18) $DifferentVariables(c)$

(19) $DifferentVariables(c')$

(21) $c = c'$ EXCEPT $V_1, \ldots, V_o$

(21a) $\{[\![V_1]\!]^{c'}, \ldots, [\![V_o]\!]^{c'}\} \subseteq space(c)$

From (2), (12), and (COC), we know

(20) $DifferentVariables(c'')$

From (1), (14) and the definition of *specifies*, we know

(22) $v = view(c)$

(23) $specifies(se(I_m), b, c)$

From (12), the definitions of *call* and *view*, and (COV), we know

(23a) c EQUALS c''

From (2), (12), and the definitions of *pushes*, *call*, and *view*, we know

(23b) $space(c'') \subseteq space(c)$

We define

(24) $c''' := push(c'', J_1, \ldots, J_p)$

(25) $s_1 := writes(s, J_1, v_1, \ldots, J_p, v_p)^{c'''}$

From (3), (12), (15), (16), (20), (22), (23), (23a), (23b), (24), (25), and finally the definition of *specifies*, we know

(26) $\{J_1, \ldots, J_p\} \cap \{I_1, \ldots, I_n\} = \emptyset$

(27) F_C, F_R and T have no free (mathematical or state) variables

(28) F_C and T do not depend on the poststate

(29) $[\![F_C \Rightarrow F_R]\!]^{c,c'''}(e)(s_1, s_0)$

(29a) $s = s_0$ EXCEPT $range(c'') \cup \text{range}(view(c))$

(30) $s = s_0$ EXCEPT^{c,c} $I_1, \ldots, I_n$

(31) $executes(control(s_0)) \vee throws(control(s_0))$

(32) $throws(control(s_0)) \Rightarrow key(control(s_0)) \in \{K_1, \ldots, K_m\}$

From (5) and (27), we know (a.1).

From (5), to show (a.2), it suffices to show

(a.2.1)
$$
\begin{array}{l}
[\![\texttt{EXISTS } \$J_1, \ldots, \$J_p, \$L_1, \ldots, \$L_p, \$R\texttt{:} \\
\quad \$J_1\texttt{=}T_1 \texttt{ AND ... AND } \$J_p\texttt{=}T_p \texttt{ AND} \\
\quad (F_C \Rightarrow F_R) \\
\qquad [\$J_1/J_1, \ldots, \$J_p/J_p, \$L_1/J_1\texttt{'}, \ldots, \$L_p/J_p\texttt{'}][\$R/L_r\texttt{'}] \\
\qquad [?V_1/V_1, \ldots, ?V_o/V_o, ?V_1\texttt{'}/V_1\texttt{'}, \ldots, ?V_o\texttt{'}/V_o\texttt{'}] \texttt{ AND} \\
\quad \texttt{IF next.executes} \\
\qquad \texttt{THEN } I_r\texttt{'} = \texttt{next.value} \\
\qquad \texttt{ELSE } I_r\texttt{'} = \$R]\!]^{c,c'}(e)(s, s')
\end{array}
$$

(a.2.2) $s = s'$ EXCEPT$^{c,c'}(\{I_1, \ldots, I_n\} \setminus \{V_1, \ldots, V_o\}) \cup$
$\{?I : I \in \{I_1, \ldots, I_n\} \cap \{V_1, \ldots, V_o\}\}$

(a.2.3) $\neg continues(control(s'))$

(a.2.4) $\neg breaks(control(s'))$

(a.2.5) $\neg returns(control(s'))$

(a.2.6) $throws(control(s')) \Rightarrow key(control(s')) \in \{K_1, \ldots, K_m\}$

From (17), (31), (32), (CW), and Lemma "State Control Predicates", we know (a.2.3), (a.2.4), (a.2.5), and (a.2.6).

We define

(33) $w_1 := read(s_0, J_1)^{c'''}, \ldots w_p := read(s_0, J_p)^{c'''}, u := read(s_0, I_r)^{c'''}$

(34) $e_0 = e[J_1 \mapsto v_1, \ldots, J_p \mapsto v_p, L_1 \mapsto w_1, \ldots, L_p \mapsto w_p, R \mapsto u]$

To show (a.2.1), from (34) and the definition of $[\![\_]\!]$, it suffices to show

(a.2.1.1) $v_1 = [\![T_1]\!]^{c,c'}(e_0)(s, s') \wedge \ldots \wedge v_p = [\![T_p]\!]^{c,c'}(e_0)(s, s')$

$$
\begin{array}{ll}
 & [\![(F_C \Rightarrow F_R) \\
\text{(a.2.1.2)} & \quad [\$J_1/J_1, \ldots, \$J_p/J_p, \$L_1/J_1\,', \ldots, \$L_p/J_p\,'][\$R/I_r\,'] \\
 & \quad [?V_1/V_1, \ldots, ?V_o/V_o, ?V_1\,'/V_1\,', \ldots, ?V_o\,'/V_o\,']]\!]^{c,c'}(e_0)(s,s') \\
 & \text{IF } \mathit{executes}(\mathit{control}(s')) \\
\text{(a.2.1.3)} & \quad \text{THEN } \mathit{read}(s',I_r)^{c'} = \mathit{value}(\mathit{control}(s')) \\
 & \quad \text{ELSE } \mathit{read}(s',I_r)^{c'} = \mathit{read}(s_0,I_r)^{c'''}
\end{array}
$$

The proofs of (a.2.1.1) and (a.2.1.2) proceed as shown in the proof of the soundness of Rule 1.

To show (a.2.1.3), we show

(a.2.1.3.1) $\mathit{executes}(\mathit{control}(s')) \Rightarrow \mathit{read}(s',I_r)^{c'} = \mathit{value}(\mathit{control}(s'))$

(a.2.1.3.2) $\neg\mathit{executes}(\mathit{control}(s')) \Rightarrow \mathit{read}(s',I_r)^{c'} = \mathit{read}(s_0,I_r)^{c'''}$

The proof of (a.2.1.3.1) proceeds as shown in the proof of the soundness of Rule 1.

To show (a.2.1.3.2), we assume

(47) $\neg\mathit{executes}(\mathit{control}(s'))$

and show

(a.2.1.3.2.a) $\mathit{read}(s',I_r)^{c'} = \mathit{read}(s_0,I_r)^{c'''}$

From (47), (a.2.4), (a.2.5), (a.2.6) (all shown above), and Lemma "State Control Predicates", we know

(48) $\mathit{throws}(\mathit{control}(s'))$

From (17), (45), and (48), we know

(49) $s' = s_0$

From (49), it suffices to show

(a.2.1.3.2.b) $\mathit{read}(s',I_r)^{c'} = \mathit{read}(s',I_r)^{c'''}$

From the definition of *read*, it suffices to show

(a.2.1.3.2.c) $[\![I_r]\!]^{c'} = [\![I_r]\!]^{c'''}$

From (10), we know

(50) $I_r \in \{I_1, \ldots, I_n\}$

(51) $I_r \notin \{V_1, \ldots, V_o\}$

From (26) and (50), we know

(52) $I_r \notin \{J_1, \ldots, J_p\}$

From (21), (51), and the definition of EXCEPT, we know

(53) $[\![I_r]\!]^{c'} = [\![I_r]\!]^c$

From (12) and the definition of *call*, we know

(54) $[\![I_r]\!]^{c''} = [\![I_r]\!]^c$

From (20), (24), (COP), and the definition of *pushes*, we know

(55) $c''' = c''$ EXCEPT $J_1, \ldots, J_p$

From (52), (55), and the definition of EXCEPT, we know

(56) $[\![I_r]\!]^{c'''} = [\![I_r]\!]^{c''}$

From (53), (54), and (56), we know (a.2.1.3.2.c).

To show (a.2.2), from the definition of EXCEPT, it suffices to take arbitrary $x \in$ Variable with

(57) $x \notin \{[\![I]\!]^{c'} : I \in \{I_1, \ldots, I_n\} \backslash \{V_1, \ldots, V_o\}\}$

(58) $x \notin \{[\![I]\!]^{c} : I \in \{I_1, \ldots, I_n\} \cap \{V_1, \ldots, V_o\}\}$

(59) $store(s)(x) \neq store(s')(x)$

and show a contradiction.

In case $x \neq [\![I_r]\!]^{c'}$, a contradiction can be shown as in the proof of the soundness of Rule (1).

We may thus assume

(60) $x = [\![I_r]\!]^{c'}$

Then (10) and (60) contradict (57). □

6.5.4 Termination of Method Calls

$$\frac{\begin{array}{l} se(I_m) = \langle (J_1,\ldots,J_p), \langle (I_1,\ldots,I_n), (K_1,\ldots,K_m), F_C, F_R, T\rangle\rangle \\ E_1 \simeq T_1, \ldots, E_p \simeq T_p \\ L_1,\ldots,L_p \text{ is a renaming of } J_1,\ldots,J_p \\ \$L_1,\ldots,\$L_p \text{ do not occur in } F_C \\ \forall s \in State: \\ \quad [\![\texttt{now.executes AND } F \texttt{ =>} \\ \quad\quad \texttt{FORALL } \$L_1,\ldots,\$L_p: \\ \quad\quad\quad \$L_1{=}T_1 \texttt{ AND } \ldots \texttt{ AND } \$L_p{=}T_p \texttt{ =>} \\ \quad\quad\quad F_C[\$L_1/J_1,\ldots,\$L_p/J_p, \$L_1/J_1\texttt{'},\ldots,\$L_p/J_p\texttt{'}] \\ \quad\quad\quad\quad [?V_1/V_1,\ldots,?V_o/V_o, ?V_1\texttt{'}/V_1\texttt{'},\ldots,?V_o\texttt{'}/V_o\texttt{'}]]\!](s) \end{array}}{se, \{V_1,\ldots,V_o\} \vdash I_r = I_m\,(E_1,\ldots,E_p) \downarrow F}$$

The proof of the soundness of the rule is given below.

Proof Take arbitrary $c, c' \in Context, me \in MethodEnv, s \in State$ and assume

(2) F has no free (mathematical or state) variables

(3) F does not depend on the poststate

(4) $specifies(se, me, c)$

(5) $pushes(c, c', \{V_1,\ldots,V_o\})$

(6) $executes(control(s))$

(7) $[\![F]\!]^{c,c'}(s)$

We have to show

(a) $[\![I_r = I_m\,(E_1,\ldots,E_p)]\!]_{\mathrm{T}}^{c',me}(s)$

From the definition of $[\![\_]\!]_{\mathrm{T}}$, it suffices to show for some $v \in View$, $b \in Behavior$, $r \in StateRelation$, $t \in StateCondition$ with

(8) $\langle v, b\rangle = me(I_m)$

(9) $\langle r, t\rangle = b^{call(v,c')}([\![E_1]\!]^{c'}(s),\ldots,[\![E_p]\!]^{c'}(s))$

that we have

(b) $t(s)$

From the hypotheses, we know

(10) $se(I_m) = \langle (J_1, \ldots, J_p), \langle (I_1, \ldots, I_n), (K_1, \ldots, K_m), F_C, F_R, T \rangle \rangle$

(14) $E_1 \simeq T_1, \ldots, E_p \simeq T_p$

(15) $L_1, \ldots, L_p$ is a renaming of $J_1, \ldots, J_p$

(16) $\$L_1, \ldots, \$L_p$ do not occur in F_C

(17)
$\forall s \in State$:
$[\![$ `now.executes AND` F `=>`
`FORALL` $\$L_1, \ldots, \$L_p$:
$\$L_1 = T_1$ `AND ... AND` $\$L_p = T_p$ `=>`
$F_C[\$L_1/J_1, \ldots, \$L_p/J_p, \$L_1/J_1', \ldots, \$L_p/J_p']$
$[?V_1/V_1, \ldots, ?V_o/V_o, ?V_1'/V_1', \ldots, ?V_o'/V_o']$ $]\!](s)$

From (4) and the definition of *specifies*, we know

(18) $specifies(se(I_m), me(I_m), c)$

From (10) and (18), we know

(19) $specifies(\langle (J_1, \ldots, J_p), \langle (I_1, \ldots, I_n), (K_1, \ldots, K_m), F_C, F_R \rangle \rangle, me(I_m), c)$

From (8), (19) and the definition of *specifies*, we know

(11) $\{J_1, \ldots, J_p\} \cap \{I_1, \ldots, I_n\} = \emptyset$

(12) F_C, F_R and T have no free (mathematical or state) variables

(13) F_C and T do not depend on the poststate

(20) $v = view(c)$

(21) $specifies(se(I_m), b, c)$

We define

(22) $c'' := call(v, c')$

From (20), (22), the definitions of *call* and *view*, and (COV), we know

(22a) c EQUALS c''

From (5) and the definition of *pushes*, we know

(23) $DifferentVariables(c)$

(24) $DifferentVariables(c')$

(25) $c = c'$ EXCEPT $V_1, \ldots, V_o$

From (5), (20), (22) and (COC), we know

(26) $\mathit{DifferentVariables}(c'')$

We define

(27) $v_1 := [\![E_1]\!]^{c'}(s), \ldots, v_p := [\![E_p]\!]^{c'}(s)$

From (6), (9), (11), (21), (22a), (25), (26), and the definition of *specifies*, we know for for some $c''' \in \mathit{Context}$ and $s_1 \in \mathit{State}$

(28) $c''' = \mathit{push}(c'', J_1, \ldots, J_p)$

(29) $s_1 = \mathit{writes}(s, J_1, v_1, \ldots, J_p, v_p)^{c'''}$

(30) $\forall e \in \mathit{Environment} : [\![F_C]\!]^{c,c'''}(e)(s_1, s_1) \Rightarrow t(s)$

From (30) and the definition of $[\![\_]\!]$, to show (b), it suffices to show for arbitrary $e \in \mathit{Environment}$

(c) $[\![F_C]\!]^{c,c'''}(e)(s_1, s_1)$

From (6), (7), (17), and the definition of $[\![\_]\!]$, we know

(31)
$$[\![\texttt{FORALL } \$L_1, \ldots, \$L_p : \\ \quad \$L_1 \texttt{=} T_1 \texttt{ AND ... AND } \$L_p \texttt{=} T_p \texttt{ =>} \\ \quad F_C[\$L_1/J_1, \ldots, \$L_p/J_p, \$L_1/J_1\texttt{'}, \ldots, \$L_p/J_p\texttt{'}] \\ \qquad [?V_1/V_1, \ldots, ?V_o/V_o, ?V_1\texttt{'}/V_1\texttt{'}, \ldots, ?V_o\texttt{'}/V_o\texttt{'}]]\!]^{c,c'}(e)(s,s)$$

From (14) and (27), we know

(32) $v_1 = [\![T_1]\!]^{c,c'}(e)(s,s) \wedge \ldots \wedge v_p = [\![T_p]\!]^{c,c'}(e)(s,s)$

(33) $T_1, \ldots, T_p$ have no free (mathematical or state) variables

We define

(34) $e_0 := e[L_1 \mapsto v_1, \ldots, L_p \mapsto V_p]$

From (31), (32), (33), (34), the definition of $[\![\_]\!]$, and (MVT'), we know

(35)
$$[\![F_C[\$L_1/J_1, \ldots, \$L_p/J_p, \$L_1/J_1\texttt{'}, \ldots, \$L_p/J_p\texttt{'}] \\ \qquad [?V_1/V_1, \ldots, ?V_o/V_o, ?V_1\texttt{'}/V_1\texttt{'}, \ldots, ?V_o\texttt{'}/V_o\texttt{'}]]\!]^{c,c'}(e_0)(s,s)$$

From (20), (22), and the definitions of *context* and *view*, we know

(36) $view(c) = view(c'')$

From (36) and (COV), we know

(37) c EQUALS c''

From (37) and the definitions of EQUALS, and AT, we know

(38) $c = c''$ AT $V_1, \ldots, V_o$

From (25), (37), and the definitions of EQUALS, and EXCEPT, we know

(39) $c' = c''$ EXCEPT $V_1, \ldots, V_o$

From (35), (38), (39), and (PMGF), we know

(40) $[\![F_C[\$L_1/J_1, \ldots, \$L_p/J_p, \$L_1/J_1', \ldots, \$L_p/J_p']]\!]^{c,c''}(e_0)(s,s)$

From (26), (28) and (COP), we know

(41) $pushes(c'', c''', \{J_1, \ldots, J_p\})$

From (41) and the definition of *pushes*, we know

(42) $c'' = c'''$ EXCEPT $J_1, \ldots, J_p$

From (40), (42), (MPVF0'), (MPVF1'), and (COF1), we know

(43) $[\![F_C[\$L_1/J_1, \ldots, \$L_p/J_p, \$L_1/J_1', \ldots, \$L_p/J_p']]\!]^{c,c'''}(e_0)(s,s)$

From (29) and (RWE), we know

(44) $read(s_1, J_1)^{c'''} = v_1 \wedge \ldots \wedge read(s_1, J_p)^{c'''} = v_p$

From (29) and (WSE), we know

(45) $s_1 = s$ EXCEPT$^{c'''}$ $J_1, \ldots, J_p$

From (29) and (CWE), we know

(46) $control(s_1) = control(s)$

From (15), (16), (34), (43), (44), (45), (46), and finally (PMVF1") and (PMVF2"), we know (c). □

6.5.5 Further Judgements

Figure 6.28 generalizes the judgements for preconditions, postconditions and assertions from the command language to the method language with contexts. The corresponding generic rules for computing pre- and postconditions are shown in Figure 6.29. They are straight-forward generalizations of the basic rules; we omit the proofs of their soundness. Likewise, the existing rules of the assertion calculus can be generalized in a straight-forward way to the method language; Figure 6.29 gives a new rule for method calls.

6.6 Reasoning about Programs

In this section, we lift the reasoning calculus from the level of individual methods to the level of whole programs. For this purpose, we first define in Figure 6.30 an auxiliary function $[\![Ms]\!]_{\mathrm{S}}$ which computes the environment of method specifications established by the method declarations Ms.

Next, we introduce in Figure 6.31 a judgement

$$[F]^{F_c,F_b,F_r,\{L_1,\ldots,L_r\}}_{M_1,\ldots,M_o} \;\S\; {}^{K_1,\ldots,K_m}_{I_1,\ldots,I_n}$$

which establishes that a command specification $[F]^{F_c,F_b,F_r,\{L_1,\ldots,L_r\}}_{M_1,\ldots,M_o}$ entails that the command does not yield a continuing or a breaking state, that it does not modify variables apart from those denoted by $I_1,\ldots,I_n$, and that it does not raise exceptions apart from those with keys $K_1,\ldots,K_m$. A command satisfying this specification thus represents a suitable body for a method specified as

$$\texttt{writesonly}\ I_1,\ldots,I_n\ \texttt{throwsonly}\ K_1,\ldots,K_m$$

The condition on the result state of the command ensures that the statements `continue` and `break` cannot escape the body of a `while` loop. The condition denoted by the method specification is in particular satisfied if $\{M_1,\ldots,M_o\} \subseteq \{I_1,\ldots,I_n\}$ and $\{L_1,\ldots,L_r\} \subseteq \{K_1,\ldots,K_m\}$. The judgement however gives the additional freedom of satisfying the condition, rather than by syntactic constraints on the method body, by proof from the specification formula F.

The derivation of this judgement is shown in Figure 6.31 with the aid of four auxiliary judgments derived in Figure 6.32:

- $F \;\S_{\mathrm{c}}\; F_c$: establishes that the specification

Method Language: Further Judgements

$$
\begin{array}{l}
se, \{V_1, \ldots, V_o\} \vdash \text{PRE}(C, Q) = P \Leftrightarrow \\
\quad Q \text{ has no primed program variables and no occurr. of } \texttt{next} \Rightarrow \\
\quad\quad P \text{ has no primed prog. variables and no occurr. of } \texttt{next} \wedge \\
\quad\quad \forall c, c' \in \mathit{Context}, me \in \mathit{MethodEnv} : \\
\quad\quad\quad \mathit{specifies}(se, me, c) \wedge \mathit{pushes}(c, c', \{V_1, \ldots, V_o\}) \Rightarrow \\
\quad\quad\quad\quad \forall e \in \mathit{Environment}, s, s' \in \mathit{State} : \\
\quad\quad\quad\quad\quad \mathit{executes}(\mathit{control}(s)) \Rightarrow \\
\quad\quad\quad\quad\quad\quad (\llbracket P \rrbracket^{c,c'}(e)(s,s) \wedge \llbracket C \rrbracket^{c',me}(s,s') \Rightarrow \\
\quad\quad\quad\quad\quad\quad\quad \llbracket Q \rrbracket^{c,c'}(e)(s',s'))
\end{array}
$$

$$
\begin{array}{l}
se, \{V_1, \ldots, V_o\} \vdash \text{POST}(C, P) = Q \Leftrightarrow \\
\quad P \text{ has no primed program variables and no occurr. of } \texttt{next} \Rightarrow \\
\quad\quad Q \text{ has no primed prog. variables and no occurr. of } \texttt{next} \wedge \\
\quad\quad \forall c, c' \in \mathit{Context}, me \in \mathit{MethodEnv} : \\
\quad\quad\quad \mathit{specifies}(se, me, c) \wedge \mathit{pushes}(c, c', \{V_1, \ldots, V_o\}) \Rightarrow \\
\quad\quad\quad\quad \forall e \in \mathit{Environment}, s, s' \in \mathit{State} : \\
\quad\quad\quad\quad\quad \mathit{executes}(\mathit{control}(s)) \Rightarrow \\
\quad\quad\quad\quad\quad\quad (\llbracket P \rrbracket^{c,c'}(e)(s,s) \wedge \llbracket C \rrbracket^{c',me}(s,s') \Rightarrow \\
\quad\quad\quad\quad\quad\quad\quad \llbracket Q \rrbracket^{c,c'}(e)(s',s'))
\end{array}
$$

$$
\begin{array}{l}
se, \{V_1, \ldots, V_o\} \vdash \text{TRANS}(C, P) = C' \Leftrightarrow \\
\quad P \text{ has no primed program variables and no occurr. of } \texttt{next} \Rightarrow \\
\quad\quad \forall c, c' \in \mathit{Context}, me \in \mathit{MethodEnv} : \\
\quad\quad\quad \mathit{specifies}(se, me, c) \wedge \mathit{pushes}(c, c', \{V_1, \ldots, V_o\}) \Rightarrow \\
\quad\quad\quad\quad \forall s, s' \in \mathit{State} : \mathit{executes}(\mathit{control}(s)) \Rightarrow \\
\quad\quad\quad\quad\quad (\llbracket P \rrbracket^{c,c'}(s) \wedge \llbracket C \rrbracket^{c',me}(s,s') \Leftrightarrow \llbracket C' \rrbracket^{c',me}(s,s'))
\end{array}
$$

Figure 6.28: Further Judgements for the Method Language

Method Language: Further Judgements

$$\frac{se, \{V_1, \ldots, V_o\} \vdash C : [F]^{F_c, F_b, F_r, \{K_1, \ldots, K_m\}}_{I_1, \ldots, I_n} \quad J_1, \ldots, J_n \text{ is a renaming of } I_1, \ldots, I_n \quad \$J_1, \ldots, \$J_n \text{ do not occur in } Q \quad \#I_s \text{ do not occur in } Q}{se, \{V_1, \ldots, V_o\} \vdash \text{PRE}(C, Q) = \texttt{FORALL } \$J_1, \ldots, \$J_n \texttt{ : ALLSTATE } \#I_s \texttt{ :} \; F[\#I_s/\texttt{next}][\$J_1/I_1\texttt{'}, \ldots, \$J_n/I_n\texttt{'}] \texttt{ =>} \; Q[\#I_s/\texttt{now}][\$J_1/I_1, \ldots, \$J_n/I_n]}$$

$$\frac{se, \{V_1, \ldots, V_o\} \vdash C : [F]^{F_c, F_b, F_r, \{K_1, \ldots, K_m\}}_{I_1, \ldots, I_n} \quad J_1, \ldots, J_n \text{ is a renaming of } I_1, \ldots, I_n \quad \$J_1, \ldots, \$J_n \text{ do not occur in } P \quad \#I_s \text{ do not occur in } P}{se, \{V_1, \ldots, V_o\} \vdash \text{POST}(C, P) = \texttt{EXISTS } \$J_1, \ldots, \$J_n \texttt{ : EXSTATE } \#I_s \texttt{ :} \; P[\#I_s/\texttt{now}][\$J_1/I_1, \ldots, \$J_n/I_n] \texttt{ AND} \; F[\#I_s/\texttt{now}, \texttt{now}/\texttt{next}] \; [\$J_1/I_1, \ldots, \$J_n/I_n, I_1/I_1\texttt{'}, \ldots, I_n/I_n\texttt{'}]}$$

$$se, \{V_1, \ldots, V_o\} \vdash \text{TRANS}(I_r = I_m\,(E_1, \ldots, E_p)\,, P) = \texttt{assert } P^+\texttt{; } I_r = I_m\,(E_1, \ldots, E_p)$$

Figure 6.29: Further Judgements for the Method Language

Method Specifications: Definitions

$[\![\_]\!]_S$: **Methods** $\rightarrow$ ***SpecEnv*** $\rightarrow$ ***SpecEnv***
$[\![\_]\!]_S(se) = se$
$[\![Ms\ M]\!]_S(se) = [\![M]\!]_S([\![Ms]\!]_S(se))$

$[\![\_]\!]_S$: **Method** $\rightarrow$ ***SpecEnv*** $\rightarrow$ ***SpecEnv***
$[\![\texttt{method}\ I_m\ (J_1,\ldots,J_p)\ \ S\ \{C\}\ ]\!]_S(se) =$
$\quad se[I_m \mapsto [\![\texttt{method}\ I_m\ (J_1,\ldots,J_p)\ \ S\ \{C\}\ ]\!]_S]$

$[\![\_]\!]_S$: **Method** $\rightarrow$ ***MethodSpec***
$[\![\texttt{method}\ I_m\ (J_1,\ldots,J_p)\ \ S\ \{C\}\ ]\!]_S = \langle (J_1,\ldots,J_p), [\![S]\!] \rangle$

$[\![\_]\!]$: **Specification** $\rightarrow$ ***Spec***
$[\![\texttt{writesonly}\ I_1,\ldots,I_n, \texttt{?}J_1,\ldots,\texttt{?}J_o\ \texttt{throwsonly}\ K_1,\ldots,K_m$
$\quad \texttt{assumes}\ F_C\ \texttt{implements}\ F_R\ \texttt{decreases}\ T\ ]\!] =$
$\quad \langle (I_1,\ldots,I_n,\texttt{?}J_1,\ldots,\texttt{?}J_o),(K_1,\ldots,K_m),F_C,F_R,T \rangle$

Figure 6.30: Definitions for Method Specifications (2/2)

$\texttt{next.continues} \Rightarrow F_c$

entails that the command does not result in a continuing state.

- $F\ \S_b\ F_b$: establishes that the specification

  $\texttt{next.breaks} \Rightarrow F_b$

  entails that the command does not result in a breaking state.

- $F\ \S_s^{M_1,\ldots,M_o}_{I_1,\ldots,I_n}$: establishes that the specification

  $F\ \texttt{AND}\ \texttt{writesonly}\ M_1,\ldots,M_o$

  entails that the command does not change variables other than those denoted by $I_1,\ldots,I_n$.

- $F\ \S_e^{L_1,\ldots,L_r}_{K_1,\ldots,K_m}$: establishes that the specification

  $F\ \texttt{AND}$
  $\quad (\texttt{next.throws} \Rightarrow$
  $\quad\quad (\texttt{next.throws}\ L_1\ \texttt{OR}\ \ldots\ \texttt{OR}\ \texttt{next.throws}\ L_r))$

Method Specifications: Judgements

$$
\begin{array}{l}
[F]^{F_c,F_b,F_r,\{L_1,\ldots,L_r\}}_{M_1,\ldots,M_o} \;\S\; {}^{K_1,\ldots,K_m}_{I_1,\ldots,I_n} \;\Leftrightarrow \\
\quad \forall c \in \mathit{Context}, s, s' \in \mathit{State}, e \in \mathit{Environment} : \\
\qquad \mathit{executes}(\mathit{control}(s)) \wedge \\
\qquad [\![[F]^{F_c,F_b,F_r,\{L_1,\ldots,L_r\}}_{M_1,\ldots,M_o}]\!]^{c,c}(e)(s,s') \Rightarrow \\
\qquad\quad \neg \mathit{continues}(\mathit{control}(s')) \wedge \\
\qquad\quad \neg \mathit{breaks}(\mathit{control}(s')) \wedge \\
\qquad\quad s = s' \text{ EXCEPT}^{c,c}\ I_1, \ldots, I_n \wedge \\
\qquad\quad (\mathit{throws}(\mathit{control}(s')) \Rightarrow \\
\qquad\qquad \mathit{key}(\mathit{control}(s')) \in \{K_1, \ldots, K_m\})
\end{array}
$$

Method Specifications: Rules

$$
\frac{\begin{array}{l} F \;\S_{\mathrm{c}}\; F_c \\ F \;\S_{\mathrm{b}}\; F_b \\ F \;\S_{\mathrm{s}}\; {}^{M_1,\ldots,M_o}_{I_1,\ldots,I_n} \\ F \;\S_{\mathrm{e}}\; {}^{L_1,\ldots,L_r}_{K_1,\ldots,K_m} \end{array}}{[F]^{F_c,F_b,F_r,\{L_1,\ldots,L_r\}}_{M_1,\ldots,M_o} \;\S\; {}^{K_1,\ldots,K_m}_{I_1,\ldots,I_n}}
$$

Figure 6.31: Judgements and Rules for Method Specifications

Method Specifications: Rules

F §$_{\mathrm{c}}$ `FALSE`

$$\frac{\forall c \in \mathit{Context}, e \in \mathit{Environment}, s, s' \in \mathit{Store}: \quad [\![\texttt{now.executes AND } F \texttt{ =>} \quad \texttt{!next.continues}]\!]^{c,c}(e)(s,s')}{F \ \S_{\mathrm{c}}\ F_c}$$

F §$_{\mathrm{b}}$ `FALSE`

$$\frac{\forall c \in \mathit{Context}, e \in \mathit{Environment}, s, s' \in \mathit{Store}: \quad [\![\texttt{now.executes AND } F \texttt{ =>} \quad \texttt{!next.breaks}]\!]^{c,c}(e)(s,s')}{F \ \S_{\mathrm{b}}\ F_b}$$

$$\frac{\forall c \in \mathit{Context}, e \in \mathit{Environment}, s, s' \in \mathit{Store}: \quad [\![\texttt{now.executes AND } F \texttt{ =>} \quad M_1\texttt{=}M_o\texttt{'} \texttt{ AND ... AND } M_1\texttt{=}M_o\texttt{'}]\!]^{c,c}(e)(s,s')}{F \ \S_{\mathrm{s}}^{I_1,\ldots,I_m,M_1,\ldots,M_o}{}_{I_1,\ldots,I_m,I_{m+1},\ldots,I_n}}$$

$$\frac{\forall c \in \mathit{Context}, e \in \mathit{Environment}, s, s' \in \mathit{Store}: \quad [\![\texttt{now.executes AND } F \texttt{ =>} \quad \texttt{!next.throws } L_1 \texttt{ AND ... AND} \quad \texttt{!next.throws } L_r]\!]^{c,c}(e)(s,s')}{F \ \S_{\mathrm{s}}^{K_1,\ldots,K_n,L_1,\ldots,L_r}{}_{K_1,\ldots,K_n,K_{n+1},\ldots,K_m}}$$

Figure 6.32: Rules for Method Specifications

entails that the command does not change variables other than those denoted by $K_1, \ldots, K_m$.

The soundness of the rule is fairly obvious; we omit the proof.

In Figure 6.33, we introduce the three core judgements of the method language:

- $se \vdash M : se'$: this judgement states that in specification environment se the method M is correct with respect to its specification and that the method declaration yields a new specification environment se'.
- $se \vdash Ms : se'$: this judgement states that in specification environment se the methods Ms are correct with respect to their specifications and that the method declarations yield a new specification environment se'.
- $se \vdash Ms\ S\ \{C\}$: this is the "top-level" judgement of our calculus. Given a specification environment se and program $Ms\ S\ \{C\}$, it states that the methods Ms are correct with respect to their specifications and that the program body C is correct with respect to its specification S.

As shown in the definitions of the judgements, the notion of the "correctness of a method" is captured by the previously defined predicate *specifies*, while the "correctness of the program body" is captured by the predicate *correctness* defined in Figure 6.33. In both cases, it is essentially claimed that the transition relation induced by the method/program body is captured by the predicate $F_C =>F_R$, that the termination condition is captured by F_C and that the poststate is constrained by the frame condition and list of exceptions stated in the specification.

Actually, also our notion of method/program correctness should also comprise a "well-definedness" condition that states that the execution of the method/program body C does not encounter undefined expression values, i.e. that the transition relation $[\![C]\!]$ coincides with the transition relation $[\![C]\!]_\perp$ (and likewise for the termination condition). This, however, would require to extend our definition of method environments in order to record for every method two state relations and two termination conditions (rather than one). To keep our definitions simple, we omit the claim (but nevertheless keep the well-definedness judgement in the rules for method declarations and programs, even if it is not required for proving the soundness of the rules).

The rules for the three judgements are given in Figures 6.34 and 6.35. The soundness of the two rules for the judgement on the correctness of method declaration sequences Ms is obvious from the the rules and definition of $[\![Ms]\!]_S$; we thus focus in the following on the soundness of the rules for the other two judgements on the correctness of method declarations and on the correctness of programs.

The soundness proofs depend on the following lemmas.

Method Language: Definitions

$$
\begin{array}{l}
correctness \subseteq Spec \times StateRelation \times StateCondition \times Context \\
correctness(\langle (I_1,\ldots,I_n),(K_1,\ldots,K_m),F_C,F_R,T\rangle,r,t,c) \Leftrightarrow \\
\quad \forall s \in State, e \in Environment : executes(control(s)) \Rightarrow \\
\qquad (\forall s' \in State : r(s,s') \Rightarrow \\
\qquad\quad [\![F_C \mathrel{=>} F_R]\!]^{c,c}(e)(s,s') \wedge \\
\qquad\quad s = s' \text{ EXCEPT}^{c,c}\ I_1,\ldots,I_n \wedge \\
\qquad\quad \neg continues(control(s')) \wedge \neg breaks(control(s')) \wedge \\
\qquad\quad (throws(control(s')) \Rightarrow \\
\qquad\qquad key(control(s')) \in \{K_1,\ldots,K_m\})) \\
\quad \wedge ([\![F_C]\!]^{c,c}(e)(s,s) \Rightarrow t(s))
\end{array}
$$

Method Language: Judgements

$$
\begin{array}{l}
se \vdash M : se' \Leftrightarrow \\
\quad se' = [\![M]\!]_{\mathrm{S}}(se) \wedge \\
\quad \forall c \in Context, me \in MethodEnv : \\
\qquad DifferentVariables(c) \wedge specifies(se,me,c) \Rightarrow \\
\qquad\quad specifies(se', [\![M]\!]^{view(c)}(me), c)
\end{array}
$$

$$
\begin{array}{l}
se \vdash Ms : se' \Leftrightarrow \\
\quad se' = [\![Ms]\!]_{\mathrm{S}}(se) \wedge \\
\quad \forall c \in Context, me \in MethodEnv : \\
\qquad DifferentVariables(c) \wedge specifies(se,me,c) \Rightarrow \\
\qquad\quad specifies(se', [\![Ms]\!]^{view(c)}(me), c)
\end{array}
$$

$$
\begin{array}{l}
se \vdash Ms\ S\ \{C\} \Leftrightarrow \\
\quad \forall c \in Context, me \in MethodEnv : \\
\qquad DifferentVariables(c) \wedge specifies(se,me,c) \Rightarrow \\
\qquad\quad \text{LET } me' = [\![Ms]\!]^{view(c)}(me) \text{ IN} \\
\qquad\qquad specifies([\![Ms]\!]_{\mathrm{S}}(se), me', c) \wedge \\
\qquad\qquad correctness([\![S]\!], [\![C]\!]^{c,me'}, [\![C]\!]_{\mathrm{T}}^{c,me'}, c)
\end{array}
$$

Figure 6.33: Judgements for the Method Language

Lemma (Commands with Methods) A command can only change variables that are in the range of its context or in the views of the methods that can be called by the command:

$$\begin{array}{l}\forall C \in \text{Command}, me \in \mathit{MethodEnv}, c' \in \mathit{Context}, s, s' \in \mathit{State} : \\ \quad \text{LET } V = \bigcup_{I \in \text{Identifier}} \text{range}(\mathit{view}(me(I))) \text{ IN} \\ \quad \mathit{DifferentVariables}(c') \wedge [\![C]\!]^{c', me}(s, s') \Rightarrow \\ \quad\quad s = s' \text{ EXCEPT } \mathit{range}(c') \cup V\end{array}$$

Proof From the fact that the only variables referenced in the command are the results of the application of some view and that all views in the command are derived from either c' or from the view of some method by applications of *push* which, by Lemma "Range of Contexts" does not widen the range. □

Lemma (Changing Local Variables) When leaving the scope of a declaration, the view on the locally declared identifiers is unchanged:

$$\begin{array}{l}\forall c, c', c'' \in \mathit{Context}, I_1, \ldots, I_n, J_1, \ldots, J_m, V_1, \ldots, V_o \in \text{Identifier} : \\ \forall s_0, s_1 \in \mathit{State} : \\ \quad \mathit{DifferentVariables}(c) \wedge \mathit{DifferentVariables}(c') \wedge \\ \quad \mathit{range}(c') \subseteq \mathit{range}(c) \wedge \mathit{space}(c') \subseteq \mathit{space}(c) \wedge \\ \quad \mathit{pushes}(c', c'', \{V_1, \ldots, V_o\}) \wedge \\ \quad s_0 = s_1 \text{ EXCEPT}^{c, c''} I_1, \ldots, I_n, ?J_1, \ldots, ?J_m \Rightarrow \\ \quad\quad s_0 = s_1 \text{ EXCEPT}^{c, c'} \\ \quad\quad\quad \{I_1, \ldots, I_n\} \setminus \{V_1, \ldots, V_o\} \cup \{?J_1, \ldots, ?J_m\}\end{array}$$

Proof Take arbitrary $c, c' \in \mathit{Context}, I_1, \ldots, I_n, J_1, \ldots, J_m, V_1, \ldots, V_o \in \text{Identifier}$ and $s_0, s_1 \in \mathit{State}$. We assume

(1) $\mathit{DifferentVariables}(c)$

(2) $\mathit{DifferentVariables}(c')$

(3) $\mathit{range}(c') \subseteq \mathit{range}(c)$

(4) $\mathit{space}(c') \subseteq \mathit{space}(c)$

(5) $\mathit{pushes}(c', c'', \{V_1, \ldots, V_o\})$

(6) $s_0 = s_1 \text{ EXCEPT}^{c, c''} I_1, \ldots, I_n, ?J_1, \ldots, ?J_m$

and show

(a) $s_0 = s_1 \text{ EXCEPT}^{c,c'} \{I_1, \ldots, I_n\} \setminus \{V_1, \ldots, V_o\} \cup \{?J_1, \ldots, ?J_m\}$

From the definition of EXCEPT, it suffices to assume for arbitrary $x \in \mathit{Variable}$

(7) $x \notin \{[\![I_1]\!]^{c'}, \ldots, [\![I_n]\!]^{c'}\} \setminus \{[\![V_1]\!]^{c'}, \ldots, V_o^{c'}\} \cup \{[\![J_1]\!]^{c}, \ldots, [\![J_m]\!]^{c}\}$

(8) $x \notin \mathit{space}(c')$

(9) $\mathit{store}(s_0)(x) \neq \mathit{store}(s_1)(x)$

and show a contradiction.

From (5) and the definition of *pushes*, we know

(10) $\mathit{DifferentVariables}(c'')$

(11) $c' = c'' \text{ EXCEPT } V_1, \ldots, V_o$

(12) $\{[\![V_1]\!]^{c''}, \ldots, [\![V_o]\!]^{c''}\} \cup \mathit{space}(c'') \subseteq \mathit{space}(c')$

From (8) and (12), we know

(13) $x \notin \mathit{space}(c'')$

From (7), we know

(14) $x \notin \{[\![I_1]\!]^{c'}, \ldots, [\![I_n]\!]^{c'}\} \vee x \in \{[\![V_1]\!]^{c'}, \ldots, [\![V_o]\!]^{c'}\}$

(15) $x \notin \{[\![J_1]\!]^{c}, \ldots, [\![J_m]\!]^{c}\}$

From (6), (9), (13), (15), and the definition of EXCEPT, we know

(16) $x \in \{[\![I_1]\!]^{c''}, \ldots, [\![I_n]\!]^{c''}\}$

From (14), we have two cases.

In the first case, we assume

(17) $x \in \{[\![V_1]\!]^{c'}, \ldots, [\![V_o]\!]^{c'}\}$

From (8) and (12), we know

(18) $x \notin \{[\![V_1]\!]^{c''}, \ldots, [\![V_o]\!]^{c''}\}$

From (16) and (18), we have some $I \in$ Identifier such that

(19) $x = [\![I]\!]^{c''}$

(20) $I \notin \{V_1, \ldots, V_o\}$

From (11), (19), and (20), we know

(21) $x = [\![I]\!]^{c'}$

But (2), (17), (20), and (21), contradict the definition of *DifferentVariables*.

In the second case, we assume

(22) $x \notin \{[\![V_1]\!]^{c'}, \ldots, [\![V_o]\!]^{c'}\}$

(23) $x \notin \{[\![I_1]\!]^{c'}, \ldots, [\![I_n]\!]^{c'}\}$

From (16), we have some $I \in$ Identifier such that

(24) $x = [\![I]\!]^{c''}$

(25) $I \in \{I_1, \ldots, I_n\}$

From (8), (12), and (24), we know

(26) $I \notin \{V_1, \ldots, V_o\}$

From (11), (24), (26), and the definition of EXCEPT, we know

(27) $x = [\![I]\!]^{c'}$

But (23), (25), and (27) form a contradiction. □

Method Language: Rules

$$\frac{\begin{array}{l}
S = \texttt{writesonly}\ I_1,\ldots,I_n\ \texttt{throwsonly}\ K_1,\ldots,K_m \\
\quad \texttt{assumes}\ F_C\ \texttt{implements}\ F_R\ \texttt{decreases}\ T \\
\{J_1,\ldots,J_p\} \cap \{I_1,\ldots,I_n\} = \emptyset \\
F_C, F_R \text{ and } T \text{ have no free (mathematical or state) variables} \\
F_C, F_R \text{ and } T \text{ do not depend on the poststate} \\
se, \{J_1,\ldots,J_p\} \vdash C : [F]^{F_c,F_b,F_r,\{L_1,\ldots,L_r\}}_{M_1,\ldots,M_o} \\
L_1,\ldots,L_p \text{ is a renaming of } J_1,\ldots,J_p \\
\$J_1,\ldots,\$J_p,\$L_1,\ldots,\$L_p, \text{ do not occur in } F \\
[\texttt{EXISTS}\ \$J_1,\ldots,\$J_p,\$L_1,\ldots,\$L_p : \\
\quad F[\$J_1/J_1,\ldots,\$J_p/J_p,\$L_1/J_1\texttt{'},\ldots,\$L_p/J_p\texttt{'}] \\
\quad]^{F_c,F_b,F_r,\{L_1,\ldots,L_r\}}_{\{M_1,\ldots,M_o\}\setminus\{J_1,\ldots,J_p\}} \S\, ^{K_1,\ldots,K_m}_{I_1,\ldots,I_n} \\
se, \{J_1,\ldots,J_p\} \vdash C \checkmark F_C \\
\#I_s \text{ does not occur in } F \\
\forall s, s' \in \mathit{State}, e \in \mathit{Environment}, c_0, c_1 \in \mathit{Context} : \\
\quad [\![(\texttt{now.executes AND} \\
\qquad \texttt{EXSTATE}\ \#I_s : \\
\qquad\quad F[\#I_s/\texttt{next}]\ \texttt{AND} \\
\qquad\quad \texttt{IF}\ \#I_s\texttt{.throws THEN} \\
\qquad\qquad \texttt{next} == \#I_s \\
\qquad\quad \texttt{ELSE} \\
\qquad\qquad \texttt{next.executes AND} \\
\qquad\qquad \texttt{next.value} = \#I_s\texttt{.value}) \Rightarrow \\
\qquad (F_C \Rightarrow F_R)]\!]^{c_0,c_1}(e)(s,s') \\
se, \{J_1,\ldots,J_p\} \vdash C \downarrow F_C
\end{array}}{\begin{array}{l}
se \vdash \texttt{method}\ I_m\ (J_1,\ldots,J_p)\ S\ \{C\} : \\
\quad se[I_m \mapsto \langle (J_1,\ldots,J_p), \langle (I_1,\ldots,I_n), (K_1,\ldots,K_m), F_C, F_R, T\rangle\rangle]
\end{array}}$$

$$se \vdash \_ : se$$

$$\frac{se \vdash Ms : se' \qquad se' \vdash M : se''}{se \vdash Ms\,;M : se''}$$

Figure 6.34: Rules for the Method Language

Method Language: Rules

$$\frac{\begin{array}{l} se \vdash Ms : se' \\ S = \texttt{writesonly}\ I_1,\ldots,I_n\ \texttt{throwsonly}\ K_1,\ldots,K_m \\ \quad \texttt{assumes}\ F_C\ \texttt{implements}\ F_R\ \texttt{decreases}\ T \\ F_C, F_R \text{ and } T \text{ have no free (mathematical or state) variables} \\ F_C \text{ and} T \text{ do not depend on the poststate} \\ se', \emptyset \vdash C : [\,F\,]^{F_c,F_b,F_r,\{L_1,\ldots,L_r\}}_{M_1,\ldots,M_o} \\ [\,F\,]^{F_c,F_b,F_r,\{L_1,\ldots,L_r\}}_{M_1,\ldots,M_o} \ \S\ ^{K_1,\ldots,K_m}_{I_1,\ldots,I_n} \\ se', \emptyset \vdash C \checkmark F_C \\ \forall s, s' \in State, e \in Environment, c_0, c_1 \in Context : \\ \quad [\![(\texttt{now.executes AND}\ F) \texttt{ => } (F_C \texttt{ => } F_R)]\!]^{c_0,c_1}(e)(s,s') \\ se', \emptyset \vdash C \downarrow F_C \end{array}}{se \vdash Ms\ S\ \{C\}}$$

Figure 6.35: Rules for the Method Language

6.6.1 Verification of Method Declarations

$$
\frac{\begin{array}{l}
S = \texttt{writesonly}\ I_1,\ldots,I_n\ \texttt{throwsonly}\ K_1,\ldots,K_m \\
\quad \texttt{assumes}\ F_C\ \texttt{implements}\ F_R\ \texttt{decreases}\ T \\
\{J_1,\ldots,J_p\} \cap \{I_1,\ldots,I_n\} = \emptyset \\
F_C, F_R \text{ and } T \text{ have no free (mathematical or state) variables} \\
F_C \text{ and } T \text{ do not depend on the poststate} \\
se, \{J_1,\ldots,J_p\} \vdash C : [F]^{F_c,F_b,F_r,\{L_1,\ldots,L_r\}}_{M_1,\ldots,M_o} \\
L_1,\ldots,L_p \text{ is a renaming of } J_1,\ldots,J_p \\
\$J_1,\ldots,\$J_p,\$L_1,\ldots,\$L_p, \text{ do not occur in } F \\
[\texttt{EXISTS}\ \$J_1,\ldots,\$J_p,\$L_1,\ldots,\$L_p: \\
\quad F[\$J_1/J_1,\ldots,\$J_p/J_p,\$L_1/J_1\texttt{'},\ldots,\$L_p/J_p\texttt{'}] \\
\quad]^{F_c,F_b,F_r,\{L_1,\ldots,L_r\}}_{\{M_1,\ldots,M_o\}\setminus\{J_1,\ldots,J_p\}}\ \S\ ^{K_1,\ldots,K_m}_{I_1,\ldots,I_n} \\
se, \{J_1,\ldots,J_p\} \vdash C \checkmark F_C \\
\#I_s \text{ does not occur in } F \\
\forall s, s' \in \mathit{State}, e \in \mathit{Environment}, c_0, c_1 \in \mathit{Context}: \\
\quad [\![(\texttt{now.executes AND} \\
\qquad \texttt{EXSTATE}\ \#I_s: \\
\qquad\quad F[\#I_s/\texttt{next}]\ \texttt{AND} \\
\qquad\quad \texttt{IF}\ \#I_s\texttt{.throws THEN} \\
\qquad\qquad \texttt{next} == \#I_s \\
\qquad\quad \texttt{ELSE} \\
\qquad\qquad \texttt{next.executes AND} \\
\qquad\qquad \texttt{next.value} = \#I_s\texttt{.value}) \Rightarrow \\
\qquad (F_C \Rightarrow F_R)]\!]^{c_0,c_1}(e)(s,s') \\
se, \{J_1,\ldots,J_p\} \vdash C \downarrow F_C
\end{array}}{\begin{array}{l}
se \vdash \texttt{method}\ I_m\ (J_1,\ldots,J_p)\ S\ \{C\}: \\
\quad se[I_m \mapsto \langle (J_1,\ldots,J_p), \langle (I_1,\ldots,I_n), (K_1,\ldots,K_m), F_C, F_R\rangle\rangle]
\end{array}}
$$

This rule claims correctness of a method declaration with respect to its specification. It essentially generalizes the rule for the correctness of the program body (see the next subsection for a more detailed explanation) in that the context of reasoning is set up by the method parameters $J_1,\ldots,J_p$.

Furthermore, hypothesis 11 has become substantially more complicated: the assumption for establishing $F_C \Rightarrow F_R$ is not any more F but

$$
\begin{array}{l}
\texttt{EXSTATE}\ \#I_s\texttt{:}\ F[\#I_s/\texttt{next}]\ \texttt{AND} \\
\quad \texttt{IF}\ \#I_s\texttt{.throws THEN} \\
\qquad \texttt{next} == \#I_s \\
\quad \texttt{ELSE} \\
\qquad \texttt{next.executes AND next.value} = \#I_s\texttt{.value}
\end{array}
$$

This version of the assumption arises from our wish to formulate the method specification F_C =>F_R with respect to the control data of that state in which the execution of the program proceeds after the execution of the method (i.e. in a state that is "executing" or "throwing") rather than of that state in which the execution of the body of the method terminates (which might by e.g. also "returning"). Thus it becomes possible to specify a method as

```
method f(x)
...
implements next.executes AND next.value = x
{ return x }
```

rather than using the formula `next.returns` in the specification.

Soundness Proof Take arbitrary $c \in \mathit{Context}$ and $me \in \mathit{MethodEnv}$ and assume

(1) $\mathit{DifferentVariables}(c)$

(2) $\mathit{specifies}(se, me, c)$

We define

(3) $M := \texttt{method}\ I_m\ (J_1, \ldots, J_p)\ \ S\ \{C\}$

(4) $se' := se[I_m \mapsto \langle (J_1, \ldots, J_p), \langle (I_1, \ldots, I_n), (K_1, \ldots, K_m), F_C, F_R \rangle \rangle]$

(5) $me' := [\![M]\!]^{\mathit{view}(c)}(me)$

We have to show

(a.1) $se' = [\![M]\!]_S(se)$

(a.2) $\mathit{specifies}(se', me', c)$

From (3), (4), hypothesis 1, and the definition of $[\![\_]\!]_S$, we know (a.1).

Take arbitrary $v \in \mathit{View}, b \in \mathit{Behavior}$ and $v_1, \ldots, v_p \in \mathit{Value}, s, s_1, s' \in \mathit{State}$, $e \in \mathit{Environment}$, $c', c'' \in \mathit{Context}, r \in \mathit{StateRelation}, t \in \mathit{StateCondition}$ such that

(6) $\langle v, b \rangle = me'(I_m)$

(7) $\mathit{executes}(\mathit{control}(s))$

(8) $\mathit{DifferentVariables}(c')$

(9) c' EQUALS c

(10) $\mathit{space}(c') \subseteq \mathit{space}(c)$

(11) $\langle r,t\rangle = b^{c'}(v_1,\ldots,v_p)$

(12) $c'' = push(c',J_1,\ldots,J_p)$

(13) $s_1 = writes(s,J_1,v_1,\ldots,J_p,v_p)^{c''}$

(14) $r(s,s')$

To show (a.2), from (2), (3), (4), (5), (6), (7), (8), (9), (10), (11), (12), (13), (14), and the definition of *specifies*, it suffices to show

(a.2.1) $v = view(c)$

(a.2.2) $\{J_1,\ldots,J_p\} \cap \{I_1,\ldots,I_n\} = \emptyset$

(a.2.3) F_C, F_R and T have no free (mathematical or state) variables

(a.2.4) F_C and T do not depend on the poststate

(a.2.5) $[\![F_C \Rightarrow F_R]\!]^{c,c''}(e)(s_1,s')$

(a.2.6) $s = s'$ EXCEPT $range(c') \cup \text{range}(view(c))$

(a.2.7) $s = s'$ EXCEPT^{c,c} $I_1,\ldots,I_n$

(a.2.8) $executes(control(s')) \vee throws(control(s'))$

(a.2.9) $throws(control(s')) \Rightarrow key(control(s')) \in \{K_1,\ldots,K_m\}$

(a.2.10) $[\![F_C]\!]^{c,c''}(e)(s_1,s_1) \Rightarrow t(s)$

From (5), (6), (11), (14), (12), (13), and finally the definition of $[\![\_]\!]$, we know for some $s_0 \in State$

(15) $[\![C]\!]^{c'',me}(s_1,s_0)$

(16) $s' =$ IF $throws(control(s_0))$ THEN s_0 ELSE $executes(s_0)$

(17) $[\![C]\!]_{\mathrm{T}}^{c'',me}(s_1)$

We define

(18) $e_0 := e[I_s \mapsto control(s_0)]$

From (3), (5), (6), and the definition of $[\![\_]\!]$, we know (a.2.1). From hypothesis 2, we know (a.2.2). From hypothesis 3, we know (a.2.3). From hypothesis 4, we know (a.2.4).

In order to show (a.2.5), by hypothesis 11, (7), (18), and the definition of $[\![\_]\!]$, it suffices to show

(a.2.5.1) $executes(control(s_1))$

(a.2.5.2) $[\![F[\#I_s/\texttt{next}]]\!]^{c,c''}(e_0)(s_1,s')$

(a.2.5.3) $\mathit{throws}(\mathit{control}(s_0)) \Rightarrow \mathit{control}(s') = \mathit{control}(s_0)$

(a.2.5.4) $\neg\mathit{throws}(\mathit{control}(s_0)) \Rightarrow$
$\quad \mathit{executes}(\mathit{control}(s')) \wedge \mathit{value}(\mathit{control}(s')) = \mathit{value}(\mathit{control}(s_0))$

From (7), (13), and (CW), we know (a.2.5.1). From (16), we know (a.2.5.3). From (16) and (CD1), we know (a.2.5.4). It remains to show (a.2.5.2).

From (8), (12), and (COP), we know

(19) $\mathit{pushes}(c', c'', \{J_1, \ldots, J_p\})$

From (19) and the definition of *pushes*, we know

(20) $\mathit{DifferentVariables}(c'')$

(21) $c' = c''$ EXCEPT $J_1, \ldots, J_p$

(22) $\{[\![J_1]\!]^{c''}, \ldots, [\![J_p]\!]^{c''}\} \cup \mathit{space}(c'') \subseteq \mathit{space}(c')$

From (9), (21), and the definitions of EQUALS and EXCEPT, we know

(23) $c = c''$ EXCEPT $J_1, \ldots, J_p$

From (10) and (22), we know

(24) $\{[\![J_1]\!]^{c''}, \ldots, [\![J_p]\!]^{c''}\} \cup \mathit{space}(c'') \subseteq \mathit{space}(c)$

From (1), (20), (23), (24), and the definition of *pushes*, we know

(25) $\mathit{pushes}(c, c'', \{J_1, \ldots, J_p\})$

From hypothesis 5, (2), (15), (25), (a.2.5.1), we know

(27) $[F]^{F_c,F_b,F_r,\{L_1,\ldots,L_r\}}_{M_1,\ldots,M_o}$ has no free (mathematical or state) variables

(28) $[\![[F]^{F_c,F_b,F_r,\{L_1,\ldots,L_r\}}_{M_1,\ldots,M_o}]\!]^{c,c''}(e)(s_1,s_0)$

From (28) and the definition of $[\![\,\_\,]\!]$, we know

(29) $[\![F]\!]^{c,c''}(e)(s_1,s_0)$

(30) $s_1 = s_0$ EXCEPT$^{c,c''}$ $M_1, \ldots, M_o$

(31) $\mathit{throws}(\mathit{control}(s_0)) \Rightarrow \mathit{key}(\mathit{control}(s_0)) \in \{L_1, \ldots, L_r\}$

From (29), hypothesis 10, and (CNEF2), we know

(32) $[\![F[\#I_s/\texttt{next}]]\!]^{c,c''}(e_0)(s_1,s_0)$

From (16), (REE), (NEQ), and (CD2), we know

(33) $s' \text{ EQUALS}^{c,c''} s_0$

From (33), (CD0), and (CD4), we know

(34) $s' = (store(s), control(s'))$

From (32), (34), (REE), (NEQ), (CNEF1), and (PVFNE), we know (a.2.5.2).
From (2) and the definition of *specifies*, we know

(35) $\text{range}(view(c)) = \bigcup_{I \in \text{Identifier}} \text{range}(view(me(I)))$

From (15), (20), (35), and Lemma "Commands with Methods", we know

(36) $s_1 = s_0 \text{ EXCEPT } range(c'') \cup \text{range}(view(c))$

From (13) and the definition of *writes*, we know

(37) $\forall x \in Variable : store(s)(x) \neq store(s_1)(x) \Rightarrow x \in range(c'')$

From (16), (CD2), and (CD3), we know

(38) $store(s_0) = store(s')$

From (36), (37), (38), and the definition of EXCEPT, we know

(39) $s = s' \text{ EXCEPT } range(c'') \cup \text{range}(view(c))$

From (12) and Lemma "Range of Contexts", we know

(40) $range(c'') \subseteq range(c')$

From (39), (40), and the definition of EXCEPT, we know (a.2.6).
Assume we have the auxiliary proposition (shown below)

(a.3)
$$[\![[\texttt{EXISTS } \$J_1, \ldots, \$J_p, \$L_1, \ldots, \$L_p : F[\$J_1/J_1, \ldots, \$J_p/J_p, \$L_1/J_1\texttt{'}, \ldots, \$L_p/J_p\texttt{'}]]^{F_c,F_b,F_r,\{L_1,\ldots,L_r\}}_{\{M_1,\ldots,M_o\}\setminus\{J_1,\ldots,J_p\}}]\!]^{c,c}(e)(s,s')$$

From hypothesis 8, (7), and (a.3), we then know

(41) $\neg \mathit{continues}(\mathit{control}(s'))$

(42) $\neg \mathit{breaks}(\mathit{control}(s'))$

(43) $s = s' \text{ EXCEPT}^{c,c}\ I_1, \ldots, I_n$

(44) $\mathit{throws}(\mathit{control}(s')) \Rightarrow \mathit{key}(\mathit{control}(s')) \in \{K_1, \ldots, K_m\}$

From (43), we know (a.2.7).

From (16) and (CD1), we know (a.2.8).

From (44), we know (a.2.9).

To show (a.2.10), we assume

(45) $[\![F_C]\!]^{c,c''}(e)(s_1, s_1)$

and show

(a.2.10.a) $t(s)$

From (5), (6), (11), (12), (13), and the definition of $[\![\_]\!]$, it suffices to show

(a.2.10.b) $[\![C]\!]_{\mathrm{T}}^{c'',me}(s_1)$

From hypothesis 12, (2), (25), (45), (a.2.3), (a.2.4), and finally (a.2.5.1), we know (a.2.10.b).

To show (a.3), from (46) and the definition of $[\![\_]\!]$, it suffices to show

(a.3.1) $[\![\texttt{EXISTS } \$J_1, \ldots, \$J_p, \$L_1, \ldots, \$L_p : F[\$J_1/J_1, \ldots, \$J_p/J_p, \$L_1/J_1\texttt{'}, \ldots, \$L_p/J_p\texttt{'}]]\!]^{c,c}(e)(s, s')$

(a.3.2) $s = s' \text{ EXCEPT}^{c,c}\ \{M_1, \ldots, M_o\} \backslash \{J_1, \ldots, J_p\}$

(a.3.3) $\mathit{continues}(\mathit{control}(s')) \Rightarrow [\![F_c]\!]^{c,c}(e)(s, s')$

(a.3.4) $\mathit{breaks}(\mathit{control}(s')) \Rightarrow [\![F_b]\!]^{c,c}(e)(s, s')$

(a.3.5) $\mathit{returns}(\mathit{control}(s')) \Rightarrow [\![F_r]\!]^{c,c}(e)(s, s')$

(a.3.6) $\mathit{throws}(\mathit{control}(s')) \Rightarrow \mathit{key}(\mathit{control}(s')) \in \{L_1, \ldots, L_r\}$

From hypothesis 6, hypothesis 7, (23), (29), and (COF2), we know (a.3.1).

From (1), (25), (30) and Lemma "Changing Local Variables", we know

(46) $s_1 = s_0 \text{ EXCEPT}^{c,c}\ \{M_1, \ldots, M_o\} \backslash \{J_1, \ldots, J_p\}$

From (16), (REE), (NEQ), and (CD2), we know

(47) s' EQUALS^{c,c} s_0

From (13) and the definition of *writes*, we know

(48) $s_1 = s$ EXCEPT $\{[\![J_1]\!]^{c''}, \ldots, [\![J_p]\!]^{c''}\}$

From (1), (10), (12), and the definition of *push*, we know

(49) $\{[\![J_1]\!]^{c''}, \ldots, [\![J_p]\!]^{c''}\} \subseteq space(c)$

From (48), (49), and the definitions of EXCEPT and EQUALS, we know

(50) s_1 EQUALS^{c,c} s

From (46), (47), (50), (TRE), and (NEQ), we know (a.3.2).

From (16), (CD1), and Lemma "State Control Predicates", we know (a.3.3.), (a.3.4), and (a.3.5).

From (16), (31), (CD1), and Lemma and Lemma "State Control Predicates", we know (a.3.6). □

6.6.2 Verification of Programs

$$\frac{\begin{array}{l} se \vdash Ms : se' \\ S = \texttt{writesonly}\ I_1,\ldots,I_n\ \texttt{throwsonly}\ K_1,\ldots,K_m \\ \quad \texttt{assumes}\ F_C\ \texttt{implements}\ F_R\ \texttt{decreases}\ T \\ F_C, F_R \text{ and } T \text{ have no free (mathematical or state) variables} \\ F_C \text{ and } T \text{ do not depend on the poststate} \\ se', \emptyset \vdash C : [F]^{F_c,F_b,F_r,\{L_1,\ldots,L_r\}}_{M_1,\ldots,M_o} \\ [F]^{F_c,F_b,F_r,\{L_1,\ldots,L_r\}}_{M_1,\ldots,M_o} \ \S\ ^{K_1,\ldots,K_m}_{I_1,\ldots,I_n} \\ se', \emptyset \vdash C \checkmark F_C \\ \forall s, s' \in State, e \in Environment, c_0, c_1 \in Context : \\ \quad [\![(\texttt{now.executes}\ \texttt{AND}\ F) \Rightarrow (F_C \Rightarrow F_R)]\!]^{c_0,c_1}(e)(s,s') \\ se', \emptyset \vdash C \downarrow F_C \end{array}}{se \vdash Ms\ S\ \{C\}}$$

This rule is the "top-level" rule of our reasoning calculus. It claims correctness for the program $Ms\ S\ \{C\}$ in specification environment *se* provided that:

- $se \vdash Ms : se'$: the sequence of declarations Ms gives rise to the declaration environment se'.
- $S = \ldots$: the specification S of the program body C consists of the denoted components where F_C and F_R must be appropriately formed.
- $se', \emptyset \vdash C : [F]^{F_c,F_b,F_r,\{L_1,\ldots,L_r\}}_{M_1,\ldots,M_o}$: within the specification environment se' and a context with no local variables, C induces the state relation formula $[F]^{F_c,F_b,F_r,\{L_1,\ldots,L_r\}}_{M_1,\ldots,M_o}$.
- $[F]^{F_c,F_b,F_r,\{L_1,\ldots,L_r\}}_{M_1,\ldots,M_o}$ § $^{K_1,\ldots,K_m}_{I_1,\ldots,I_n}$: the state relation of C entails the frame condition and the exception condition of the specification formula of C.
- $se', \emptyset \vdash C \checkmark F_C$: C is well-defined with respect to the precondition F_C of the specification formula.
- $\forall \ldots : [\![(\texttt{now.executes AND } F) \Rightarrow (F_C \Rightarrow F_R)]\!]^{c_0,c_1}(e)(s,s')$: the state relation F induced by C entails the specification condition $F_C \Rightarrow F_R$.
- $se', \emptyset \vdash C \downarrow F_C$: C terminates provided that the precondition F_C of the specification formula holds.

Among these conditions, only the last four ones give rise to actual "reasoning tasks". Furthermore, the triple role of F_C should be noted:

- as the precondition for verifying the well-definedness of C,
- as the hypothesis in the specification condition of C,
- as the precondition for verifying the termination of C.

Logically, these three roles are independent and can be served by different formulas. However, from the pragmatic point of view (because it reflects the most typical case), we find it more appropriate to cover these roles by a single precondition F_C; anyway, this design decision has no deeper impact on the calculus and can be easily reverted.

As previously discussed, the soundness of the rule (and the corresponding proof) does not depend on the hypothesis $se', \emptyset \vdash C \checkmark F_C$ because the soundness claim does not state that the execution of C does only encounter expressions with well-defined values. Furthermore, as discussed in Section 5.9.4, the verification of $se, \emptyset \vdash C \checkmark F_C$ may be omitted if we rather choose to verify a transformed command C' with runtime checks for undefined expressions inserted.

Soundness Proof Take arbitrary $c \in \mathit{Context}$ and $me, me' \in \mathit{MethodEnv}$ and assume

(1) $\mathit{DifferentVariables}(c)$

(2) $\mathit{specifies}(se, me, c)$

(3) $me' = [\![Ms]\!]^{\mathit{view}(c)}(me)$

We have to show

(a) $\mathit{specifies}([\![Ms]\!]_{\mathrm{S}}(se), me')$

(b) $\mathit{correctness}([\![S]\!], [\![C]\!]^{c,me'}, [\![C]\!]^{c,me'}_{\mathrm{T}}, c)$

From hypothesis 1, we know with (1) and (2)

(4) $se' = [\![Ms]\!]_{\mathrm{S}}(se)$

(5) $\mathit{specifies}(se', [\![Ms]\!]^{\mathit{view}(c)}(me), c)$

From (3), (4), and (5), we know (a).

To show (b), from hypothesis 2 and the definitions of $[\![\_]\!]$ and *correctness*, it suffices to assume for arbitrary $s, s' \in \mathit{State}, e \in \mathit{Environment}$

(6) $\mathit{executes}(\mathit{control}(s))$

(7) $[\![C]\!]^{c,me'}(s, s')$

and show

(b.1) $[\![F_C \mathrel{=>} F_R]\!]^{c,c}(e)(s, s')$

(b.2) $s = s'$ EXCEPT^{c,c} $I_1, \ldots, I_n$

(b.3) $\neg \mathit{continues}(\mathit{control}(s'))$

(b.4) $\neg \mathit{breaks}(\mathit{control}(s'))$

(b.5) $\mathit{throws}(\mathit{control}(s')) \Rightarrow \mathit{key}(\mathit{control}(s')) \in \{K_1, \ldots, K_m\}$

(b.6) $[\![F_C]\!]^{c,c}(e)(s, s) \Rightarrow [\![C]\!]^{c,me'}_{\mathrm{T}}(s)$

From (1) and the definition of *pushes*, we know

(8) $\mathit{pushes}(c, c, \emptyset)$

From hypothesis 5, we know with (3), (5), (6), (7), and (8)

(9) $[F]^{F_c,F_b,F_r,\{L_1,\ldots,L_r\}}_{M_1,\ldots,M_o}$ has no free (mathematical or state) variables

(10) $[\![[F]^{F_c,F_b,F_r,\{L_1,\ldots,L_r\}}_{M_1,\ldots,M_o}]\!]^{c,c}(e)(s,s')$

From (9) and (10), we know by the definitions of $[\_]^{\_}_{\_}$ and $[\![\_]\!]$

(11) F has no free (mathematical or state) variables

(12) $[\![F]\!]^{c,c}(e)(s,s')$

(13) $s = s'$ EXCEPT^{c,c} $M_1,\ldots,M_o$

(14) $continues(control(s')) \Rightarrow [\![F_c]\!]^{c,c}(e)(s,s')$

(15) $breaks(control(s')) \Rightarrow [\![F_c]\!]^{c,c}(e)(s,s')$

(16) $returns(control(s')) \Rightarrow [\![F_r]\!]^{c,c}(e)(s,s')$

(17) $throws(control(s')) \Rightarrow key(control(s')) \in \{L_1,\ldots,L_r\}$

From (6), (12), hypothesis 8 and the definition of $[\![\_]\!]$, we know (b.1).

From (6), (10), and hypothesis 6, we know

(18) $\neg continues(control(s'))$

(19) $\neg breaks(control(s'))$

(20) $s = s'$ EXCEPT^{c,c} $I_1,\ldots,I_n$

(21) $throws(control(s')) \Rightarrow key(control(s')) \in \{K_1,\ldots,K_m\}$

From (20), we know (b.2). From (18), we know (b.3). From (19), we know (b.4). From (21), we know (b.5).

From hypotheses 3, 4, 9, (3), (5), (6), and (8), we know (b.6). □

6.7 Recursion

The semantics of programs presented in Figure 6.8 allows every method M to call only those methods that are visible at the point of the declaration of M i.e. that are declared before M. This rules out that M calls itself (direct recursion) or that M calls methods that are declared after M and in turn might call M (indirect recursion). In this section, we get rid of this restriction: we allow every method to call every other method (including itself) i.e. we discuss the modeling of and reasoning about programs with directly or indirectly recursive methods.

The core idea is that we describe the behavior of a recursive method M by an infinite sequence of non-recursive methods that is constructed by unfolding the

Recursion Language: Definitions

$RecBehavior_0 := Behavior$
$RecMethodEnv_0 := MethodEnv$

$RecBehavior_{i+1} := Context \times RecMethodEnv_i \times Value^* \to$
$\quad (StateRelation \times StateCondition)$
$RecMethodEnv_{i+1} :=$
$\quad \text{Identifier} \to (View \times RecMethodEnv_i \times RecBehavior_{i+1})$

$RecMethodEnvs_i := \bigcup_{j \in \mathbb{N}, j \leq i} RecMethodEnv_j$
$RecMethodEnvs := \bigcup_{i \in \mathbb{N}} RecMethodEnv_i$

Figure 6.36: Definitions for Programs with Recursion (1/2)

recursive calls of M: a method M_{i+1} in that sequence may call its predecessor M_i which in turn may call its predecessor; however, calling the base method M_0 gives rise to a non-terminating computation. Any application of the recursive method M in a prestate s is considered to terminate normally with a poststate s' if the application of *some* non-recursive method M_i in s terminates with s' (which corresponds to an execution of M with less than i recursive invocations).

Since method M_{i+1} may call method M_i, it is given an environment that holds the behavior of M_i; we denote the type of such a "recursive method environment" as $RecMethodEnv_i$ and the type of the "recursive behavior" of M_i as $RecBehavior_i$. A recursive method environment of type $RecBehavior_{i+1}$ maps every method name to a triple $\langle v, me, b \rangle$ where b is a method behavior b type $RecBehavior_{i+1}$ which receives from the caller as arguments (in addition to the value arguments) the variable view v and the method environment me of type $RecMethodEnv_i$. The base types $RecMethodEnv_0$ and $RecBehavior_0$ are identified with $Behavior$ and $MethodEnv$; they can be considered to describe all "pre-defined" methods that may be called by the "user-defined" methods contained in the program. The environment $RecMethodEnvs_i$ holds the behaviors of all methods M_j with $j \leq i$, likewise $RecMethodEnvs$ holds the behaviors of all M_j. The corresponding definitions are given in Figure 6.36.

Given a view v in which methods are declared, a method environment me of pre-defined methods, and a sequence of user-defined methods Ms, we can thus construct the infinite sequence $envSeq^{v,me,Ms}$ of method environments: each method in position $i+1$ of that sequence may call each method in position i; at position 0, the sequence holds the behaviors of all pre-defined methods respectively, for all

Recursion Language: Definitions (Contd)

$[\![\_]\!]_{\mathrm{I}} : \text{Method} \to \text{Identifier}$
$[\![\texttt{method}\ I_m\ (J_1,\ldots,J_p)\ S\ \{C\}]\!]_{\mathrm{I}} := I_m$

$[\![\_]\!]_{\mathrm{I}} : \text{Methods} \to \mathbb{P}(\text{Identifier})$
$[\![\_]\!]_{\mathrm{I}} := \emptyset$
$[\![Ms\ M]\!]_{\mathrm{I}} := [\![Ms]\!]_{\mathrm{I}} \cup \{[\![M]\!]_{\mathrm{I}}\}$

$envBase : View \times MethodEnv \times \mathbb{P}(\text{Identifier}) \to RecMethodEnv_0$
$envBase^{v,me,Is}(I) :=$
  IF $I \in Is$ THEN
    LET
      $b : RecBehavior_0$
      $b^c(v_1,\ldots,v_p) := \langle \emptyset, \emptyset \rangle$
    IN $\langle v, b \rangle$
  ELSE $me(I)$

$envNext_i : View \times RecMethodEnv_i \times \to RecMethodEnv_{i+1}$
$envNext_0^{v,me,}(I) :=$
  LET
    $b : RecBehavior_1$
    $b^{c,me}(v_1,\ldots,v_p) :=$
      LET $\langle v, b_0 \rangle = me(I)$ IN $b_0^{c,me}(v_1,\ldots,v_p)$
  IN $\langle v, me, b \rangle$
$envNext_{i+1}^{v,me,}(I) :=$
  LET
    $b : RecBehavior_{i+2}$
    $b^{c,me}(v_1,\ldots,v_p) :=$
      LET $\langle v, me_0, b_0 \rangle = me(I)$ IN $b_0^{c,me_0}(v_1,\ldots,v_p)$
  IN $\langle v, me, b \rangle$

$envSeq : View \times MethodEnv \times \text{Methods} \to RecMethodEnvs^{\infty}$
$envSeq^{v,me,Ms}(0) = envBase^{v,me,[\![Ms]\!]_{\mathrm{I}}}$
$envSeq^{v,me,Ms}(i+1) =$
  LET $me' = envSeq^{v,me,Ms}(i)$ IN
  $[\![Ms]\!]_i^v(me', envNext_i^{v,me'})$

Figure 6.37: Definitions for Programs with Recursion (2/2)

Recursion Language: Valuation Functions

$$[\![\_]\!] : \textbf{Program} \to \textit{Context} \to \textit{MethodEnv} \to (\textit{StateRelation} \times \textit{StateCondition})$$
$$[\![\textit{Ms}\ S\ \{C\}]\!]^c(me) =$$
LET
$$\quad mes = envSeq^{view(c),me,Ms}$$
$$\quad R : StateRelation, R(s,s') \Leftrightarrow \exists i \in \mathbb{N} : [\![C]\!]_i^{c,mes(i)}(s,s')$$
$$\quad T : StateCondition, T(s) \Leftrightarrow \exists i \in \mathbb{N} : [\![C]\!]_{\mathrm{T}_i}^{c,mes(i)}(s)$$
IN $\langle R, T \rangle$

$$[\![\_]\!]_i : \textbf{Methods} \to \textit{View} \to (\textit{RecMethodEnvs}_i \times \textit{RecMethodEnvs}_{i+1}) \to \textit{RecMethodEnvs}_{i+1}$$
$$[\![\_]\!]_i^v(me, me') = me'$$
$$[\![\textit{Ms}\ M]\!]_i^v(me, me') = [\![M]\!]_i^v(me, [\![\textit{Ms}]\!]_i^v(me, me'))$$

Figure 6.38: Programs with Recursion (1/3)

user-defined methods, non-terminating behaviors. The corresponding definitions are given in Figure 6.37.

Based on these definitions, the semantics of programs with recursion is stated in Figure 6.38. Given a prestate s, the program terminates in a post-state s', if there is some position i in the sequence of method environments mes such that the program when executed with the method environment $mes(i)$ in prestate s terminates with poststate s'. The sequence of method declarations Ms in the program is used to construct, starting with the base environment $mes(0)$, from every method environment $mes(i)$ it successor $mes(i+1)$.

Figure 6.39 describes the semantics of the declaration of a method with name I_m. Given the current variable view v, the environment me of methods that may be called by I_m, and the environment me' that is to be updated by the declaration, a behavior b is constructed and me' is updated by mapping I_m to the triple $\langle v, me, b \rangle$. When b is invoked, it receives as arguments (in addition to the argument values $v_1, \ldots, v_p$) a context c (with view v) and me such that it can access the variables and invoke the methods that were visible at the point of the method declaration.

Figure 6.40 depicts the updated semantics of the call of a method I_m in context c and method environment me. We look up $me(I_m)$ to get the triple $\langle v, me', b \rangle$ describing the method and then invoke $b^{call(v,c),me'}$ with the given argument values.

Recursion Language: Valuation Functions (Contd)

$$[\![\_]\!]_i : \textbf{Method} \to \textit{View} \to (\textit{RecMethodEnvs}_i \times \textit{RecMethodEnvs}_{i+1}) \to \textit{RecMethodEnvs}_{i+1}$$

$$[\![\texttt{method}\ I_m\ (J_1,\ldots,J_p)\ \ S\ \{C\}]\!]_i^v(me,me') =$$
LET
  $b = [\![\texttt{method}\ I_m\ (J_1,\ldots,J_p)\ \ S\ \{C\}]\!]_i$
IN $me'[I_m \mapsto \langle v, me, b\rangle]$

$$[\![\_]\!]_i : \textbf{Method} \to \textit{RecBehavior}_{i+1}$$

$$[\![\texttt{method}\ I_m\ (J_1,\ldots,J_p)\ \ S\ \{C\}]\!]_i =$$
LET
  $b : \textit{RecBehavior}_i$
  $b^{c,me}(v_1,\ldots,v_p) =$
    LET
      $c' = \textit{push}(c, J_1, \ldots, J_p)$
      $r \in \textit{StateRelation}$
      $r(s,s') \Leftrightarrow$
        $\exists s_0 : \textit{State} :$
          $[\![C]\!]_i^{c',me}(\textit{writes}(s, J_1, v_1, \ldots, J_p, v_p)^{c'}, s_0) \wedge$
          $s' =$ IF $\textit{throws}(\textit{control}(s_0))$
                 THEN s_0 ELSE $\textit{executes}(s_0)$
      $t \in \textit{StateCondition}$
      $t(s) \Leftrightarrow$
        $[\![C]\!]_{\mathrm{T}_i}^{c',me}(\textit{writes}(s, J_1, v_1, \ldots, J_p, v_p)^{c'})$
    IN $\langle r, t\rangle$
IN b

Figure 6.39: Programs with Recursion (2/3)

Recursion Language: Valuation Functions (Contd)

$[\![\_]\!]_i$: **Command** $\rightarrow$
$(\textbf{\textit{Context}} \times \textbf{\textit{RecMethodEnvs}}_i) \rightarrow \textbf{\textit{StateRelation}}$

$[\![\ldots]\!]_i^{c,me}(s,s') \Leftrightarrow \ldots$
$[\![I_r = I_m\ (E_1,\ldots,E_p)\ ]\!]_0^{c,me}(s,s') \Leftrightarrow$
  LET $\langle v,b\rangle = me(I_m)$ IN
  LET $\langle r,t\rangle = b^{call(v,c)}([\![E_1]\!]^c(s),\ldots,[\![E_p]\!]^c(s))$ IN
  $\exists s_0 \in State : r(s,s_0) \wedge$
    IF $throws(control(s_0))$
      THEN $s' = s_0$
      ELSE $s' = write(s_0,I_r,value(control(s_0)))^c$
$[\![I_r = I_m\ (E_1,\ldots,E_p)\ ]\!]_{i+1}^{c,me}(s,s') \Leftrightarrow$
  LET $\langle v,me',b\rangle = me(I_m)$ IN
  LET $\langle r,t\rangle = b^{call(v,c),me'}([\![E_1]\!]^c(s),\ldots,[\![E_p]\!]^c(s))$ IN
  $\exists s_0 \in State : r(s,s_0) \wedge$
    IF $throws(control(s_0))$
      THEN $s' = s_0$
      ELSE $s' = write(s_0,I_r,value(control(s_0)))^c$

$[\![\_]\!]_{\mathrm{T}_i}$: **Command** $\rightarrow$
$(\textbf{\textit{Context}} \times \textbf{\textit{RecMethodEnvs}}_i) \rightarrow \textbf{\textit{StateCondition}}$

$[\![\ldots]\!]_{\mathrm{T}_i}^{c,me}(s) \Leftrightarrow \ldots$
$[\![I_r = I_m\ (E_1,\ldots,E_p)\ ]\!]_{\mathrm{T}_0}^{c,me}(s) \Leftrightarrow$
  LET $\langle v,b\rangle = me(I_m)$ IN
  LET $\langle r,t\rangle = b^{call(v,c)}([\![E_1]\!]^c(s),\ldots,[\![E_p]\!]^c(s))$ IN
  $t(s)$
$[\![I_r = I_m\ (E_1,\ldots,E_p)\ ]\!]_{\mathrm{T}_{i+1}}^{c,me}(s) \Leftrightarrow$
  LET $\langle v,me',b\rangle = me(I_m)$ IN
  LET $\langle r,t\rangle = b^{call(v,c),me'}([\![E_1]\!]^c(s),\ldots,[\![E_p]\!]^c(s))$ IN
  $t(s)$

Figure 6.40: Programs with Recursion (3/3)

Recursion Language: Definitions

F makes $\$I$ a natural number $\equiv$
$\quad \forall e \in \mathit{Environment}, c, c' \in \mathit{Context}, s, s' \in \mathit{State}:$
$\quad\quad [\![F]\!]^{c,c'}(e)(s,s') \Rightarrow e(I) \in \mathbb{N}$

F makes T a natural number $\equiv$
$\quad \forall e \in \mathit{Environment}, c, c' \in \mathit{Context}, s, s' \in \mathit{State}:$
$\quad\quad [\![F]\!]^{c,c'}(e)(s,s') \Rightarrow [\![T]\!]^{c,c'}(e)(s,s') \in \mathbb{N}$

$\mathit{wellformed} \subseteq \mathit{MethodSpec}$
$\mathit{wellformed}(\langle (J_1,\ldots,J_p), \langle (I_1,\ldots,I_n), (K_1,\ldots,K_m), F_C, F_R, T\rangle\rangle) \Leftrightarrow$
$\quad \{J_1,\ldots,J_p\} \cap \{I_1,\ldots,I_n\} = \emptyset \wedge$
$\quad F_C, F_R$ and T have no free (mathematical or state) variables $\wedge$
$\quad F_C$ and T do not depend on the poststate $\wedge$
$\quad F_C$ makes T a natural number

Figure 6.41: Definitions for the Recursion Language (1/4)

The core problem in the calculus of the recursion language is to ensure that a method terminates in a finite number of recursive invocations (analogous to a loop that terminates in a finite number of iterations). For this purpose, every method specification is accompanied by a termination term T that denotes (in the context of the precondition F_C) a natural number. For every invocation of a method this value must be decreased; consequently the value of T represents a bound for the number of recursive invocations.

In order to formulate this calculus and state its soundness, we need a couple of auxiliary definitions. First, Figure 6.41 introduces a predicate *wellformed* that constrains the construction of method specifications; on this basis, Figure 6.42 defines the predicate $\mathit{rspecifies}_i$ which states that a recursive method named I with behavior b of type $\mathit{RecBehavior}_{i+1}$ is correctly described by a specification.

Then Figure 6.43 introduces a predicate $\mathit{rspecifies}_i$ which states that a specification environment *se* correctly describes a method environment *me* of type $\mathit{RecMethodEnvs}_i$. On this basis, $\mathit{rspecifies}(se, M, Is)$ states that *se* correctly specifies a method declaration M where Is are the names of all methods declared in the program; likewise, the predicate $\mathit{rspecifies}(se, Ms, Is)$ states that *se* correctly specifies a sequence of method declarations Ms.

Finally, Figure 6.44 defines a predicate *correctness* that states the core correctness

Recursion Language: Definitions (Contd)

$$
\begin{array}{l}
\mathit{rspecifies}_i : \mathit{MethodSpec} \times \\
\quad \mathit{RecBehavior}_{i+1} \times \mathit{Context} \times \mathit{RecMethodEnv}_i \times \\
\quad \mathbb{P}(\text{Identifier}) \times \text{Identifier} \\
\mathit{rspecifies}_i(\langle(J_1,\ldots,J_p),\langle(I_1,\ldots,I_n),(K_1,\ldots,K_m),F_C,F_R,T\rangle\rangle, \\
\quad b,c,me,Is,I) \Leftrightarrow \\
\quad \mathit{wellformed}(\langle(J_1,\ldots,J_p),\langle(I_1,\ldots,I_n),(K_1,\ldots,K_m),F_C,F_R,T\rangle\rangle) \\
\quad \wedge \\
\quad \forall v_1,\ldots,v_p \in \mathit{Value}, s \in \mathit{State}, e \in \mathit{Environment}, c' \in \mathit{Context} : \\
\quad\quad \mathit{executes}(\mathit{control}(s)) \wedge \mathit{DifferentVariables}(c') \wedge \\
\quad\quad c' \text{ EQUALS } c \wedge\ \mathit{space}(c') \subseteq \mathit{space}(c)\ \Rightarrow \\
\quad\quad \text{LET} \\
\quad\quad\quad \langle r,t\rangle = b^{c',me}(v_1,\ldots,v_p), \\
\quad\quad\quad c'' = \mathit{push}(c',J_1,\ldots,J_p), \\
\quad\quad\quad s_1 = \mathit{writes}(s,J_1,v_1,\ldots,J_p,v_p)^{c''} \\
\quad\quad\quad m = [\![T]\!]^{c,c''}(e)(s_1,s_1) \\
\quad\quad \text{IN } (\forall s' \in \mathit{State} : r(s,s')\ \Rightarrow \\
\quad\quad\quad\quad [\![F_C \Rightarrow F_R]\!]^{c,c''}(e)(s_1,s') \wedge \\
\quad\quad\quad\quad s = s' \text{ EXCEPT } \mathit{range}(c') \cup \text{range}(\mathit{view}(c)) \wedge \\
\quad\quad\quad\quad s = s' \text{ EXCEPT}^{c,c}\ I_1,\ldots,I_n \wedge \\
\quad\quad\quad\quad (\mathit{executes}(\mathit{control}(s')) \vee \mathit{throws}(\mathit{control}(s'))) \wedge \\
\quad\quad\quad\quad (\mathit{throws}(\mathit{control}(s'))\ \Rightarrow \\
\quad\quad\quad\quad\quad \mathit{key}(\mathit{control}(s')) \in \{K_1,\ldots,K_m\})) \\
\quad\quad \wedge\ ([\![F_C]\!]^{c,c''}(e)(s_1,s_1) \wedge (I \in Is \Rightarrow m \in \mathbb{N} \wedge m < i) \Rightarrow t(s))
\end{array}
$$

Figure 6.42: Definitions for the Recursion Language (2/4)

claim for the specification of a whole program.

In the following, we give a crucial relationship between the predicate *rspecifies* and the predicate $rspecifies_i$ which will become important later in proving the soundness of the calculus of the recursion language.

Lemma (Specification of Recursively Defined Functions) If a method is well-specified, it is well-specified with respect to every environment in the sequence of method environments constructed from the method declarations:

$$\begin{array}{l}\forall se \in SpecEnv, Ms \in \text{Methods}: \\ \quad rspecifies(se, Ms, [\![Ms]\!]_{\mathrm{I}}) \Rightarrow \\ \quad\quad \forall c \in Context, me \in MethodEnv: \\ \quad\quad\quad DifferentVariables(c) \wedge specifies(se, me, c) \Rightarrow \\ \quad\quad\quad\quad \text{LET } se' = [\![Ms]\!]_{\mathrm{S}}(se), mes := envSeq^{view(c), me, Ms} \text{ IN} \\ \quad\quad\quad\quad \forall i \in \mathbb{N}: rspecifies_i(se', mes(i), c, [\![Ms]\!]_{\mathrm{I}})\end{array}$$

Proof Take arbitrary $se \in SpecEnv, Ms \in \text{Methods}$ as well as $c \in Context, me \in MethodEnv$ and define

(1) $se' = [\![Ms]\!]_{\mathrm{S}}(se)$

(2) $mes := envSeq^{view(c), me, Ms}$

We assume

(3) $rspecifies(se, Ms, [\![Ms]\!]_{\mathrm{I}})$

(4) $DifferentVariables(c)$

(5) $specifies(se, me, c)$

We show

(a) $\forall i \in \mathbb{N}: rspecifies_i(se', mes(i), c, Is)$

by induction on i.

As the induction base, we show

(a.1) $rspecifies_0(se', mes(0), c, Is)$

From (2) and the definition of *envSeq*, it suffices to show

(a.1.a) $rspecifies_0(se', envBase^{view(c), me, [\![Ms]\!]_{\mathrm{I}}}, c, [\![Ms]\!]_{\mathrm{I}})$

Recursion Language: Definitions (Contd)

$$
\begin{array}{l}
rspecifies_i \subseteq SpecEnv \times RecMethodEnvs_i \times Context \times \mathbb{P}(\text{Identifier}) \\
rspecifies_0(se, me, c, Is) \Leftrightarrow \\
\quad \forall I \in \text{Identifier} : \\
\qquad \text{LET } \langle v, b \rangle = me(I) \text{ IN} \\
\qquad v = view(c) \wedge \\
\qquad \text{IF } I \in Is \text{ THEN} \\
\qquad\quad wellformed(se(I)) \wedge \\
\qquad\quad \forall c \in Context, v_1, \ldots, v_p \in Value : b^c(v_1, \ldots, v_p) = \langle \emptyset, \emptyset \rangle \\
\qquad \text{ELSE} \\
\qquad\quad specifies(se(I), b, c) \\
rspecifies_{i+1}(se, me, c, Is) \Leftrightarrow \\
\quad \forall I \in \text{Identifier} : \\
\qquad \text{LET } \langle v, me', b \rangle = me(I) \text{ IN} \\
\qquad v = view(c) \wedge rspecifies_i(se(I), b, c, me', Is, I)
\end{array}
$$

$$
\begin{array}{l}
rspecifies \subseteq SpecEnv \times \text{Method} \times \mathbb{P}(\text{Identifier}) \\
rspecifies(se, M, Is) \Leftrightarrow \\
\quad \forall c \in Context : DifferentVariables(c) \Rightarrow \\
\qquad \forall i \in \mathbb{N}, me \in RecMethodEnvs_i, me' \in RecMethodEnvs_{i+1} : \\
\qquad\quad rspecifies_i(se, me, c, Is) \wedge rspecifies_{i+1}(se, me', c, Is) \Rightarrow \\
\qquad\qquad rspecifies_{i+1}(se, [\![M]\!]_i^{view(c)}(me, me'), c, Is)
\end{array}
$$

$$
\begin{array}{l}
rspecifies \subseteq SpecEnv \times Methods \times \mathbb{P}(\text{Identifier}) \\
rspecifies(se, Ms, Is) \Leftrightarrow \\
\quad \forall c \in Context : DifferentVariables(c) \Rightarrow \\
\qquad \forall i \in \mathbb{N}, me \in RecMethodEnvs_i : \\
\qquad\quad rspecifies_i(se, me, c, Is) \Rightarrow \\
\qquad\qquad \text{LET } me' = [\![Ms]\!]_i^{view(c)}(me, envNext_i^{view(c), me}) \text{ IN} \\
\qquad\qquad rspecifies_{i+1}(se, me', c, Is)
\end{array}
$$

Figure 6.43: Definitions for the Recursion Language (3/4)

Recursion Language: Definitions (Contd)

$$assumption : Spec \to Formula$$
$$assumption(\langle (I_1,\ldots,I_n),(K_1,\ldots,K_m),F_C,F_R,T\rangle) = F_C$$

$$correctness \subseteq Spec \times Context \times Environment \times State \times State$$
$$\begin{aligned} correctness(\langle (I_1,\ldots,I_n),(K_1,\ldots,K_m),F_C,F_R,T\rangle,c,e,s,s') \Leftrightarrow \\ [\![F_C \Rightarrow F_R]\!]^{c,c}(e)(s,s') \wedge \\ s = s' \text{ EXCEPT}^{c,c}\ I_1,\ldots,I_n \wedge \\ \neg continues(control(s')) \wedge \\ \neg breaks(control(s')) \wedge \\ (throws(control(s')) \Rightarrow key(control(s')) \in \{K_1,\ldots,K_m\}) \end{aligned}$$

Figure 6.44: Definitions for the Recursion Language (4/4)

By the definition of $rspecifies_0$, to show (a.1.a), it suffices to take arbitrary $I \in$ Identifier and $v \in View, b \in Behavior$ such that

(6) $\langle v,b\rangle = envBase^{view(c),me,[\![Ms]\!]_I}(I)$

and show

(a.1.a.1) $v = view(c)$

(a.1.a.2) $I \in [\![Ms]\!]_I \Rightarrow \forall c \in Context, v_1,\ldots,v_p \in Value : b^c(v_1,\ldots,v_p) = \langle \emptyset,\emptyset\rangle$

(a.1.a.3) $I \notin [\![Ms]\!]_I \Rightarrow specifies(se'(I),b,c)$

If $I \in [\![Ms]\!]_I$, we know (a.1.a.3), and, from (6), (7), and the definition of *envBase*, also (a.1.a.2).

Thus we may assume

(7) $I \notin [\![Ms]\!]_I$

From (6), (7), and the definition of *envBase*, we know

(8) $\langle v,b\rangle \in me(I)$

From (5), (8), and the definition of *specifies*, we know (a.1.a.1).

From (7), we know (a.1.a.2).

From (1), (6), and the definition of $[\![\_]\!]_S$, we know

(9) $se(I) = se'(I)$

From (5), (8), (9), and the definition of *specifies*, we know (a.1.a.3).

As for the induction step, we assume

(10) $rspecifies_i(se', mes(i), c, [\![Ms]\!]_{\mathrm{I}})$

and show

(a.2) $rspecifies_{i+1}(se', mes(i+1), c, [\![Ms]\!]_{\mathrm{I}})$

We define

(11) $me' := [\![Ms]\!]_i^{view(c)}(mes(i), envNext_i^{view(c),mes(i)})$

From (2), (11) and the definition of *envSeq*, it suffices to show

(a.2.a) $rspecifies_{i+1}(se', me', c, [\![Ms]\!]_{\mathrm{I}})$

From (3), (4), (10), (11), and the definition of *rspecifies*, we know (a.2.a). □

Based on the previously defined predicates and functions, Figure 6.45 introduces new judgements for the recursion language. The first two judgements are used to derive the specification environment se' constructed by a method declaration M respectively a sequence of method declarations Ms in a given specification environment me. The next two judgements state the correctness of a method declaration M respectively a sequence of method declarations Ms in a specification environment se where Is are the names of all user-defined methods. The last judgement states the correctness of a program with method declarations Ms, program specification S and program body C.

Figure 6.46 updates the previously stated judgements for the well-definedness, correctness, and termination of commands which now all take as additional arguments the names Is of the user-defined methods. The last judgement for termination also includes a parameter I which refers to a mathematical variable $\$I$ that describes by a natural number the value of the termination term T when the current method was invoked. Since this value represents a bound for the number of recursive invocations, the soundness claim is only valid for a method environment $me \in MethodEnvs_i$ where i is at least as large as $\$I$.

The rules for these judgements are given in Figures 6.47, 6.48, and 6.49. Among these, the second rule in Figure 6.48 is most important: it establishes the termination of a (possibly recursive) call of a user-defined method by ensuring that the value of the termination term T at the point of the invocation is less than $\$I$. A

Recursion Language: Judgements

$$se \vdash M : se' \Leftrightarrow se' = [\![M]\!]_{\mathrm{S}}(se)$$

$$se \vdash Ms : se' \Leftrightarrow se' = [\![Ms]\!]_{\mathrm{S}}(se)$$

$$se, Is \vdash M \Leftrightarrow rspecifies(se, M, Is)$$

$$se, Is \vdash Ms \Leftrightarrow rspecifies(se, Ms, Is)$$

$$\begin{array}{l} se \vdash Ms\ S\ \{C\} \Leftrightarrow \\ \quad rspecifies(se, Ms, [\![Ms]\!]_{\mathrm{I}}) \wedge \\ \quad \forall c \in Context, me \in MethodEnv : \\ \qquad DifferentVariables(c) \wedge specifies(se, me, c) \Rightarrow \\ \qquad \mathrm{LET}\ mes = envSeq^{view(c), me, Ms}\ \mathrm{IN} \\ \qquad \forall s, s' \in State, e \in Environment : executes(control(s)) \Rightarrow \\ \qquad\quad (\forall i \in \mathbb{N} : [\![C]\!]_i^{c, mes(i)}(s) \Rightarrow correctness([\![S]\!]_{\mathrm{S}}, c, e, s, s')) \wedge \\ \qquad\quad ([\![assumption(S)]\!]^{c,c}(e)(s,s) \Rightarrow \exists i \in \mathbb{N} : [\![C]\!]_{\mathrm{T}_i}^{c, mes(i)}(s)) \end{array}$$

Figure 6.45: Judgements for the Recursion Language

Commands: Judgements

$se, \{V_1, \ldots, V_o\}, Is \vdash C \checkmark F \Leftrightarrow$
F has no free (mathematical or state) variables $\wedge$
F does not depend on the poststate $\Rightarrow$
$\forall c, c' \in Context, i \in \mathbb{N}, me \in MethodEnv_i :$
$rspecifies_i(se, me, c, Is) \wedge pushes(c, c', \{V_1, \ldots, V_o\}) \Rightarrow$
$\forall s, s' \in State : executes(control(s)) \wedge [\![F]\!]^{c,c'}(s) \Rightarrow$
$([\![C]\!]_i^{c',me}(s, s') \Leftrightarrow [\![C]\!]_{\perp_i}^{c',me}(s, s'))$

$se, \{V_1, \ldots, V_o\}, Is \vdash C : F \Leftrightarrow$
F has no free (mathematical or state) variables $\wedge$
$\forall c, c' \in Context, i \in \mathbb{N}, me \in MethodEnv_i :$
$rspecifies_i(se, me, c, Is) \wedge pushes(c, c', \{V_1, \ldots, V_o\}) \Rightarrow$
$\forall s, s' \in State, e \in Environment :$
$executes(control(s)) \wedge [\![C]\!]_i^{c',me}(s, s') \Rightarrow$
$[\![F]\!]^{c,c'}(e)(s, s')$

$se, \{V_1, \ldots, V_o\}, Is \vdash C \downarrow^I F \Leftrightarrow$
F has as its only free variable $\$I \wedge$
F does not depend on the poststate $\wedge$
F makes $\$I$ a natural number $\Rightarrow$
$\forall c, c' \in Context, i \in \mathbb{N}, me \in MethodEnvs_i :$
$rspecifies_i(se, me, c, Is) \wedge pushes(c, c', \{V_1, \ldots, V_o\}) \Rightarrow$
$\forall s \in State, e \in Environment : executes(control(s)) \Rightarrow$
$[\![F]\!]^{c,c'}(e)(s, s) \wedge i \geq e(I) \Rightarrow [\![C]\!]_{\mathrm{T}_i}^{c',me}(s)$

Figure 6.46: Judgements for Commands of the Recursion Language

Recursion Language: Rules

$$\frac{S = \texttt{writesonly}\ I_1,\ldots,I_n\ \texttt{throwsonly}\ K_1,\ldots,K_m \quad \texttt{assumes}\ F_C\ \texttt{implements}\ F_R\ \texttt{decreases}\ T}{se \vdash \texttt{method}\, I_m\,(J_1,\ldots,J_p)\ S\ \{C\} : se[I_m \mapsto \langle (J_1,\ldots,J_p), \langle (I_1,\ldots,I_n),(K_1,\ldots,K_m),F_C,F_R,T\rangle\rangle]}$$

$$se \vdash \sqcup : se$$

$$\frac{se \vdash Ms : se' \quad se' \vdash M : se''}{se \vdash Ms\ M : se''}$$

Figure 6.47: Rules for the Recursion Language (1/3)

similar proof obligation is also included in the rule for the correctness of programs in Figure 6.49; however, here it is only used for uniformity of the calculus (we could well do without this obligation and just use the correctness rules for commands given in the previous section).

We now turn our attention to the soundness of these rules. The soundness of the rules given in Figure 6.47 for building up specification environments from (sequences of) method declarations is obvious from the rules and the definition of $[\![\sqcup]\!]_S$. The soundness of the remaining rules is shown in the following subsections.

6.7.1 Method Calls (Pre-Defined Methods)

$$\frac{\begin{array}{l} I_m \notin Is \\ se(I_m) = \langle (J_1,\ldots,J_p), \langle (I_1,\ldots,I_n),(K_1,\ldots,K_m),F_C,F_R,T\rangle\rangle \\ E_1 \simeq T_1,\ldots,E_p \simeq T_p \\ L_1,\ldots,L_p \text{ is a renaming of } J_1,\ldots,J_p \\ \$L_1,\ldots,\$L_p \text{ do not occur in } F_C \\ \forall s \in State : \\ \quad [\![\texttt{now.executes AND (EXISTS } \$I : F) \texttt{ =>} \\ \qquad \texttt{FORALL } \$L_1,\ldots,\$L_p : \\ \qquad\quad \$L_1{=}T_1 \texttt{ AND } \ldots \texttt{ AND } \$L_p{=}T_p \texttt{ =>} \\ \qquad\quad F_C[\$L_1/J_1,\ldots,\$L_p/J_p,\$L_1/J_1{}',\ldots,\$L_p/J_p{}'] \\ \qquad\qquad [?V_1/V_1,\ldots,?V_o/V_o,?V_1{}'/V_1{}',\ldots,?V_o{}'/V_o{}']]\!](s) \end{array}}{se,\{V_1,\ldots,V_o\},Is \vdash I_r = I_m\,(E_1,\ldots,E_p) \downarrow^I F}$$

Recursion Language: Rules (Contd)

$$\frac{\begin{array}{l} I_m \notin Is \\ se(I_m) = \langle (J_1,\ldots,J_p), \langle (I_1,\ldots,I_n), (K_1,\ldots,K_m), F_C, F_R, T\rangle\rangle \\ E_1 \simeq T_1, \ldots, E_p \simeq T_p \\ L_1,\ldots,L_p \text{ is a renaming of } J_1,\ldots,J_p \\ \$L_1,\ldots,\$L_p \text{ do not occur in } F_C \\ \forall s \in State: \\ \quad [\![\texttt{now.executes AND (EXISTS } \$I : F) \texttt{ =>} \\ \quad\quad \texttt{FORALL } \$L_1,\ldots,\$L_p : \\ \quad\quad\quad \$L_1{=}T_1 \texttt{ AND } \ldots \texttt{ AND } \$L_p{=}T_p \texttt{ =>} \\ \quad\quad\quad F_C[\$L_1/J_1,\ldots,\$L_p/J_p, \$L_1/J_1\texttt{'},\ldots,\$L_p/J_p\texttt{'}] \\ \quad\quad\quad\quad [?V_1/V_1,\ldots,?V_o/V_o, ?V_1\texttt{'}/V_1\texttt{'},\ldots,?V_o\texttt{'}/V_o\texttt{'}]]\!](s) \end{array}}{se, \{V_1,\ldots,V_o\}, Is \vdash I_r = I_m\,(E_1,\ldots,E_p) \downarrow^I F}$$

$$\frac{\begin{array}{l} I_m \in Is \\ se(I_m) = \langle (J_1,\ldots,J_p), \langle (I_1,\ldots,I_n), (K_1,\ldots,K_m), F_C, F_R, T\rangle\rangle \\ E_1 \simeq T_1, \ldots, E_p \simeq T_p \\ L_1,\ldots,L_p \text{ is a renaming of } J_1,\ldots,J_p \\ \$L_1,\ldots,\$L_p \text{ do not occur in } F_C \\ I \notin \{L_1,\ldots,L_p\} \\ \forall s \in State, e \in Environment, c, c' \in Context: \\ \forall m \in \mathbb{N}, v_1,\ldots,v_p \in Value: \\ \quad \text{LET } e_0 = e[I \mapsto m, L_1 \mapsto v_1, \ldots, L_p \mapsto v_p] \text{ IN} \\ \quad [\![\texttt{now.executes AND } F \texttt{ AND} \\ \quad\quad \$L_1{=}T_1 \texttt{ AND } \ldots \texttt{ AND } \$L_p{=}T_p]\!]^{c,c'}(e_0)(s,s) \;\Rightarrow \\ \quad ([\![F_C[\$L_1/J_1,\ldots,\$L_p/J_p, \$L_1/J_1\texttt{'},\ldots,\$L_p/J_p\texttt{'}] \\ \quad\quad [?V_1/V_1,\ldots,?V_o/V_o, ?V_1\texttt{'}/V_1\texttt{'},\ldots,?V_o\texttt{'}/V_o\texttt{'}]]\!]^{c,c'} \\ \quad\quad (e_0)(s,s) \wedge \\ \quad \text{LET } m' = \\ \quad\quad [\![T[\$L_1/J_1,\ldots,\$L_p/J_p, \$L_1/J_1\texttt{'},\ldots,\$L_p/J_p\texttt{'}] \\ \quad\quad\quad [?V_1/V_1,\ldots,?V_o/V_o, ?V_1\texttt{'}/V_1\texttt{'},\ldots,?V_o\texttt{'}/V_o\texttt{'}]]\!]^{c,c'} \\ \quad\quad\quad (e_0)(s,s) \\ \quad \text{IN } m' \in \mathbb{N} \wedge m > m') \end{array}}{se, \{V_1,\ldots,V_o\}, Is \vdash I_r = I_m\,(E_1,\ldots,E_p) \downarrow^I F}$$

Figure 6.48: Rules for the Recursion Language (2/3)

Recursion Language: Rules (Contd)

$\ldots$
(all hypotheses except the last one from the rule in Figure 6.34)
$I_m \in Is$
T has no free (mathematical or state) variables
T does not depend on the poststate
F_C makes T a natural number
$se, \{J_1, \ldots, J_p\}, Is \vdash C \downarrow^I F_C \texttt{ AND \$}I = T$
$se, Is \vdash \texttt{method}\ I_m\ (J_1, \ldots, J_p)\ S\ \{C\}$

$se, Is \vdash \llcorner$

$se, Is \vdash Ms$
$se, Is \vdash M$
$se, Is \vdash Ms\ M$

$se \vdash Ms : se'$
$Is = [\![Ms]\!]_{\mathrm{I}}$
$se', Is \vdash Ms$
$S = \texttt{writesonly}\ I_1, \ldots, I_n\ \texttt{throwsonly}\ K_1, \ldots, K_m$
$\quad \texttt{assumes}\ F_C\ \texttt{implements}\ F_R\ \texttt{decreases}\ T$
F_C, F_R and T have no free (mathematical or state) variables
F_C and T do not depend on the poststate
F_C makes T a natural number
$se', \emptyset, Is \vdash C : [F]^{F_c,F_b,F_r,\{L_1,\ldots,L_r\}}_{M_1,\ldots,M_o}$
$[F]^{F_c,F_b,F_r,\{L_1,\ldots,L_r\}}_{M_1,\ldots,M_o}\ \S\ ^{K_1,\ldots,K_m}_{I_1,\ldots,I_n}$
$se', \emptyset, Is \vdash C \checkmark F_C$
$\forall s, s' \in State, e \in Environment, c_0, c_1 \in Context :$
$\quad [\![(\texttt{now.executes AND}\ F) \texttt{ => } (F_C \texttt{ => } F_R)]\!]^{c_0,c_1}(e)(s,s')$
$se', \emptyset, Is \vdash C \downarrow^I F_C \texttt{ AND \$}I = T$
$se \vdash Ms\ S\ \{C\}$

Figure 6.49: Rules for the Recursion Language (3/3)

Soundness Proof Take arbitrary $c, c' \in \mathit{Context}, i \in \mathbb{N}, me \in \mathit{MethodEnvs}_i$ and $s \in \mathit{State}, e \in \mathit{Environment} : \mathit{executes}(\mathit{control}(s))$ and assume

(1) F has as its only free variable $\$I$

(2) F does not depend on the poststate

(3) F makes $\$I$ a natural number

(4) $\mathit{rspecifies}_i(se, me, c, Is)$

(5) $\mathit{pushes}(c, c', \{V_1, \ldots, V_o\})$

(6) $\mathit{executes}(\mathit{control}(s))$

(7) $[\![F]\!]^{c,c'}(e)(s,s)$

(8) $i \geq e(I)$

We show

(a) $[\![I_r = I_m\ (E_1, \ldots, E_p)\ ]\!]^{c',me}_{\mathrm{T}_i}(s)$

From the first hypothesis, we know

(9) $I_m \notin Is$

From (3), (8), and the definition of "makes a natural number", we know

(10) $i > 0$

From (10), we have some $j \in \mathbb{N}$ with

(11) $i = j + 1$

From (11) and the definition of $[\![\_]\!]_{\mathrm{T}_\_}$, it suffices to take arbitrary $v \in \mathit{View}, me' \in \mathit{RecMethodEnv}_j, b \in \mathit{RecBehavior}_j, r \in \mathit{StateRelation}, t \in \mathit{StateCondition}$ with

(12) $\langle v, me', b \rangle = me(I_m)$

(13) $\langle r, t \rangle = b^{call(v,c),me'}([\![E_1]\!]^{c'}(s), \ldots, [\![E_p]\!]^{c'}(s))$

and show

(b) $t(s)$

From (4), (11), (12), (13), and the definition of $\mathit{rspecifies}_{j+1}$, we know

(14) $v = \mathit{view}(c)$

(15) $rspecifies_j(se(I_m), b, c, me', Is, I_m)$

We define

(16) $c'' = push(call(v,c), J_1, \ldots, J_p)$

(17) $s_1 := writes(s, J_1, [\![E_1]\!]^{c'}(s), \ldots, J_p, [\![E_p]\!]^{c'}(s))^{c'}$

(18) $m := [\![T]\!]^{c,c''}(e)(s_1, s_1)$

In close analogy to the proof given in Section 6.5.4, we can show

(19) $[\![F_C]\!]^{c,c''}(e)(s_1, s_1)$

From (9), we know

(20) $I_m \in I_s \Rightarrow m \in \mathbb{N} \wedge m < i$

From (15), (16), (17), (18), (19), (20), and ultimately the definition of $rspecifies_j$, we know (b). □

6.7.2 Method Calls (User-Defined Methods)

$$
\begin{array}{l}
I_m \in Is \\
se(I_m) = \langle (J_1, \ldots, J_p), \langle (I_1, \ldots, I_n), (K_1, \ldots, K_m), F_C, F_R, T \rangle \rangle \\
E_1 \simeq T_1, \ldots, E_p \simeq T_p \\
L_1, \ldots, L_p \text{ is a renaming of } J_1, \ldots, J_p \\
\$L_1, \ldots, \$L_p \text{ do not occur in } F_C \\
I \notin \{L_1, \ldots, L_p\} \\
\forall s \in State, e \in Environment, c, c' \in Context : \\
\forall m \in \mathbb{N}, v_1, \ldots, v_p \in Value : \\
\quad \text{LET } e_0 = e[I \mapsto m, L_1 \mapsto v_1, \ldots, L_p \mapsto v_p] \text{ IN} \\
\quad [\![\texttt{now.executes AND } F \texttt{ AND} \\
\qquad \$L_1{=}T_1 \texttt{ AND } \ldots \texttt{ AND } \$L_p{=}T_p]\!]^{c,c'}(e_0)(s,s) \Rightarrow \\
\quad ([\![F_C[\$L_1/J_1, \ldots, \$L_p/J_p, \$L_1/J_1\texttt{'}, \ldots, \$L_p/J_p\texttt{'}] \\
\qquad [?V_1/V_1, \ldots, ?V_o/V_o, ?V_1\texttt{'}/V_1\texttt{'}, \ldots, ?V_o\texttt{'}/V_o\texttt{'}]]\!]^{c,c'} \\
\qquad (e_0)(s,s) \wedge \\
\quad \text{LET } m' = \\
\qquad [\![T[\$L_1/J_1, \ldots, \$L_p/J_p, \$L_1/J_1\texttt{'}, \ldots, \$L_p/J_p\texttt{'}] \\
\qquad\quad [?V_1/V_1, \ldots, ?V_o/V_o, ?V_1\texttt{'}/V_1\texttt{'}, \ldots, ?V_o\texttt{'}/V_o\texttt{'}]]\!]^{c,c'} \\
\qquad\quad (e_0)(s,s) \\
\quad \text{IN } m' \in \mathbb{N} \wedge m > m') \\
\hline
se, \{V_1, \ldots, V_o\}, Is \vdash I_r = I_m\,(E_1, \ldots, E_p) \downarrow^I F
\end{array}
$$

Soundness Proof Take arbitrary $c, c' \in \mathit{Context}, i \in \mathbb{N}, \mathit{me} \in \mathit{MethodEnvs}_i$ and $s \in \mathit{State}, e \in \mathit{Environment} : \mathit{executes}(\mathit{control}(s))$ and assume

(1) F has as its only free variable $\$I$

(2) F does not depend on the poststate

(3) F makes $\$I$ a natural number

(4) $\mathit{rspecifies}_i(se, me, c, Is)$

(5) $\mathit{pushes}(c, c', \{V_1, \ldots, V_o\})$

(6) $\mathit{executes}(\mathit{control}(s))$

(7) $[\![F]\!]^{c,c'}(e)(s,s)$

(8) $i \geq e(I)$

We show

(a) $[\![I_r = I_m\ (E_1, \ldots, E_p)\ ]\!]^{c',me}_{\mathrm{T}_i}(s)$

From the first hypothesis, we know

(9) $I_m \notin Is$

From (3), (8), and the definition of "makes a natural number", we know

(10) $i > 0$

From (10), we have some $j \in \mathbb{N}$ with

(11) $i = j + 1$

From (11) and the definition of $[\![\_]\!]_{\mathrm{T}_\_}$, it suffices to take arbitrary $v \in \mathit{View}, \mathit{me}' \in \mathit{RecMethodEnv}_j, b \in \mathit{RecBehavior}_j, r \in \mathit{StateRelation}, t \in \mathit{StateCondition}$ with

(12) $\langle v, \mathit{me}', b \rangle = \mathit{me}(I_m)$

(13) $\langle r, t \rangle = b^{\mathit{call}(v,c),\mathit{me}'}([\![E_1]\!]^c(s), \ldots, [\![E_p]\!]^c(s))$

and show

(b) $t(s)$

From (4), (11), (12), (13), and the definition of $\mathit{rspecifies}_{j+1}$, we know

(14) $v = \mathit{view}(c)$

(15) $rspecifies_j(se(I_m), b, c, me', Is, I_m)$

We define

(16) $c'' = push(call(v, c), J_1, \ldots, J_p)$

(17) $s_1 := writes(s, J_1, [\![E_1]\!]^{c'}(s), \ldots, J_p, [\![E_p]\!]^{c'}(s))^{c'}$

In close analogy to the proof given in Section 6.5.4, we can show

(18) $[\![F_C]\!]^{c,c''}(e)(s_1, s_1)$

We define

(19) $m := [\![T]\!]^{c,c''}(e)(s_1, s_1)$

From (15), (16), (17), (18), and the definition of $rspecifies_j$, it suffices to show

(c.1) $m \in \mathbb{N}$

(c.2) $m < i$

From (15) and the definition of $rspecifies_j$, we know

(20) F_C `makes` T `a natural number`

From (18), (19), (20), and then the definition of "makes a natural number", we know (c.1).

We define

(21) $v_1 := [\![E_1]\!]^{c'}(s), \ldots, v_p := [\![E_p]\!]^{c'}(s)$

(22) $e_0 := e[I \mapsto e(I), L_1 \mapsto v_1, \ldots, L_p \mapsto v_p]$

(23) $m' := [\![T[\$L_1/J_1, \ldots, \$L_p/J_p, \$L_1/J_1\,', \ldots, \$L_p/J_p\,'] [?V_1/V_1, \ldots, ?V_o/V_o, ?V_1\,'/V_1\,', \ldots, ?V_o\,'/V_o\,']]\!]^{c,c'}(e_0)(s, s)$

From (21), (22), (23), and hypothesis 6, we know

(24) $[\![$`now.executes AND` F `AND` $\$L_1$=$T_1$ `AND ... AND` $\$L_p$=$T_p$ $]\!]^{c,c'}(e_0)(s, s) \Rightarrow m' \in \mathbb{N} \wedge e(I) > m'$

To show (c.2), from (8), (24), and the definition of $[\![\,\_\,]\!]$, it suffices to show

(c.2.1) $executes(control(s))$

(c.2.2) $[\![F]\!]^{c,c'}(e_0)(s,s)$

(c.2.3) $v_1 = [\![T_1]\!]^{c,c'}(e_0)(s,s) \wedge \ldots \wedge v_p = [\![T_p]\!]^{c,c'}(e_0)(s,s)$

(c.2.4) $m = m'$

From (6), we know (c.2.1).

From (22), we know

(25) $e(I) = e_0(I)$

From (1), (7), and (25), we know (c.2.2).

From hypothesis 3, (5), and the definitions of *pushes* and $\simeq$, we know (c.2.3).

From (5) and the definition of *pushes*, we know

(25a) $c = c'$ EXCEPT $V_1, \ldots, V_o$

From (23), (25a), (PMGF), and the definition of AT, we know

(26) $m' = [\![T[\$L_1/J_1, \ldots, \$L_p/J_p, \$L_1/J_1\text{'}, \ldots, \$L_p/J_p\text{'}]]\!]^{c,c}(e_0)(s,s)$

From (26) and the definition of *call*, we know

(27) $m' =$
$[\![T[\$L_1/J_1, \ldots, \$L_p/J_p, \$L_1/J_1\text{'}, \ldots, \$L_p/J_p\text{'}]]\!]^{c,call(v,c)}(e_0)(s,s)$

From (16) and the definition of *push*, we know

(28) $call(v,c) = c''$ EXCEPT $J_1, \ldots, J_o$

From (MPVF0') and (MPVF1'), we know

(29) $J_1, \ldots, J_p$ do not occur in
$T[\$L_1/J_1, \ldots, \$L_p/J_p, \$L_1/J_1\text{'}, \ldots, \$L_p/J_p\text{'}]$

(30) $J_1\text{'}, \ldots, J_p\text{'}$ do not occur in
$T[\$L_1/J_1, \ldots, \$L_p/J_p, \$L_1/J_1\text{'}, \ldots, \$L_p/J_p\text{'}]$

From (27), (28), (29), (30), and the definitions of $[\![\_]\!]$ and EXCEPT, we know

(31) $m' = [\![T[\$L_1/J_1, \ldots, \$L_p/J_p, \$L_1/J_1\text{'}, \ldots, \$L_p/J_p\text{'}]]\!]^{c,c''}(e_0)(s,s)$

From (4), hypothesis 2, and the definition of $rspecifies_i$, we know

(32) $\$J_1, \ldots, \$J_p$ do not occur in T

From hypothesis 4, we know

(33) $L_1, \ldots, L_p$ is a renaming of $J_1, \ldots, J_p$

From (17) and (WSE), we know

(34) $s = s_1$ EXCEPT $J_1, \ldots, J_p$

From (17) and (CWE), we know

(35) $control(s) = control(s_1)$

From (17) and (RWE), we know

(36) $read(s_1, J_1)^{c'} = [\![E_1]\!]^{c'}(s) \wedge \ldots \wedge read(s_1, J_p)^{c'} = [\![E_p]\!]^{c'}(s)$

From (21), (22), and (36), we know

(37) $e_0 = e[L_1 \mapsto [\![E_1]\!]^{c'}(s), \ldots, L_p \mapsto [\![E_p]\!]^{c'}(s)]$

From (31), (33), (34), (36), (37), (PMVT1"), and (PMVT"), we know

(38) $m' = [\![T]\!]^{c,c''}(e)(s_1, s_1)$

From (19) and (38), we know (c.2.4). □

6.7.3 Method Declarations

$$\frac{\begin{array}{l}\ldots\\ \text{(all hypotheses except the last one from the rule in Figure 6.34)}\\ I_m \in Is\\ T \text{ has no free (mathematical or state) variables}\\ T \text{ does not depend on the poststate}\\ F_C \text{ makes } T \text{ a natural number}\\ se, \{J_1, \ldots, J_p\}, Is \vdash C \downarrow^I F_C \texttt{ AND } \$I = T\end{array}}{se, Is \vdash \texttt{method } I_m \texttt{(}J_1, \ldots, J_p\texttt{) } S \texttt{ \{}C\texttt{\}}}$$

Soundness Proof We define

(1) $M := \texttt{method}\ I_m\ (J_1, \ldots, J_p)\ S\ \{C\}$

and prove

(a) $\mathit{rspecifies}(se, M, Is)$

From the definition of *rspecifies*, it suffices to take arbitrary $c \in \mathit{Context}, i \in \mathbb{N}, me \in \mathit{RecMethodEnvs}_i$ and $me', me'' \in \mathit{RecMethodEnvs}_{i+1}$ with

(2) $\mathit{DifferentVariables}(c)$
(3) $\mathit{rspecifies}_i(se, me, c, Is)$
(4) $\mathit{rspecifies}_{i+1}(se, me', c, Is)$
(5) $me'' = [\![M]\!]_i^{\mathit{view}(c)}(me, me')$

and show

(b) $\mathit{rspecifies}_{i+1}(se, me'', c, Is)$

To show (b), from the definition of $\mathit{rspecifies}_{i+1}$, it suffices to take arbitrary $I \in \mathrm{Identifier}, v \in \mathit{View}, me''' \in \mathit{RecMethodEnvs}_i, b \in \mathit{RecBehavior}_{i+1}$ such that

(6) $\langle v, me''', b \rangle = me''(I)$

and show

(b.1) $v = \mathit{view}(c)$
(b.2) $\mathit{rspecifies}_i(se(I), b, c, me''', Is, I)$

We proceed by case distinction.

In the first case, we assume

(7) $I \neq I_m$

From (1), (5), (7), and the definition of $[\![M]\!]$, we know

(8) $me''(I) = me'(I)$

From (4), (8), and the definition of $\mathit{rspecifies}_{i+1}$, we know (b.1) and (b.2).

In the second case, we assume

(9) $I = I_m$

From (1), (5), (6), (9), and the definition of $[\![M]\!]$, we know (b.1) and

(10) $me''' = me$

To show (b.2), from (10) and the definition of *rspecifies*, it suffices to take arbitrary $v_1, \ldots, v_p \in \mathit{Value}, s, s_1, s' \in \mathit{State}$, $e \in \mathit{Environment}$, $c', c'' \in \mathit{Context}, r \in \mathit{StateRelation}, t \in \mathit{StateCondition}, m \in \mathit{Value}$ such that

(11) $\mathit{executes}(\mathit{control}(s))$

(12) $\mathit{DifferentVariables}(c')$

(13) c' EQUALS c

(14) $\mathit{space}(c') \subseteq \mathit{space}(c)$

(15) $\langle r, t \rangle = b^{c', me}(v_1, \ldots, v_p)$

(16) $c'' = \mathit{push}(c', J_1, \ldots, J_p)$

(17) $s_1 = \mathit{writes}(s, J_1, v_1, \ldots, J_p, v_p)^{c''}$

(18) $r(s, s')$

(18a) $m = [\![T]\!]^{c, c''}(e)(s_1, s_1)$

and show

(b.2.1) $\{J_1, \ldots, J_p\} \cap \{I_1, \ldots, I_n\} = \emptyset$

(b.2.2) F_C, F_R and T have no free (mathematical or state) variables

(b.2.3) F_C and$T$ do not depend on the poststate

(b.2.4) $[\![F_C \Rightarrow F_R]\!]^{c, c''}(e)(s_1, s')$

(b.2.5) $s = s'$ EXCEPT $\mathit{range}(c') \cup \mathrm{range}(\mathit{view}(c))$

(b.2.6) $s = s'$ EXCEPT^{c,c} $I_1, \ldots, I_n$

(b.2.7) $\mathit{executes}(\mathit{control}(s')) \vee \mathit{throws}(\mathit{control}(s'))$

(b.2.8) $\mathit{throws}(\mathit{control}(s')) \Rightarrow \mathit{key}(\mathit{control}(s')) \in \{K_1, \ldots, K_m\}$

(b.2.9) $[\![F_C]\!]^{c, c''}(e)(s_1, s_1) \wedge (I \in Is \Rightarrow m \in \mathbb{N} \wedge m < i) \Rightarrow t(s)$

We can show (b.2.1),. . . ,(b.2.8) as illustrated in Section 6.6.1.

To show (b.2.9), we assume

(19) $[\![F_C]\!]^{c, c''}(e)(s_1, s_1)$

(20) $I \in Is \Rightarrow m \in \mathbb{N} \wedge m < i$

and show

(b.2.9.a) $t(s)$

From (5), (6), (9), (15), (16), (17), and the definition of $[\![\_]\!]$, it suffices to show

(b.2.9.b) $[\![C]\!]^{c'',me}_{\mathrm{T}_i}(s_1)$

From the first new hypothesis, we know

(21) $I \in Is$

From (18a), (20), and (21), we know

(22) $[\![T]\!]^{c,c''}(e)(s_1,s_1) < i$

From hypothesis 3 and the second new hypothesis, we know

(23) F_C `AND` $\$I = T$ has as its only free variable $\$I$

From hypothesis 4 and the third new hypothesis, we know

(24) F_C `AND` $\$I = T$ does not depend on the poststate

From the fourth new hypothesis and the definitions of "makes a natural number", we know

(25) F_C `AND` $\$I = T$ makes $\$I$ a natural number

From (23), (24), (25), and the fifth new hypothesis, we know

(26)
$$\begin{array}{l}\forall c, c' \in Context, i \in \mathbb{N}, me \in MethodEnvs_i : \\ \quad rspecifies_i(se, me, c, Is) \wedge pushes(c, c', \{V_1, \ldots, V_o\}) \Rightarrow \\ \quad\quad \forall s \in State, e \in Environment : executes(control(s)) \Rightarrow \\ \quad\quad\quad [\![F_C \texttt{ AND } \$I = T]\!]^{c,c'}(e)(s,s) \wedge i \geq e(I) \Rightarrow [\![C]\!]^{c',me}_{\mathrm{T}_i}(s)\end{array}$$

From (12), (16) and (COP), we know

(27) $pushes(c', c'', \{J_1, \ldots, J_p\})$

From (2), (13), (14), (27), (NEQ), (AVE), (TRE), and the definition of *pushes*, we know

(28) $pushes(c, c'', \{J_1, \ldots, J_p\})$

From (11), (17), and (CW), we know

(29) $executes(control(s_1))$

We define

(30) $e_0 := e[I \mapsto [\![T]\!]^{c,c''}(e)(s_1, s_1)]$

From (3), (26), (28), and (29), we know

(31) $[\![F_C \texttt{ AND } \$I = T]\!]^{c,c''}(e_0)(s_1, s_1) \wedge i \geq e_0(I) \Rightarrow [\![C]\!]^{c'',me}_{T_i}(s_1)$

From (19), the third assumption, and (MVF'), we know

(32) $[\![F_C]\!]^{c,c''}(e_0)(s_1, s_1)$

From the last but fourth assumption, we know

(33) $[\![T]\!]^{c,c''}(e)(s_1, s_1) = [\![T]\!]^{c,c''}(e_0)(s_1, s_1)$

From (30), (33), and the definition of $[\![\_]\!]$, we know

(34) $[\![\$I = T]\!]^{c,c''}(e_0)(s_1, s_1)$

From (22) and (30), we know

(35) $e_0(I) < i$

From (31), (32), (34), and (35), we know (b.2.9.b). □

6.7.4 Empty Method Sequences

$se, Is \vdash \_$

Soundness Proof We have to show

(a) $\mathit{rspecifies}(se, \_, Is)$

From the definition of *rspecifies*, it suffices to take arbitrary $c \in \mathit{Context}, i \in \mathbb{N}, me \in \mathit{RecMethodEnvs}_i$ and $me' \in \mathit{RecMethodEnvs}_{i+1}$ with

(1) $\mathit{DifferentVariables}(c)$
(2) $\mathit{rspecifies}_i(se, me, c, Is)$
(3) $me' = [\![\_]\!]_i^{\mathit{view}(c)}(me, \mathit{envNext}_i^{\mathit{view}(c), me})$

and show

(b) $\mathit{rspecifies}_{i+1}(se, me', c, Is)$

To show (b), from the definition of $\mathit{rspecifies}_{i+1}$, it suffices to take arbitrary $I \in \mathrm{Identifier}, v \in \mathit{View}, me'' \in \mathit{RecMethodEnvs}_i, b \in \mathit{RecBehavior}_{i+1}$ such that

(4) $\langle v, me'', b \rangle = me'(I)$

and show

(b.1) $v = \mathit{view}(c)$
(b.2) $\mathit{rspecifies}_i(se(I), b, c, me'', Is, I)$

From (3) and the definition of $[\![\_]\!]_i$, we know

(5) $me' = \mathit{envNext}_i^{\mathit{view}(c), me}$

From (4), (5), and the definition of $\mathit{envNext}_i$, we know

(5a) $me'' = me$

From (4), (5), and the definition of $\mathit{envNext}_i$, we know (b.1).

We proceed by case distinction.

In the first case, we assume

(6) $i = 0$

We take $v_0 \in \mathit{View}, b_0 \in \mathit{Behavior}$ such that

(7) $\langle v_0, b_0 \rangle = me(I)$

From (4), (5), (5a), (6), (7), and the definition of $envNext_0$, we know

(8) $\forall v_1,\ldots,v_p \in Value : b^{c,me}(v_1,\ldots,v_p) = b_0^c(v_1,\ldots,v_p)$

We proceed with two subcases. In the first subcase, we assume

(9) $I \in Is$

From (2), (6), (7), and (9), we know

(10) $wellformed(se(I)$

(11) $\forall c \in Context, v_1,\ldots,v_p \in Value : b_0^c(v_1,\ldots,v_p) = \langle \emptyset, \emptyset \rangle$

From (5a), (6), (8), (10), (11), and the definition of $rspecifies_0$, we can show (b.2).

In the second subcase, we assume

(12) $I \notin Is$

From (2), (6), (7), and (12), we know

(13) $specifies(se(I), b_0, c)$

From (5a), (6), (8), (13), and the definitions of *specifies* and $rspecifies_0$, we can show (b.2).

In the second case, we assume for some $j \in \mathbb{N}$

(14) $i = j+1$

We take $v_0 \in View, me_0 \in RecMethodEnv_j, b_0 \in RecBehavior_{j+1}$ such that

(15) $\langle v_0, me_0, b_0 \rangle = me(I)$

From, (4), (5), (5a), (14), (15), and the definition of $envNext_{j+1}$, we know

(16) $\forall v_1,\ldots,v_p \in Value : b^{c,me}(v_1,\ldots,v_p) = b_0^{c,me_0}(v_1,\ldots,v_p)$

From (2), (14), and (15), we know

(17) $rspecifies_{j+1}(se(I), b_0, c, me_0, Is, I)$

From (5a), (14), (16), and (17), we can show (b.2). □

6.7.5 Non-Empty Method Sequences

$$\frac{se, Is \vdash Ms \quad se, Is \vdash M}{se, Is \vdash Ms\ M}$$

Soundness Proof We have to show

(a) $rspecifies(se, Ms\ M, Is)$

From the definition of *rspecifies*, it suffices to take arbitrary $c \in Context, i \in \mathbb{N}, me \in RecMethodEnvs_i$ and $me' \in RecMethodEnvs_{i+1}$ with

(1) $DifferentVariables(c)$

(2) $rspecifies_i(se, me, c, Is)$

(3) $me' = [\![Ms\ M]\!]_i^{view(c)}(me, envNext_i^{view(c), me})$

and show

(b) $rspecifies_{i+1}(se, me', c, Is)$

We define

(4) $me_0 := [\![Ms]\!]_i^{view(c)}(me, envNext_i^{view(c), me})$

From (3), (4), and the definition of $[\![\_]\!]$, we know

(5) $me' = [\![M]\!]_i^{view(c)}(me, me_0)$

From hypothesis 1, (1), (2), (4), and the definition of *rspecifies*, we know

(6) $rspecifies_{i+1}(se, me_0, c, Is)$

From hypothesis 2, (1), (2), (5), (6), and the definition of *rspecifies*, we know (b). □

6.7.6 Verification of Programs

$$\frac{\begin{array}{l} se \vdash Ms : se' \\ Is = [\![Ms]\!]_{\mathrm{I}} \\ se', Is \vdash Ms \\ S = \texttt{writesonly}\ I_1,\ldots,I_n\ \texttt{throwsonly}\ K_1,\ldots,K_m \\ \quad \texttt{assumes}\ F_C\ \texttt{implements}\ F_R\ \texttt{decreases}\ T \\ F_C, F_R \text{ and } T \text{ have no free (mathematical or state) variables} \\ F_C \text{ and } T \text{ do not depend on the poststate} \\ F_C \text{ makes } T \text{ a natural number} \\ se', \emptyset, Is \vdash C : [F]^{F_c,F_b,F_r,\{L_1,\ldots,L_r\}}_{M_1,\ldots,M_o} \\ [F]^{F_c,F_b,F_r,\{L_1,\ldots,L_r\}}_{M_1,\ldots,M_o} \ \S\ ^{K_1,\ldots,K_m}_{I_1,\ldots,I_n} \\ se', \emptyset, Is \vdash C \checkmark F_C \\ \forall s, s' \in State, e \in Environment, c_0, c_1 \in Context : \\ \quad [\![(\texttt{now.executes AND}\ F) \texttt{ => } (F_C \texttt{ => } F_R)]\!]^{c_0,c_1}(e)(s,s') \\ se', \emptyset, Is \vdash C \downarrow^I F_C\ \texttt{AND \$}I = T \end{array}}{se \vdash Ms\ S\ \{C\}}$$

Soundness Proof Take arbitrary $c \in Context, me \in MethodEnv, s, s' \in State, e \in Environment$ and define

(1) $mes := envSeq^{view(c),me,Ms}$

We assume

(2) $specifies(se, me, c)$

(3) $DifferentVariables(c)$

(4) $executes(control(s))$

and show

(a.1) $rspecifies(se, Ms, [\![Ms]\!]_{\mathrm{I}})$

(a.2) $\forall i \in \mathbb{N} : [\![C]\!]_i^{c,mes(i)}(s) \Rightarrow correctness([\![S]\!]_{\mathrm{S}}, c, e, s, s')$

(a.3) $[\![assumption(S)]\!]^{c,c}(e)(s,s) \Rightarrow \exists i \in \mathbb{N} : [\![C]\!]_{\mathrm{T}_i}^{c,mes(i)}(s)$

From hypotheses 1, 2, and 3 we know

(5) $se' = [\![Ms]\!]_{\mathrm{S}}(se)$

(6) $rspecifies(se, Ms, [\![Ms]\!]_{\mathrm{I}})$

From hypotheses 4, 5, and 6, we know

$$(7)\quad \begin{array}{l} S = \texttt{writesonly}\ I_1,\ldots,I_n\ \texttt{throwsonly}\ K_1,\ldots,K_m \\ \quad \texttt{assumes}\ F_C\ \texttt{implements}\ F_R\ \texttt{decreases}\ T \end{array}$$

(8) F_C, F_R and T have no free (mathematical or state) variables

(9) F_C and T do not depend on the poststate

From hypothesis 7 and the definition of "makes a natural number", we know

$$(10)\quad \begin{array}{l} \forall e \in \mathit{Environment}, c, c' \in \mathit{Context}, s, s' \in \mathit{State} : \\ \quad [\![F_C]\!]^{c,c'}(e)(s,s') \Rightarrow [\![T]\!]^{c,c'}(e)(s,s') \in \mathbb{N} \end{array}$$

From hypotheses 2 and 8 and the definition of $[\,\_\,]^{\_}_{\_}$, we know

(11) F has no free (mathematical or state) variables

$$(12)\quad \begin{array}{l} \forall c, c' \in \mathit{Context}, i \in \mathbb{N}, \mathit{me} \in \mathit{MethodEnv}_i : \\ \quad \mathit{rspecifies}_i(\mathit{se}', \mathit{me}, c, [\![\mathit{Ms}]\!]_{\mathrm{I}}) \wedge \mathit{pushes}(c, c', \emptyset)\ \Rightarrow \\ \qquad \forall s, s' \in \mathit{State}, e \in \mathit{Environment} : \\ \qquad\quad \mathit{executes}(\mathit{control}(s)) \wedge [\![C]\!]_i^{c',\mathit{me}}(s,s') \Rightarrow \\ \qquad\qquad [\![[F]^{F_c,F_b,F_r,\{L_1,\ldots,L_r\}}_{M_1,\ldots,M_o}]\!]^{c,c'}(e)(s,s') \end{array}$$

From hypothesis 9, we know

$$(13)\quad \begin{array}{l} \forall c \in \mathit{Context}, s, s' \in \mathit{State}, e \in \mathit{Environment} : \\ \quad \mathit{executes}(\mathit{control}(s)) \wedge \\ \quad [\![[F]^{F_c,F_b,F_r,\{L_1,\ldots,L_r\}}_{M_1,\ldots,M_o}]\!]^{c,c}(e)(s,s') \Rightarrow \\ \qquad \neg \mathit{continues}(\mathit{control}(s')) \wedge \\ \qquad \neg \mathit{breaks}(\mathit{control}(s')) \wedge \\ \qquad s = s'\ \text{EXCEPT}^{c,c}\ I_1,\ldots,I_n \wedge \\ \qquad (\mathit{throws}(\mathit{control}(s'))\ \Rightarrow \\ \qquad\quad \mathit{key}(\mathit{control}(s'))\ \in \{K_1,\ldots,K_m\}) \end{array}$$

From hypothesis 11, we know

$$(14)\quad \begin{array}{l} \forall s, s' \in \mathit{State}, e \in \mathit{Environment}, c_0, c_1 \in \mathit{Context} : \\ \quad [\![(\texttt{now.executes AND}\ F) \texttt{ => } (F_C \texttt{ => } F_R)]\!]^{c_0,c_1}(e)(s,s') \end{array}$$

From (8), (9), and hypotheses 2 and 12, we know

$$(15)\quad \begin{array}{l} \forall c, c' \in \mathit{Context}, i \in \mathbb{N}, \mathit{me} \in \mathit{MethodEnvs}_i : \\ \quad \mathit{rspecifies}_i(\mathit{se}', \mathit{me}, c, [\![\mathit{Ms}]\!]_{\mathrm{I}}) \wedge \mathit{pushes}(c, c', \{V_1,\ldots,V_o\})\ \Rightarrow \\ \qquad \forall s \in \mathit{State}, e \in \mathit{Environment} : \mathit{executes}(\mathit{control}(s)) \Rightarrow \\ \qquad\quad [\![F_C\ \texttt{AND \$}I\texttt{=}T]\!]^{c,c'}(e)(s,s) \wedge i \geq e(I) \Rightarrow [\![C]\!]_{\mathrm{T}_i}^{c',\mathit{me}}(s) \end{array}$$

From (1), (2), (3), (5), (6) and Lemma "Specification of Recursively Defined Functions", we know

(16) $rspecifies_i(se', mes(i), c, [\![Ms]\!]_{\mathrm{I}})$

From (3) and the definition of *pushes*, we know

(17) $pushes(c, c, \emptyset)$

From (6), we know (a.1).

To show (a.2), we take $i \in \mathbb{N}, me \in RecMethodEnvs_i$ and assume

(18) $[\![C]\!]_i^{c,mes(i)}(s)$

From (7) and the definitions of *correctness* and $[\![\_]\!]_{\mathrm{S}}$, it suffices to show

(a.2.1) $[\![F_C \Rightarrow F_R]\!]^{c,c}(e)(s, s')$

(a.2.2) $hs = s' \text{ EXCEPT}^{c,c} I_1, \ldots, I_n$

(a.2.3) $\neg continues(control(s'))$

(a.2.4) $\neg breaks(control(s'))$

(a.2.5) $throws(control(s')) \Rightarrow key(control(s')) \in \{K_1, \ldots, K_m\}$

From (4), (12), (16), (17) and (18), we know

(19) $[\![[F]_{M_1,\ldots,M_o}^{F_c,F_b,F_r,\{L_1,\ldots,L_r\}}]\!]^{c,c}(e)(s, s')$

From (19) and the definitions of $[\_]^{\_}_{\_}$ and $[\![\_]\!]$, we know

(20) $[\![F]\!]^{c,c}(e)(s, s')$

From (4), (14), (20), and the definition of $[\![\_]\!]$, we know (a.2.1).

From (4), (13), and (19), we know

(21) $\neg continues(control(s'))$

(22) $\neg breaks(control(s'))$

(23) $s = s' \text{ EXCEPT}^{c,c} I_1, \ldots, I_n$

(24) $throws(control(s')) \Rightarrow key(control(s')) \in \{K_1, \ldots, K_m\}$

From (23), we know (a.2.2). From (21), we know (a.2.3). From (22), we know (a.2.4). From (24), we know (a.2.5).

To show (a.3), by (7) and the definition of *assumption*, it suffices to assume

(25) $[\![F_C]\!]^{c,c}(e)(s,s)$

and show

(a.3.a) $\exists i \in \mathbb{N} : [\![C]\!]^{c,mes(i)}_{\mathrm{T}_i}(s)$

We define

(26) $i := [\![T]\!]^{c,c}(e)(s,s)$

From (10), (25), and (26), we know

(27) $i \in \mathbb{N}$

From (27), to show (a.3.a), it suffices to show

(a.3.b) $[\![C]\!]^{c,mes(i)}_{\mathrm{T}_i}(s)$

We define

(28) $e_0 := e[I \mapsto [\![T]\!]^{c,c}(e)(s,s)]$

From (4), (15), (16), (17), (25), and the definition of $[\![\_]\!]$, it suffices to show

(a.4.c.1) $[\![F_C]\!]^{c,c}(e_0)(s,s)$
(a.4.c.2) $e_0(I) = [\![T]\!]^{c,c}(e_0)(s,s)$
(a.4.c.3) $i \geq e_0(I)$

From (8), (25), (26), (28), (MVF), and (MVFT), we know (a.4.c.1), (a.4.c.2) and (a.4.c.3). □

6.7.7 Generalizations

The calculus of the recursion language presented up to now is very rigid in the sense that all user-defined methods are considered as (potential) participants of a recursion cycle; correspondingly a universal termination measure has to be found that is increased by every method invocation. In reality, however, many functions are not recursive (such that actually no termination measure would be required for them); furthermore, different function definitions may be actually part of different recursion cycles (with different termination measures).

The solution for dealing with such situations by a more flexible form of the calculus is conceptually simple: take the directed graph whose vertices represent the user-defined methods and whose edges represents the "is called by" relation. We identify in this graph all strongly connected components (SCCs) where a SCC is a maximal subgraph with a path between every pair of nodes that fully lies within that subgraph. Each SCC thus represents a set of methods that are defined recursively, either directly or indirectly by calls of other methods in that set.

By contracting each SCC to a "macro-vertex", we get a directed acyclic graph. By topological sorting, we may put all SCCs in that graph in a sequence such that each method is only called by methods in the same SCC or by a method in a SCC that appears prior in that sequence. The semantics of recursive method declarations given in the previous subsection may thus be considered as the semantics of a single SCC in that sequence; the "base environment" used in the semantics is the environment sequence constructed by the preceding SCC; the base environment of the first SCC is the set of "pre-defined" functions.

In this construction, for every SCC a separate measure may be used which only has to consider the methods within the SCC. Furthermore, rather than using the same well-founded ordering $\langle \mathbb{N}, < \rangle$ as the domain of every measure, we may use different well-founded orderings for different measures: in practice, e.g. $\langle \mathbb{N}^n, <^n \rangle$ is frequently used, where $\mathbb{N}^n$ is the set of n-tuples of natural numbers and $<^n$ represents the lexicographical order among such tuples.

We leave the concrete elaboration of this sketch to future work.

Appendix A

Mathematical Language

This appendix summarizes the mathematical language used in this document. Our theoretical framework is classical Zermelo-Fraenkel set theory (ZF) formalized in first-order predicate logic (FOL), as it is presented in typical introductions to mathematics for computer scientists. We therefore mostly refrain from providing formal definitions of the concepts but focus on a presentation of the notations we use together with informal explanations of their interpretations. The exact definitions can be looked up in various text books on this subject.

In the following descriptions, we use the meta-variables $x, x_1, \ldots$ to denote object variables, $f, f_1, \ldots$ to denote function names, $p, p_1, \ldots$ to denote predicate names, $T, T_1, \ldots$ to denote terms, $F, F_1, \ldots$ to denote formulas, and $P, P_1, \ldots$ to denote generic phrases (terms or formulas).

Terms We use the following kinds of terms to denote values:

- $\mathbf{x}$: the variable x.
- $\mathbf{f}$: the constant (0-ary function) f.
- $\mathbf{f}(\mathbf{T_1}, \ldots, \mathbf{T_n})$: the application of the n-ary function f to $T_1, \ldots, T_n$ ($n \geq 1$).
- $\textbf{SUCH}\ \mathbf{x} : \mathbf{F}$: some x such that F is true for x, if F is true for any value, and x is arbitrary, otherwise.

Formulas We use the following kinds of formulas to denote propositions:

- $\mathbf{p}(\mathbf{T_1}, \ldots, \mathbf{T_n})$: the n-ary predicate p is true for $T_1, \ldots, T_n$ ($n \geq 1$).

- $\mathbf{T_1} = \mathbf{T_2}$: T_1 equals T_2.
- $\neg\mathbf{F}$: F is not true.
- $\mathbf{F_1} \wedge \mathbf{F_2}$: F_1 is true and F_2 is true.
- $\mathbf{F_1} \vee \mathbf{F_2}$: F_1 is true or F_2 is true (also both may be true).
- $\mathbf{F_1} \Rightarrow \mathbf{F_2}$: if F_1 is true, then F_2 is also true (but F_1 may be false).
- $\mathbf{F_1} \Leftrightarrow \mathbf{F_2}$: both F_1 and F_2 are true or both are false.
- $\forall \mathbf{x} : \mathbf{F}$: for every x, F is true.
- $\exists \mathbf{x} : \mathbf{F}$: for some x, F is true.

Generic Phrases We use the following generic phrases:

- **IF F THEN $\mathbf{P_1}$ ELSE $\mathbf{P_2}$**: if F is true, then P_1, else P_2.
- **LET $\mathbf{x} = \mathbf{T}$ IN P**: P, where the value of x is the value of T (the meaning of this phrase is the same as that of $P[T/x]$, see below).

Free and Bound Variables An occurrence of a variable x in one of the phrases SUCH $x : F$, $\forall x : F$, $\exists x : F$, or LET $x = T$ IN P is called *bound* by that phrase. A non-bound occurrence of a variable in a phrase is *free* in that phrase.

Term Substitutions We introduce the following substitutions of terms by other terms in a phrase P:

- $\mathbf{P}[\mathbf{U_1}/\mathbf{T_1}, \ldots, \mathbf{U_n}/\mathbf{T_n}]$: that variant of P where all occurrences of terms $T_1, \ldots, T_n$ (pairwise-different) are replaced by terms $U_1, \ldots, U_n$.
- $\mathbf{P}[\mathbf{U_1}(\mathbf{x})/\mathbf{T_1}(\mathbf{x}), \ldots, \mathbf{U_n}(\mathbf{x})/\mathbf{T_n}(\mathbf{x}) : \mathbf{F}]$: that variant of P where, for every term x for which F is true, all occurrences of terms $T_1(x), \ldots, T_n(x)$ (pairwise different and different for different x) are replaced by the corresponding terms $U_1(x), \ldots, U_n(x)$.

Sets We use the following predicates, constants, and functions on sets:

- $\mathbf{T_1} \in \mathbf{T_2}$: T_1 is in T_2.
- $\mathbf{T_1} \subseteq \mathbf{T_2}$: T_1 is a subset of T_2, i.e., every x which is in T_1 is also in T_2.
- $\emptyset$: the empty set.
- $T_1 \cap T_2$,$T_1 \cup T_2$, $T_1 \setminus T_2$: the intersection, union, and difference of T_1 and T_2.
- $\{\mathbf{T_1}, \ldots, \mathbf{T_n}\}$: the set of values $T_1, \ldots, T_n$.
- $\{\mathbf{x} \in \mathbf{T} : \mathbf{F}\}$: the set of values x in T for which F is true.
- $\{\mathbf{f}(\mathbf{x_1}, \ldots, \mathbf{x_n}) \in \mathbf{T} : \mathbf{F}\}$: the set of values $f(x_1, \ldots, x_n)$ in T such that F is true for $x_1, \ldots, x_n$.
- $\mathbb{B}$: the set $\{\text{TRUE}, \text{FALSE}\}$ of truth values (Boolean values).
- $\mathbb{N}$: the set $\{0, 1, 2, \ldots\}$ of the natural numbers including 0.
- $\mathbb{N}_\mathbf{n}$: the set $\{0, 1, \ldots, n-1\}$ of the n natural numbers less than n (hence $\mathbb{N}_0 = \emptyset$).
- $\mathbb{Z}$: the set $\{0, 1, -1, 2, -2, \ldots\}$ of the integer numbers including 0 (hence $\mathbb{Z}_0 = \emptyset$).
- $\mathbb{Z}_\mathbf{n}$: the set $\{-n, \ldots, -1, 0, 1, \ldots, n-1\}$ of the $2n$ integer numbers greater than or equal $-n$ and less than n.
- $\mathbb{Q}$: the set of all rational numbers.
- $\mathbb{P}(\mathbf{T})$: the powers et of T (the set of its subsets), also considered as the set of relations on T: for every r in $\mathbb{P}(T)$ (i.e. $r \subseteq T$) and for every x in T, $r(x)$ (i.e. $x \in r$) is true or false.
- $\mathbb{P}(\mathbf{T})^\infty$: the set of all infinite subsets of T.

Tuples The datatype *tuple* (ordered sequence of unnamed values) is introduced as follows:

- $\mathbf{T_1} \times \ldots \times \mathbf{T_n}$: the set of tuples $\langle v_1, \ldots, v_n \rangle$ with $v_1 \in T_1, \ldots, v_n \in T_n$; if $t = \langle v_1, \ldots, v_n \rangle$, then $t.i = v_i$ $(1 \leq i \leq n)$.
- $\mathbf{T_t}[\mathbf{i} \mapsto \mathbf{T}]$: the tuple which is identical to tuple T_t except that $T_t.i = T$ (i.e. $T_t[i \mapsto T].i' = T_r.i'$, for all $i' \neq i$).

Records The datatype *record* (ordered sequence of named values) is introduced as follows:

- $\mathbf{t_1} : \mathbf{T_1} \times \ldots \times \mathbf{t_n} : \mathbf{T_n}$: the set of records $\langle t_1 : v_1, \ldots, t_n : v_n \rangle$ ($= \{\langle t_1, v_1 \rangle, \ldots, \langle t_n, v_n \rangle\}$) where $t_1, \ldots, t_n$ are disjoint values (tags) and $v_1 \in T_1, \ldots, v_n \in T_n$; if $r = \langle t_1 : v_1, \ldots, t_n : v_n \rangle$, then $r.t_i = v_i$ $(1 \leq i \leq n)$.
- $\mathbf{T_r}[\mathbf{t} \mapsto \mathbf{T}]$: the record which is identical to record T_r except that $T_r.t = T$ (i.e. $T_r[t \mapsto T].t' = T_r.t'$, for all $t' \neq t$).

Maps The datatype *map* (the set-theoretic counterpart of a function) is introduced as follows:

- $\mathbf{T_1} \xrightarrow{\textbf{part.}} \mathbf{T_2}$: the set of partial maps from T_1 to T_2; for every f in $T_1 \xrightarrow{\text{part.}} T_2$ and for every x in domain$(f) \subseteq T_1$, $f(x)$ is in T_2 and $\langle x, f(x) \rangle$ is in f (i.e. $T_1 \xrightarrow{\text{part.}} T_2$ is a subset of $T_1 \times T_2$).
- $\mathbf{T_1} \rightarrow \mathbf{T_2}$: the set of total maps from T_1 to T_2 (a subset of $T_1 \xrightarrow{\text{part.}} T_2$); for every f in $T_1 \rightarrow T_2$ and for every x in T_1, $f(x)$ is in T_2 (i.e. domain$(f) = T_1$).
- $\mathbf{T_m}[\mathbf{T_x} \mapsto \mathbf{T_y}]$: the map which is identical to map T_m except that $T_m[T_x \mapsto T_y](T_x) = T_y$ (i.e. $T_m[T_x \mapsto T_y](x) = T_m(x)$ for every $x \neq T_x$).
- $[\mathbf{T_x} \mapsto \mathbf{T_y}]$: the map m such that domain$(m) = \{T_x\}$ and $m(T_x) = T_y$.

Sequences The domain *sequence* (finite sequence of arbitrary length) is introduced as follows:

- $\mathbf{A^k} := \mathbb{N}_\mathbf{k} \rightarrow \mathbf{A}$: the set of sequences of length $k \in \mathbb{N}$ whose values are in A; for every $s \in A^k, i \in \mathbb{N}_k, v \in A$, we have $s(i) \in A$ and $s[i \mapsto v] \in A^k$.
- $\mathbf{A^*} := \bigcup_{\mathbf{k} \in \mathbb{N}} \mathbf{A^k}$: the set of finite sequences whose values are in A with function LENGTH $: A^* \rightarrow \mathbb{N}$: for every $s \in A^k \subseteq A^*$, we have LENGTH$(s) = k$.

The domain of infinite sequences is correspondingly introduced:

- $\mathbf{A^\infty} := \mathbb{N} \rightarrow \mathbf{A}$: the set of infinite sequences whose values are in A; for every $s \in A^\infty, i \in \mathbb{N}, v \in A$, we have $s(i) \in A$ and $s[i \mapsto v] \in A^\infty$.

Abbreviations We use the following syntactic abbreviations of terms and formulas ("$P_1 \equiv P_2$" means "P_1 is an abbreviation of P_2"):

- $\textbf{SUCH } \mathbf{x} \in \mathbf{T} : \mathbf{F} \equiv \text{ SUCH } x : x \in T \wedge F$
- $\forall \mathbf{x} \in \mathbf{T} : \mathbf{F} \equiv \forall x : x \in T \Rightarrow F$
- $\forall \mathbf{x_1} \in \mathbf{T_1}, \ldots, \mathbf{x_n} \in \mathbf{T_n} : \mathbf{F} \equiv \forall x_1 \in T_1 : \ldots : \forall x_n \in T_n : F$
- $\forall \mathbf{x_1}, \ldots, \mathbf{x_n} \in \mathbf{T} : \mathbf{F} \equiv \forall x_1 \in T, \ldots, x_n \in T : F$
- $\exists \mathbf{x} \in \mathbf{T} : \mathbf{F} \equiv \exists x : x \in T \wedge F$
- $\exists \mathbf{x_1} \in \mathbf{T_1}, \ldots, \mathbf{x_n} \in \mathbf{T_n} : \mathbf{F} \equiv \exists x_1 \in T_1 : \ldots : \exists x_n \in T_n : F$
- $\exists \mathbf{x_1}, \ldots, \mathbf{x_n} \in \mathbf{T} : \mathbf{F} \equiv \exists x_1 \in T, \ldots, x_n \in T : F$
- $\textbf{LET } \mathbf{x_1} = \mathbf{T_1}, \ldots, \mathbf{x_n} = \mathbf{T_n} \textbf{ IN } \mathbf{P} \equiv \text{ LET } x_1 = T_1 \text{ IN LET } \ldots \text{ IN LET } x_n = T_n \text{ IN } P$
- $\mathbf{T}[\mathbf{T_1} \mapsto \mathbf{T'_1}, \ldots, \mathbf{T_n} \mapsto \mathbf{T'_n}] \equiv T[T_1 \mapsto T'_1] \ldots [T_n \mapsto T'_n]$
- $[\mathbf{T_1} \mapsto \mathbf{T'_1}, \ldots, \mathbf{T_n} \mapsto \mathbf{T'_n}] \equiv [T_1 \mapsto T'_1] \ldots [T_n \mapsto T'_n]$
- $\text{MIN } \mathbf{x} \in \mathbf{S} : \mathbf{F} \equiv \text{ SUCH } x \in S : (F \wedge \neg \exists y \in S : y < x \wedge F[y/x])$
- $\text{MAX } \mathbf{x} \in \mathbf{S} : \mathbf{F} \equiv \text{ SUCH } x \in S : (F \wedge \neg \exists y \in S : y > x \wedge F[y/x])$

Definitions of Relations and Maps We use the following formats to define relations and maps.

- $\mathbf{r} \in \mathbb{P}(\mathbf{T_1} \times \ldots \times \mathbf{T_n}),\ \mathbf{r}(\mathbf{x_1}, \ldots, \mathbf{x_n}) \Leftrightarrow \mathbf{F}$

  This definition introduces a relation r on $T_1 \times \ldots \times T_n$ such that, for all $x_1 \in T_1, \ldots, x_n \in T_n$, the formula $r(x_1, \ldots, x_n)$ is true if and only if F is true.

  The relation $r \in \mathbb{P}(T_1 \times \ldots \times T_n)$ is the set

  $$r = \{\langle x_1, \ldots, x_n \rangle \in T_1 \times \ldots \times T_n : F\}$$

  The formula $r(x_1, \ldots, x_n)$ is the syntactic abbreviation

  $$r(x_1, \ldots, x_n) \equiv \langle x_1, \ldots, x_n \rangle \in r$$

- $\mathbf{f \in T_1 \times \ldots \times T_n \to T_0,\ f(x_1, \ldots, x_n) = T}$

  This definition introduces a total map f from $T_1 \times \ldots \times T_n$ to T_0 such that, for all $x_1 \in T_1, \ldots, x_n \in T_n$, the value of the term $f(x_1, \ldots, x_n)$ is in T_0 and equals the value of T.

  The map $f \in T_1 \times \ldots \times T_n \to T_0$ is the set

  $$f = \{\langle x_1, \ldots, x_n, y\rangle \in T_1 \times \ldots \times T_n \times T_0 :\ y = T\}$$

  The term $f(x_1, \ldots, x_n)$ is the syntactic abbreviation

  $$f(x_1, \ldots, x_n) \ \equiv\ \text{SUCH } y \in T_0 : \langle x_1, \ldots, x_n, y\rangle \in f$$

Since relations and maps are just special sets, they may serve as the values of variables in terms and formulas (in contrast to logical predicates and functions). On the other hand, the abbreviations $r(x_1, \ldots, x_n)$ and $f(x_1, \ldots, x_n)$ make a relation r and a map f usable like a predicate respectively function; we thus use the notions "predicate" and "relation" respectively "function" and "map" interchangeably.

Appendix B

Mathematical Properties

This appendix summarizes various properties of domains introduced in this document and presents (or at least sketches) their proofs.

B.1 States as Plain Stores

This section deals with the properties of states as plain stores based on the following definitions:

$$\begin{aligned}
&\mathit{Store} := \mathit{Variable} \to \mathit{Value}\\
&\mathit{State} := \mathit{Store}
\end{aligned}$$

$$\begin{aligned}
&\mathit{read} : \mathit{State} \times \text{Identifier} \to \mathit{Value}\\
&\mathit{read}(s,I) = s([\![I]\!])
\end{aligned}$$

$$\begin{aligned}
&\mathit{write} : \mathit{State} \times \text{Identifier} \times \mathit{Value} \to \mathit{State}\\
&\mathit{write}(s,I,v) = s[[\![I]\!] \mapsto v]
\end{aligned}$$

$$\mathit{writes}(s,I_1,v_1,\ldots,I_n,v_n) \equiv s[[\![I_1]\!] \mapsto v_1]\ldots[[\![I_n]\!] \mapsto v_n]$$

$$\begin{aligned}
&s_0 \text{ EQUALS } s_1 \equiv\\
&\quad \forall I \in \text{Identifier} : \mathit{read}(s_0,I) = \mathit{read}(s_1,I)
\end{aligned}$$

$$\begin{aligned}
&s_0 = s_1 \text{ AT } I_1,\ldots,I_n \equiv\\
&\quad \forall I \in \text{Identifier} : I = I_1 \vee \ldots \vee I = I_n \Rightarrow \mathit{read}(s_0,I) = \mathit{read}(s_1,I)
\end{aligned}$$

$$s_0 = s_1 \text{ EXCEPT } I_1, \ldots, I_n \equiv$$
$$\forall I \in \text{Identifier} : I \neq I_1 \wedge \ldots \wedge I \neq I_n \Rightarrow \mathit{read}(s_0, I) = \mathit{read}(s_1, I)$$

$$\mathit{DifferentVariables} :\Leftrightarrow \forall I_1, I_2 \in \text{Identifier} : I_1 \neq I_2 \Rightarrow [\![I_1]\!] \neq [\![I_2]\!]$$

The lemmas listed in Figure B.1 are:

ID (Store Identity) This lemma states that writing in a store the value read from that store does not change the store.

RW1 (Reading and Writing Stores 1) This lemma states that reading the last variable written to a store returns the value that was last written.

RW2 (Reading and Writing Stores 2) This lemma says that reading a variable different from the one that was last written to a store returns the value of the variable in the original store.

WS (Writing Stores) This lemma says that the store to which a variable denoted by identifier I has been written and the original store agree in the values of all variables denoted by identifiers different from I *provided that* different identifiers denote different variables.

RS (Reading Stores) This lemma says that if two stores differ only in the value of a variable I_1, they agree on the value of any other variable I_2.

RE (Reflexivity) This lemma says that the plain equality of stores is an exceptional equality for any variable.

SY (Symmetry) This lemma states that exceptional equality is symmetric on stores for single exceptions.

TR (Transitivity) This lemma states that exceptional equality is transitive on stores for single exceptions.

AV (Addition of Variables) This lemma says that any variable may be added to an exception of a store equality.

RV (Removal of Variables) This lemma says that an identifier may be removed from two exceptions to the equality of two stores, if the denoted variable has the same value in both stores.

SV (Swapping of Variables) This lemma says that the order of two identifiers in the list of exceptions of a store equality may be swapped.

Definition

$$\mathit{DifferentVariables} :\Leftrightarrow \forall I_1, I_2 \in \text{Identifier} : I_1 \neq I_2 \Rightarrow [\![I_1]\!] \neq [\![I_2]\!]$$

Store Identity

(ID) $\forall s \in \mathit{State}, I \in \text{Identifier} : s = \mathit{write}(s, I, \mathit{read}(s, I))$

Reading and Writing Stores

(RW1) $\forall s \in \mathit{State}, I \in \text{Identifier}, v \in \mathit{Value} :$
$\mathit{read}(\mathit{write}(s, I, v), I) = v$

(RW2) $\forall s \in \mathit{State}, I_1, I_2 \in \text{Identifier}, v \in \mathit{Value} :$
$[\![I_1]\!] \neq [\![I_2]\!] \Rightarrow \mathit{read}(\mathit{write}(s, I_1, v), I_2) = \mathit{read}(s, I_2)$

Equal Stores with Exceptions

(WS) $\mathit{DifferentVariables} \Rightarrow$
$\forall s \in \mathit{State}, I \in \text{Identifier}, v \in \mathit{Value} :$
$\mathit{write}(s, I, v) = s \text{ EXCEPT } I$

(RS) $\forall s_1, s_2 \in \mathit{State}, I_1, I_2 \in \text{Identifier} :$
$I_1 \neq I_2 \wedge s_1 = s_2 \text{ EXCEPT } I_1 \Rightarrow \mathit{read}(s_1, I_2) = \mathit{read}(s_2, I_2)$

Basic Store Equalities

(RE) $\forall s \in \mathit{State}, I \in \text{Identifier} : s = s \text{ EXCEPT } I$

(SY) $\forall s_1, s_2 \in \mathit{State}, I \in \text{Identifier} :$
$s_1 = s_2 \text{ EXCEPT } I \Rightarrow s_2 = s_1 \text{ EXCEPT } I$

(TR) $\forall s_1, s_2, s_3 \in \mathit{State}, I \in \text{Identifier} :$
$s_1 = s_2 \text{ EXCEPT } I \wedge s_2 = s_3 \text{ EXCEPT } I \Rightarrow s_1 = s_3 \text{ EXCEPT } I$

(AV) $\forall s_1, s_2 \in \mathit{State}, I_1, I_2 \in \text{Identifier} :$
$s_1 = s_2 \text{ EXCEPT } I_1 \Rightarrow s_1 = s_2 \text{ EXCEPT } I_1, I_2$

(RV) $\forall s_1, s_2 \in \mathit{State}, I_1, I_2 \in \text{Identifier} :$
$s_1 = s_2 \text{ EXCEPT } I_1, I_2 \wedge \mathit{read}(s_1, I_2) = \mathit{read}(s_2, I_2) \Rightarrow$
$s_1 = s_2 \text{ EXCEPT } I_1$

(SV) $\forall s_1, s_2 \in \mathit{State}, I_1, I_2 \in \text{Identifier} :$
$s_1 = s_2 \text{ EXCEPT } I_1, I_2 \Rightarrow s_1 = s_2 \text{ EXCEPT } I_2, I_1$

Figure B.1: Store Lemmas (Part 1 of 2)

Extended Store Properties

(IDE) $\forall s \in \mathit{State}, I_1, \ldots, I_n \in \text{Identifier} :$
$s = \mathit{writes}(s, I_1, \mathit{read}(s, I_1), \ldots, I_n, \mathit{read}(s, I_n))$

(RWE) $\forall s \in \mathit{State}, I_1, \ldots, I_n \in \text{Identifier}, v_1, \ldots, v_n \in \mathit{Value} :$
$\text{LET } s' = \mathit{writes}(s, I_1, v_1 \ldots, I_n, v_n) \text{ IN}$
$\mathit{read}(s', I_1) = v_1 \wedge \ldots \wedge \mathit{read}(s', I_n) = v_n$

(WSE) $\mathit{DifferentVariables} \Rightarrow$
$\forall s \in \mathit{State}, I_1, \ldots, I_n \in \text{Identifier}, v_1, \ldots, v_n \in \mathit{Value} :$
$\mathit{writes}(s, I_1, v_1, \ldots, I_n, v_n) = s \text{ EXCEPT } I_1, \ldots, I_n$

(RSE) $\forall s_1, s_2 \in \mathit{State}, I_1, \ldots, I_n, I \in \text{Identifier} :$
$s_1 = s_2 \text{ EXCEPT } I_1, \ldots, I_n \wedge I \neq I_1 \wedge \ldots \wedge I \neq I_n \Rightarrow$
$\mathit{read}(s_1, I) = \mathit{read}(s_2, I)$

(REE) $\forall s \in \mathit{State}, I_1, \ldots, I_n \in \mathit{Identifier} : s = s \text{ EXCEPT } I_1, \ldots, I_n$

(SYE) $\forall s_1, s_2 \in \mathit{State}, I_1, \ldots, I_n \in \mathit{Identifier} :$
$s_1 = s_2 \text{ EXCEPT } I_1, \ldots, I_n \Rightarrow s_2 = s_1 \text{ EXCEPT } I_1, \ldots, I_n$

(TRE) $\forall s_1, s_2, s_3 \in \mathit{State}, I_1, \ldots, I_n \in \mathit{Identifier} :$
$s_1 = s_2 \text{ EXCEPT } I_1, \ldots, I_n \wedge s_2 = s_3 \text{ EXCEPT } I_1, \ldots, I_n \Rightarrow$
$s_1 = s_3 \text{ EXCEPT } I_1, \ldots, I_n$

(AVE) $\forall s_1, s_2 \in \mathit{State}, I_1, \ldots, I_n, J_1, \ldots, J_m \in \text{Identifier} :$
$s_1 = s_2 \text{ EXCEPT } I_1, \ldots, I_n \Rightarrow s_1 = s_2 \text{ EXCEPT } I_1, \ldots, I_n, J_1, \ldots, J_m$

(RVE) $\forall s_1, s_2 \in \mathit{State}, I_1, \ldots, I_n, J_1, \ldots, J_m \in \text{Identifier} :$
$s_1 = s_2 \text{ EXCEPT } I_1, \ldots, I_n, J_1, \ldots, J_m \wedge$
$\mathit{read}(s_1, J_1) = \mathit{read}(s_2, J_1) \wedge \ldots \wedge \mathit{read}(s_1, J_m) = \mathit{read}(s_2, J_m) \Rightarrow$
$s_1 = s_2 \text{ EXCEPT } I_1, \ldots, I_n$

(SVE) $\forall s_1, s_2 \in \mathit{State}, I_1, \ldots I_n \in \mathit{Identifier}, p :$
$s_1 = s_2 \text{ EXCEPT } I_1, \ldots I_n \wedge$
$p \text{ is a permutation of } \{1, \ldots, n\} \Rightarrow$
$s_1 = s_2 \text{ EXCEPT } I_{p(1)}, \ldots, I_{p(n)}$

(NEQ) $\forall s_1, s_2 \in \mathit{State} : s_1 \text{ EQUALS } s_2 \Leftrightarrow s_1 = s_2 \text{ EXCEPT } \_$

Figure B.2: Store Lemmas (Part 2 of 2)

The lemmas listed in Figure B.2 are:

IDE (Store Identity Extended) This lemma states that writing in a store the values read from that store does not change the store.

RWE (Reading and Writing Stores Extended) This states that if we write into a store into multiple variables the values of the same variables from another store, then both stores hold the same values in these variables.

WSE (Writing Stores Extended) This lemma says that if two a store the values of the variables denoted by identifiers $I_1, \ldots, I_n$ are written, the new store and the old store agree on the value of all variables denoted by other identifiers *provided that* different identifiers denote different variables.

RSE (Reading Stores Extended) This lemma says that if two stores differ only in the values of variables $I_1, \ldots, I_n$, they agree on the value of any other variable I.

REE (Reflexivity Extended) This lemma says that the plain equality of stores is an exceptional equality for any collection of variables.

SYE (Symmetry Extended) This lemma says that exceptional equality is symmetric on stores for multiple exceptions.

TRE (Transitivity Extended) This lemma says that exceptional equality is transitive on stores for multiple exceptions.

AVE (Addition of Variables Extended Variables) This lemma states that arbitrary variables may be added to the list of exceptions of a store equality.

RVE (Removal of Variables Extended) This lemma says that identifiers may be removed from the exceptions to the equality of two stores, if the denoted variables have the same value in both stores.

SVE (Swapping of Variables Extended) This lemma says that the identifiers in the list of exceptions of a store equality may be arbitrarily permuted.

NEQ (No Exception Equality) This lemma states that the predicate EQUALS describes equality without exceptions.

The following subsections gives the proofs of these properties.

B.1.1 Reading and Writing Stores

Store Identity

(ID) $\forall s \in \mathit{State}, I \in \text{Identifier} : s = \mathit{write}(s, I, \mathit{read}(s, I))$

Proof: Take arbitrary $s \in \mathit{State}, I \in \text{Identifier}$. By the definitions of *read* and *write*, we know $\mathit{write}(s, I, \mathit{read}(s, I)) = s[[\![I]\!] \mapsto s([\![I]\!])] = s$. □

Reading and Writing 1

(RW1) $\forall s \in \mathit{State}, I \in \text{Identifier}, v \in \mathit{Value} :$
$\quad \mathit{read}(\mathit{write}(s, I, v), I) = v$

Proof: Take arbitrary $s \in \mathit{State}, I \in \text{Identifier}, v \in \mathit{Value}$. Then we know by the definitions of *read* and *write* that $\mathit{read}(\mathit{write}(s, I, v), I) = s[[\![I]\!] \mapsto v]([\![I]\!]) = v$. □

Reading and Writing 2

(RW2) $\forall s \in \mathit{State}, I_1, I_2 \in \text{Identifier}, v \in \mathit{Value} :$
$\quad [\![I_1]\!] \neq [\![I_2]\!] \Rightarrow \mathit{read}(\mathit{write}(s, I_1, v), I_2) = \mathit{read}(s, I_2)$

Proof: Take arbitrary $s \in \mathit{State}, I_1, I_2 \in \text{Identifier}, v \in \mathit{Value}$ and assume

(1) $[\![I_1]\!] \neq [\![I_2]\!]$

Then we know by the definitions of *read* and *write* that $\mathit{read}(\mathit{write}(s, I_1, v), I_2) = s[[\![I_1]\!] \mapsto v]([\![I_2]\!]) = s([\![I_2]\!]) = \mathit{read}(s, I_2)$. □

B.1.2 Equal Stores with Exceptions

Writing Stores

(WS) $\mathit{DifferentVariables} \Rightarrow$
$\quad \forall s \in \mathit{State}, I \in \text{Identifier}, v \in \mathit{Value} :$
$\quad\quad \mathit{write}(s, I, v) = s \text{ EXCEPT } I$

Proof: Assume *DifferentVariables*, i.e.

(1) $\forall I_1, I_2 \in \text{Identifier} : I_1 \neq I_2 \Rightarrow [\![I_1]\!] \neq [\![I_2]\!]$

We show

(a) $\forall s \in \mathit{State}, I \in \text{Identifier}, v \in \mathit{Value} :$
$\quad \mathit{write}(s, I, v) = s \text{ EXCEPT } I$

Take arbitrary $s \in \mathit{State}, I \in \text{Identifier}, v \in \mathit{Value}$. By the definition of EXCEPT, we have to show

(b) $\forall J \in \text{Identifier} : J \neq I \Rightarrow \mathit{read}(\mathit{write}(s, I, v), J) = \mathit{read}(s, J)$

Take arbitrary $J \in \text{Identifier}$ and assume

(2) $J \neq I$

We have to show

(c) $\mathit{read}(\mathit{write}(s, I, v), J) = \mathit{read}(s, J)$

From (1) and (2), we know

(3) $[\![J]\!] \neq [\![I]\!]$

From this and (RW2), we know (c). □

Reading Stores

(RS) $\forall s_1, s_2 \in \mathit{State}, I_1, I_2 \in \text{Identifier} :$
$\quad s_1 = s_2 \text{ EXCEPT } I_1 \wedge I_1 \neq I_2 \Rightarrow \mathit{read}(s_1, I_2) = \mathit{read}(s_2, I_2)$

Proof: Take arbitrary $s_1, s_2 \in \mathit{State}, I_1, I_2 \in \text{Identifier}$ and assume

(1) $s_1 = s_2 \text{ EXCEPT } I_1$
(2) $I_1 \neq I_2$

We have to show

(a) $\mathit{read}(s_1, I_2) = \mathit{read}(s_2, I_2)$

From (1) and the definition of EXCEPT, we know

(3) $\forall I \in \text{Identifier} : I \neq I_1 \Rightarrow \mathit{read}(s, I) = \mathit{read}(s, I)$

From (2) and (3), we know (a). □

B.1.3 Basic Store Equalities

Reflexivity

(RE) $\forall s \in \mathit{State}, I \in \text{Identifier} : s = s \text{ EXCEPT } I$

Proof: Take arbitrary $s \in \mathit{State}, I \in$ Identifier. By the definition of EXCEPT, we have to show

(a) $\forall J \in \text{Identifier} : J \neq I \Rightarrow \mathit{read}(s,J) = \mathit{read}(s,J)$

which is of course true. □

Symmetry

(SY) $\forall s_1, s_2 \in \mathit{State}, I \in \text{Identifier} :$
$s_1 = s_2 \text{ EXCEPT } I \Rightarrow s_2 = s_1 \text{ EXCEPT } I$

Proof: Take arbitrary $s_1, s_2 \in \mathit{State}, I \in$ Identifier and assume

(1) $s_1 = s_2 \text{ EXCEPT } I$

i.e., by the definition of EXCEPT,

(2) $\forall J \in \text{Identifier} : J \neq I \Rightarrow \mathit{read}(s_1,J) = \mathit{read}(s_2,J)$

We have to show

(a) $s_2 = s_1 \text{ EXCEPT } I$

i.e., by the definition of EXCEPT,

(2) $\forall J \in \text{Identifier} : J \neq I \Rightarrow \mathit{read}(s_2,J) = \mathit{read}(s_1,J)$

which follows from (2). □

Transitivity

(TR) $\forall s_1, s_2, s_3 \in \mathit{State}, I \in \text{Identifier} :$
$s_1 = s_2 \text{ EXCEPT } I \wedge s_2 = s_3 \text{ EXCEPT } I \Rightarrow s_1 = s_3 \text{ EXCEPT } I$

Proof: Take arbitrary $s_1, s_2, s_3 \in \mathit{State}, I \in \text{Identifier}$ and assume

(1) $s_1 = s_2 \text{ EXCEPT } I$

(2) $s_2 = s_3 \text{ EXCEPT } I$

We have to show

(a) $s_1 = s_3 \text{ EXCEPT } I$

i.e., by the definition of EXCEPT,

(b) $\forall J \in \text{Identifier} : J \neq I \Rightarrow \mathit{read}(s_1, J) = \mathit{read}(s_3, J)$

From (1) and (2) we know

(3) $\forall J \in \text{Identifier} : J \neq I \Rightarrow \mathit{read}(s_1, J) = \mathit{read}(s_2, J)$

(4) $\forall J \in \text{Identifier} : J \neq I \Rightarrow \mathit{read}(s_2, J) = \mathit{read}(s_3, J)$

from which we know (b). □

Addition of Variables

(AV) $\forall s_1, s_2 \in \mathit{State}, I_1, I_2 \in \text{Identifier} :$
$s_1 = s_2 \text{ EXCEPT } I_1 \Rightarrow s_1 = s_2 \text{ EXCEPT } I_1, I_2$

Proof: Take arbitrary $s_1, s_2 \in \mathit{State}, I_1, I_2 \in \text{Identifier}$ and assume

(1) $s_1 = s_2 \text{ EXCEPT } I_1$

We have to show

(a) $s_1 = s_2 \text{ EXCEPT } I_1, I_2$

i.e. by the definition of EXCEPT

(b) $\forall I \in \text{Identifier} : I \neq I_1 \wedge I \neq I_2 \Rightarrow \mathit{read}(s_1, I) = \mathit{read}(s_2, I)$

From (1) we know by the definition of EXCEPT

(b) $\forall I \in \text{Identifier} : I \neq I_1 \Rightarrow \mathit{read}(s_1, I) = \mathit{read}(s_2, I)$

and thus (b). □

Removal of Variables

$$\text{(RV)}\quad \begin{array}{l}\forall s_1, s_2 \in \mathit{State}, I_1, I_2 \in \text{Identifier}: \\ \quad s_1 = s_2 \text{ EXCEPT } I_1, I_2 \wedge \mathit{read}(s_1, I_2) = \mathit{read}(s_2, I_2) \Rightarrow \\ \quad\quad s_1 = s_2 \text{ EXCEPT } I_1\end{array}$$

Proof: Take arbitrary $s_1, s_2 \in \mathit{State}, I_1, I_2 \in \text{Identifier}$ and assume

(1) $s_1 = s_2$ EXCEPT I_1, I_2

(2) $\mathit{read}(s_1, I_2) = \mathit{read}(s_2, I_2)$

From (1), we know by the definition of EXCEPT

(3) $\forall I \in \text{Identifier} : I \neq I_1 \wedge I \neq I_2 \Rightarrow \mathit{read}(s_1, I) = \mathit{read}(s_2, I)$

We have to show

(a) $s_1 = s_2$ EXCEPT I_1

i.e. by the definition of EXCEPT

(b) $\forall I \in \text{Identifier} : I \neq I_1 \Rightarrow \mathit{read}(s_1, I) = \mathit{read}(s_2, I)$

Take arbitrary $I \in \text{Identifier}$ and assume

(4) $I \neq I_1$

We have to show

(c) $\mathit{read}(s_1, I) = \mathit{read}(s_2, I)$

In the case of $I = I_2$, we know (c) from (2). In the case of $I \neq I_2$, we know (c) from (3) and (4). □

Swapping of Variables

$$\text{(SV)}\quad \begin{array}{l}\forall s_1, s_2 \in \mathit{State}, I_1, I_2 \in \text{Identifier}: \\ \quad s_1 = s_2 \text{ EXCEPT } I_1, I_2 \Rightarrow s_1 = s_2 \text{ EXCEPT } I_2, I_1\end{array}$$

Proof: Take arbitrary $s_1, s_2 \in \mathit{State}, I_1, I_2 \in \text{Identifier}$ and assume

(1) $s_1 = s_2$ EXCEPT I_1, I_2

We have to show

(a) $s_1 = s_2$ EXCEPT I_2, I_1

i.e. by the definition of EXCEPT

(b) $\forall I \in \text{Identifier} : I \neq I_1 \wedge I \neq I_2 \Rightarrow read(s,I) = read(s,I)$

From (1) we know by the definition of EXCEPT

(2) $\forall I \in \text{Identifier} : I \neq I_2 \wedge I \neq I_1 \Rightarrow read(s,I) = read(s,I)$

and thus (b). □

B.1.4 Extended Properties

Store Identity (Extended)

(IDE) $\forall s \in State, I_1, \ldots, I_n \in \text{Identifier} :$
$s = writes(s, I_1, read(s,I_1), \ldots, I_n, read(s,I_n))$

Proof: An extended version of the proof of (ID). □

Reading and Writing Stores (Extended)

(RWE) $\forall s_1, s_2 \in State, I_1, \ldots, I_n \in \text{Identifier} :$
LET $s = writes(s_1, I_1, read(s_2,I_1), \ldots, I_n, read(s_2,I_n))$ IN
$read(s,I_1) = read(s_2,I_1) \wedge \ldots \wedge read(s,I_n) = read(s_2,I_n)$

Proof: An extended version of the proof of (RWE). It should be noted that the lemma is not true for an arbitrary sequence of values written into the store: since the same variable may be denoted by different occurrences in the identifier sequence, it is essential that for these occurrences the same value is written; this is guaranteed by reading these values from another store (where these identifiers then also denote the same variables). □

Writing Stores (Extended)

(WSE) $DifferentVariables \Rightarrow$
$\forall s \in State, I_1, \ldots, I_n \in \text{Identifier}, v_1, \ldots, v_n \in Value :$
$writes(s, I_1, v_1, \ldots, I_n, v_n) = s$ EXCEPT $I_1, \ldots, I_n$

Proof: Analogous to the proof of (WS). □

Reading Stores (Extended)

$$\text{(RSE)}\quad \begin{array}{l}\forall s_1, s_2 \in \mathit{State}, I_1, \ldots, I_n, I \in \text{Identifier}: \\ \quad s_1 = s_2 \text{ EXCEPT } I_1, \ldots, I_n \wedge I \neq I_1 \wedge \ldots \wedge I \neq I_n \Rightarrow \\ \quad\quad \mathit{read}(s_1, I) = \mathit{read}(s_2, I)\end{array}$$

Proof: Analogous to the proof of (RS). □

Reflexivity (Extended)

$$\text{(REE)}\quad \begin{array}{l}\forall s \in \mathit{State}, I_1, \ldots, I_n \in \mathit{Identifier}: \\ \quad s = s \text{ EXCEPT } I_1, \ldots, I_n\end{array}$$

Proof: By (RE) and repeated application of (AV). □

Symmetry (Extended)

$$\text{(SYE)}\quad \begin{array}{l}\forall s_1, s_2 \in \mathit{State}, I_1, \ldots, I_n \in \mathit{Identifier}: \\ \quad s_1 = s_2 \text{ EXCEPT } I_1, \ldots, I_n \Rightarrow s_2 = s_1 \text{ EXCEPT } I_1, \ldots, I_n\end{array}$$

Proof: Analogous to the proof of (SY).

Transitivity (Extended)

$$\text{(TRE)}\quad \begin{array}{l}\forall s_1, s_2, s_3 \in \mathit{State}, I_1, \ldots, I_n \in \mathit{Identifier}: \\ \quad s_1 = s_2 \text{ EXCEPT } I_1, \ldots, I_n \wedge s_2 = s_3 \text{ EXCEPT } I_1, \ldots, I_n \Rightarrow \\ \quad\quad s_1 = s_3 \text{ EXCEPT } I_1, \ldots, I_n\end{array}$$

Proof: Analogous to the proof of (TR).

Addition of Variables (Extended)

$$\text{(AVE)}\quad \begin{array}{l}\forall s_1, s_2 \in \mathit{State}, I_1, \ldots, I_n, I \in \text{Identifier}: \\ \quad s_1 = s_2 \text{ EXCEPT } I_1, \ldots, I_n \Rightarrow s_1 = s_2 \text{ EXCEPT } I_1, \ldots, I_n, I\end{array}$$

Proof: Analogous to the proof of (AV). □

Removal of Variables (Extended)

$$\text{(RVE)}\quad \begin{array}{l}\forall s_1, s_2 \in \mathit{State}, I_1, \ldots, I_n, I \in \text{Identifier}: \\ \quad s_1 = s_2 \text{ EXCEPT } I_1, \ldots, I_n, I \wedge \mathit{read}(s_1, I) = \mathit{read}(s_2, I) \Rightarrow \\ \quad\quad s_1 = s_2 \text{ EXCEPT } I\end{array}$$

Proof: Analogous to the proof of (RVE). □

Swapping of Variables (Extended)

$$\text{(SVE)} \quad \begin{array}{l} \forall s_1, s_2 \in \mathit{State}, I_1, \ldots I_n \in \mathit{Identifier}, p : \\ \quad p \text{ is a permutation of } \{1, \ldots, n\} \Rightarrow \\ \quad\quad s_1 = s_2 \text{ EXCEPT } I_{p(1)}, \ldots, I_{p(n)} \end{array}$$

Proof: By repeated application of (SV). □

No Exception Equality

$$\text{(NEQ)} \quad \forall s_1, s_2 \in \mathit{State} : s_1 \text{ EQUALS } s_2 \Leftrightarrow s_1 = s_2 \text{ EXCEPT } \_$$

Proof: by definition of EQUALS and EXCEPT. □

B.2 Formulas and Terms

The lemmas in this section state that formula respectively term values remain invariant under certain conditions. The proofs of these lemmas are omitted.

The lemmas in Figures B.3 and B.4 state invariance properties under certain modifications of contexts (states or environments):

ESF/EST (Equal Stores) The value of a phrase is invariant in all states that have the same values for the variables denoted by identifiers (i.e. the value of a phrase does not depend on unreferencable variables).

MVF/MVT (Mathematical Variables) A phrase without any free variable occurrences cannot distinguish between

MVF/MVT0 (Mathematical Variables 0) That value of a phrase that does not have any free occurrence of a certain variable is not sensitive to environment updates with respect to that variable.

PVF/PVT1 (Program Variables 1) A phrase without references to the prestate values of program variables cannot distinguish between different prestates.

PVF/PVT1 (Program Variables 2) A phrase without references to the poststate values of program variables cannot distinguish between different poststates.

PVF/PVT3 (Program Variables 3) A phrase without references to the prestate values of program variables $I_1, \ldots, I_n$ cannot distinguish between prestates that only differ in the contents of these variables.

Basic Formula Lemmas

(ESF) $\forall F \in \text{Formula}, e \in \mathit{Environment}, s_0, s_0', s_1, s_1' \in \mathit{State}:$
$s_0 \text{ EQUALS } s_1 \wedge s_0' \text{ EQUALS } s_1' \Rightarrow$
$[\![F]\!](e)(s_0, s_0') \Leftrightarrow [\![F]\!](e)(s_1, s_1')$

(MVF) $\forall F \in \text{Formula} : F \text{ has no free variables } \Rightarrow$
$\forall e_1, e_2 \in \mathit{Environment}, s, s' \in \mathit{State}:$
$[\![F]\!](e_1)(s, s') \Leftrightarrow [\![F]\!](e_2)(s, s')$

(MVF0) $\forall F \in \text{Formula}, I \in \text{Identifier} : \$I \text{ is not free in } F \Rightarrow$
$\forall e \in \mathit{Environment}, s, s' \in \mathit{State}, v \in \mathit{Value}:$
$[\![F]\!](e)(s, s') \Leftrightarrow [\![F]\!](e[I \mapsto v])(s, s')$

(PVF1) $\forall F \in \text{Formula} : F \text{ has no plain program variables } \Rightarrow$
$\forall e \in \mathit{Environment}, s_1, s_2, s' \in \mathit{State}:$
$[\![F]\!](e)(s_1, s') \Leftrightarrow [\![F]\!](e)(s_2, s')$

(PVF2) $\forall F \in \text{Formula} : F \text{ has no primed program variables } \Rightarrow$
$\forall e \in \mathit{Environment}, s, s_1', s_2' \in \mathit{State}:$
$[\![F]\!](e)(s, s_1') \Leftrightarrow [\![F]\!](e)(s, s_2')$

(PVF3) $\forall F \in \text{Formula}, I_1, \ldots, I_n \in \text{Identifier} : I_1, \ldots, I_n \text{ do not occur in } F \Rightarrow$
$\forall e \in \mathit{Environment}, s_1, s_2, s' \in \mathit{State}:$
$s_1 = s_2 \text{ EXCEPT } I_1, \ldots, I_n \Rightarrow [\![F]\!](e)(s_1, s') \Leftrightarrow [\![F]\!](e)(s_2, s')$

(PVF4) $\forall F \in \text{Formula}, I_1, \ldots, I_n \in \text{Identifier} : I_1\texttt{'}, \ldots, I_n\texttt{'} \text{ do not occur in } F \Rightarrow$
$\forall e \in \mathit{Environment}, s, s_1', s_2' \in \mathit{State}:$
$s_1' = s_2' \text{ EXCEPT } I_1, \ldots, I_n \Rightarrow [\![F]\!](e)(s, s_1') \Leftrightarrow [\![F]\!](e)(s, s_2')$

Figure B.3: Basic Formula Lemmas

Basic Term Lemmas

(EST) $\forall T \in \text{Term}, e \in \mathit{Environment}, s_0, s_0', s_1, s_1' \in \mathit{State}:$
$s_0 \text{ EQUALS } s_1 \wedge s_0' \text{ EQUALS } s_1' \Rightarrow$
$[\![T]\!](e)(s_0, s_0') = [\![T]\!](e)(s_1, s_1')$

(MVT) $\forall T \in \text{Term} : T \text{ has no free variables } \Rightarrow$
$\forall e_1, e_2 \in \mathit{Environment}, s, s' \in \mathit{State}:$
$[\![T]\!](e_1)(s, s') = [\![T]\!](e_2)(s, s')$

(MVT0) $\forall T \in \text{Term}, I \in \text{Identifier} : \$I \text{ is not free in } T \Rightarrow$
$\forall e \in \mathit{Environment}, s, s' \in \mathit{State}, v \in \mathit{Value}:$
$[\![T]\!](e)(s, s') = [\![T]\!](e[I \mapsto v])(s, s')$

(PVT1) $\forall T \in \text{Term} : T \text{ has no plain program variables } \Rightarrow$
$\forall e \in \mathit{Environment}, s_1, s_2, s' \in \mathit{State}:$
$[\![T]\!](e)(s_1, s') = [\![F]\!](e)(s_2, s')$

(PVT2) $\forall T \in \text{Term} : T \text{ has no primed program variables } \Rightarrow$
$\forall e \in \mathit{Environment}, s, s_1', s_2' \in \mathit{State}:$
$[\![T]\!](e)(s, s_1') = [\![T]\!](e)(s, s_2')$

(PVT3) $\forall T \in \text{Term}, I_1, \ldots, I_n \in \text{Identifier} : I_1, \ldots, I_n \text{ do not occur in } T \Rightarrow$
$\forall e \in \mathit{Environment}, s_1, s_2, s' \in \mathit{State}:$
$s_1 = s_2 \text{ EXCEPT } I_1, \ldots, I_n \Rightarrow [\![T]\!](e)(s_1, s') = [\![T]\!](e)(s_2, s')$

(PVT4) $\forall T \in \text{Term}, I_1, \ldots, I_n \in \text{Identifier} : I_1\text{'}, \ldots, I_n\text{'} \text{ do not occur in } T \Rightarrow$
$\forall e \in \mathit{Environment}, s, s_1', s_2' \in \mathit{State}:$
$s_1' = s_2' \text{ EXCEPT } I_1, \ldots, I_n \Rightarrow [\![T]\!](e)(s, s_1') = [\![T]\!](e)(s, s_2')$

Figure B.4: Basic Term Lemmas

PVF/PVT3 (Program Variables 4) A phrase without references to the poststate values of program variables $I_1, \ldots, I_n$ cannot distinguish between poststates that only differ in the contents of these variables.

The lemmas in Figure B.5 state that substituting mathematical variables by program variables and vice versa lets these variables disappear from the corresponding phrases (formulas or terms); likewise substituting poststate references to program variables by corresponding prestate references and vice versa lets these references disappear.

The lemmas in Figures B.6 and B.7 state invariance properties under certain variable substitutions:

PPVF1/PPVT1 (Program to Program Variables 1) Replacing all references to the prestate values of program variables by references to their poststate values has the same effect as updating the variables in the prestate by their poststate values.

PPVF2/PPVT2 (Program to Program Variables 2) Replacing all references to the poststate values of program variables by references to their prestate values has the same effect as updating the variables in the poststate by their prestate values.

PMVF1/PMVT1 (Program to Mathematical Variables 1) Replacing all references to the prestate values of some program variables by new mathematical variables does not change the value of a phrase, if the mathematical variables denote these values; furthermore the prestate can be replaced by any prestate that has the same values in all other variables.

PMVF2/PMVT2 (Program to Mathematical Variables 2) Replacing all references to the poststate some values of program variables by new mathematical variables does not change the value of a phrase, if the mathematical variables denote these values; furthermore the poststate can be replaced by any poststate that has the same values in all other variables.

MPVF0/MPVT0 (Mathematical to Program Variables 0) Replacing all references to some plain program variables by mathematical variables generates formulas/terms that do not refer to these plain program variables.

MPVF1/MPVT1 (Mathematical to Program Variables 1) Replacing all references to some primed program variables by mathematical variables generates formulas/terms that do not refer to these primed program variables.

Basic Phrase Substitution Lemmas

(PMVF0) $\forall F \in \text{Formula}, I_1, \ldots, I_n, J_1, \ldots, J_n \in \text{Identifier}:$
$\$J_1, \ldots, \$J_n$ do not occur in $F[I_1/\$J_1, \ldots, I_n/\$J_n]$

(MPVF0) $\forall F \in \text{Formula}, I_1, \ldots, I_n, J_1, \ldots, J_n \in \text{Identifier}:$
$I_1, \ldots, I_n$ do not occur in $F[\$J_1/I_1, \ldots, \$J_n/I_n]$

(MPVF1) $\forall F \in \text{Formula}, I_1, \ldots, I_n, J_1, \ldots, J_n \in \text{Identifier}:$
$I_1\prime, \ldots, I_n\prime$ do not occur in $F[\$J_1/I_1\prime, \ldots, \$J_n/I_n\prime]$

(PMVT0) $\forall T \in \text{Term}, I_1, \ldots, I_n, J_1, \ldots, J_n \in \text{Identifier}:$
$\$J_1, \ldots, \$J_n$ do not occur in $T[I_1/\$J_1, \ldots, I_n/\$J_n]$

(MPVT0) $\forall T \in \text{Term}, I_1, \ldots, I_n, J_1, \ldots, J_n \in \text{Identifier}:$
$I_1, \ldots, I_n$ do not occur in $T[\$J_1/I_1, \ldots, \$J_n/I_n]$

(MPVT1) $\forall T \in \text{Term}, I_1, \ldots, I_n, J_1, \ldots, J_n \in \text{Identifier}:$
$I_1\prime, \ldots, I_n\prime$ do not occur in $T[\$J_1/I_1\prime, \ldots, \$J_n/I_n\prime]$

(PPVF0) $\forall F \in \text{Formula}, I_1, \ldots, I_n \in \text{Identifier}:$
$I_1\prime, \ldots, I_n\prime$ do not occur in $F[I_1/I_1\prime, \ldots, I_n/I_n\prime]$

(PPVT0) $\forall T \in \text{Term}, I_1, \ldots, I_n \in \text{Identifier}:$
$I_1\prime, \ldots, I_n\prime$ do not occur in $T[I_1/I_1\prime, \ldots, I_n/I_n\prime]$

Figure B.5: Basic Phrase Substitution Lemmas

Formula Substitution Lemmas

$$\forall F \in \text{Formula}, e \in \mathit{Environment}, s, s' \in \mathit{State}, I_1, \ldots, I_n \in \text{Identifier}:$$
(PPVF1) $$[\![F[I_1\,'/I_1, \ldots, I_n\,'/I_n]]\!](e)(s,s') \Leftrightarrow$$
$$[\![F]\!](e)(\mathit{writes}(s, I_1, \mathit{read}(s', I_1), \ldots, I_n, \mathit{read}(s', I_n)), s')$$

$$\forall F \in \text{Formula}, e \in \mathit{Environment}, s, s' \in \mathit{State}, I_1, \ldots, I_n \in \text{Identifier}:$$
(PPVF2) $$[\![F[I_1/I_1\,', \ldots, I_n/I_n\,']]\!](e)(s,s') \Leftrightarrow$$
$$[\![F]\!](e)(s, \mathit{writes}(s', I_1, \mathit{read}(s, I_1), \ldots, I_n, \mathit{read}(s, I_n)))$$

$$\forall F \in \text{Formula}, J_1, \ldots, J_n \in \text{Identifier} : \$J_1, \ldots, \$J_n \text{ not in } F \Rightarrow$$
$$\forall I_1, \ldots, I_n \in \text{Identifier}, e \in \mathit{Environment}, s, s', s_0 \in \mathit{State}:$$
$$J_1, \ldots, J_n \text{ is a renaming of } I_1, \ldots, I_n \wedge$$
(PMVF1) $$s = s_0 \text{ EXCEPT } I_1, \ldots, I_n \Rightarrow$$
$$[\![F]\!](e)(s_0, s') \Leftrightarrow$$
$$[\![F[\$J_1/I_1, \ldots, \$J_n/I_n]]\!]$$
$$(e[J_1 \mapsto \mathit{read}(s_0, I_1), \ldots, J_n \mapsto \mathit{read}(s_0, I_n)])(s, s')$$

$$\forall F \in \text{Formula}, J_1, \ldots, J_n \in \text{Identifier} : \$J_1, \ldots, \$J_n \text{ not in } F \Rightarrow$$
$$\forall I_1, \ldots, I_n \in \text{Identifier}, e \in \mathit{Environment}, s, s', s_1 \in \mathit{State}:$$
$$J_1, \ldots, J_n \text{ is a renaming of } I_1, \ldots, I_n \wedge$$
(PMVF2) $$s_1 = s' \text{ EXCEPT } I_1, \ldots, I_n \Rightarrow$$
$$[\![F]\!](e)(s, s_1) \Leftrightarrow$$
$$[\![F[\$J_1/I_1\,', \ldots, \$J_n/I_n\,']]\!]$$
$$(e[J_1 \mapsto \mathit{read}(s_1, I_1), \ldots, J_n \mapsto \mathit{read}(s_1, I_n)])(s, s')$$

$$\forall F \in \text{Formula}, I_1, \ldots, I_n, J_1, \ldots, J_n \in \text{Identifier}:$$
$$J_1, \ldots, J_n \text{ is a renaming of } I_1, \ldots, I_n \wedge$$
(MPVF2) $$I_1, \ldots, I_n \text{ do not occur in } F \Rightarrow$$
$$F[I_1/\$J_1, \ldots, I_n/\$J_n][\$J_1/I_1, \ldots, \$J_n/I_n] = F$$

Figure B.6: Formula Substitution Lemmas

Term Substitution Lemmas

(PPVT1)
$$\begin{array}{l}\forall T \in \text{Term}, e \in \mathit{Environment}, s, s' \in \mathit{State}, I_1, \ldots, I_n \in \text{Identifier} : \\ \quad [\![T[I_1\,' / I_1, \ldots, I_n\,' / I_n]]\!](e)(s,s') = \\ \quad [\![T]\!](e)(\mathit{writes}(s, I_1, \mathit{read}(s', I_1), \ldots, I_n, \mathit{read}(s', I_n)), s')\end{array}$$

(PPVT2)
$$\begin{array}{l}\forall T \in \text{Term}, e \in \mathit{Environment}, s, s' \in \mathit{State}, I_1, \ldots, I_n \in \text{Identifier} : \\ \quad [\![T[I_1/I_1\,', \ldots, I_n/I_n\,']]\!](e)(s,s') = \\ \quad\quad [\![T]\!](e)(s, \mathit{writes}(s', I_1, \mathit{read}(s, I_1), \ldots, I_n, \mathit{read}(s, I_n)))\end{array}$$

(PMVT1)
$$\begin{array}{l}\forall T \in \text{Term}, J_1, \ldots, J_n \in \text{Identifier} : \$J_1, \ldots, \$J_n \text{ do not occur in } F \Rightarrow \\ \quad \forall I_1, \ldots, I_n \in \text{Identifier}, e \in \mathit{Environment}, s, s', s_0 \in \mathit{State} : \\ \quad\quad J_1, \ldots, J_n \text{ is a renaming of } I_1, \ldots, I_n \wedge \\ \quad\quad s = s_0 \text{ EXCEPT } I_1, \ldots, I_n \Rightarrow \\ \quad\quad\quad [\![T]\!](e)(s_0, s') = \\ \quad\quad\quad [\![T[\$J_1/I_1, \ldots, \$J_n/I_n]]\!] \\ \quad\quad\quad\quad (e[J_1 \mapsto \mathit{read}(s_0, I_1), \ldots, J_n \mapsto \mathit{read}(s_0, I_n)])(s, s')\end{array}$$

(PMVT2)
$$\begin{array}{l}\forall T \in \text{Term}, J_1, \ldots, J_n \in \text{Identifier} : \$J_1, \ldots, \$J_n \text{ do not occur in } F \Rightarrow \\ \quad \forall I_1, \ldots, I_n \in \text{Identifier}, e \in \mathit{Environment}, s, s', s_1 \in \mathit{State} : \\ \quad\quad J_1, \ldots, J_n \text{ is a renaming of } I_1, \ldots, I_n \wedge \\ \quad\quad s_1 = s' \text{ EXCEPT } I_1, \ldots, I_n \Rightarrow \\ \quad\quad\quad [\![T]\!](e)(s, s_1) = \\ \quad\quad\quad [\![T[\$J_1/I_1\,', \ldots, \$J_n/I_n\,']]\!] \\ \quad\quad\quad\quad (e[J_1 \mapsto \mathit{read}(s_1, I_1), \ldots, J_n \mapsto \mathit{read}(s_1, I_n)])(s, s')\end{array}$$

(MPVT0)
$$\begin{array}{l}\forall T \in \text{Term}, I_1, \ldots, I_n, J_1, \ldots, J_n \in \text{Identifier} : \\ \quad I_1, \ldots, I_n \text{ do not occur in } T[\$J_1/I_1, \ldots, \$J_n/I_n]\end{array}$$

(MPVT2)
$$\begin{array}{l}\forall T \in \text{Term}, I_1, \ldots, I_n, J_1, \ldots, J_n \in \text{Identifier} : \\ \quad J_1, \ldots, J_n \text{ is a renaming of } I_1, \ldots, I_n \wedge \\ \quad I_1, \ldots, I_n \text{ do not occur in } T \Rightarrow \\ \quad\quad T[I_1/\$J_1, \ldots, I_n/\$J_n][\$J_1/I_1, \ldots, \$J_n/I_n] = T\end{array}$$

Figure B.7: Term Lemmas

Control Data Lemmas

(CD0) $\forall s \in \mathit{State} : s = (\mathit{store}(s), \mathit{control}(s))$

(CD1)
$$\begin{array}{l}\forall s \in \mathit{State}, k \in \mathit{Key}, v \in \mathit{Value} : \\ \quad \mathit{executes}(\mathit{control}(\mathit{execute}(s))) \wedge \\ \quad \mathit{value}(\mathit{control}(\mathit{execute}(s))) = \mathit{value}(\mathit{control}(s)) \wedge \\ \quad \mathit{continues}(\mathit{control}(\mathit{continue}(s))) \wedge \\ \quad \mathit{breaks}(\mathit{control}(\mathit{break}(s))) \wedge \\ \quad \mathit{returns}(\mathit{control}(\mathit{return}(s,v))) \wedge \\ \quad \mathit{value}(\mathit{control}(\mathit{return}(s,v))) = v \wedge \\ \quad \mathit{throws}(\mathit{control}(\mathit{throw}(s,k,v))) \wedge \\ \quad \mathit{key}(\mathit{control}(\mathit{throw}(s,k,v))) = k \wedge \\ \quad \mathit{value}(\mathit{control}(\mathit{throw}(s,k,v))) = v\end{array}$$

(CD2)
$$\begin{array}{l}\forall s \in \mathit{State}, k \in \mathit{Key}, v \in \mathit{Value} : \\ \quad s \text{ EQUALS } \mathit{execute}(s) \ \wedge \\ \quad s \text{ EQUALS } \mathit{continue}(s) \ \wedge \\ \quad s \text{ EQUALS } \mathit{break}(s) \ \wedge \\ \quad s \text{ EQUALS } \mathit{return}(s,v) \ \wedge \\ \quad s \text{ EQUALS } \mathit{throw}(s,k,v)\end{array}$$

(CD3) $\forall s \in \mathit{State}, c \in \mathit{Control} : s \text{ EQUALS } (\mathit{store}(s), c)$

(CD4) $\forall s, s' \in \mathit{State} : s \text{ EQUALS } s' \Leftrightarrow \mathit{store}(s) = \mathit{store}(s')$

Figure B.8: Lemmas for States with Control Data

MPVF2/MPVT2 (Mathematical to Program Variables 2) Replacing all of the occurrences of some mathematical variables by plain program variables that do not occur in the formula/term and then replacing these plain program variables by the mathematical variables again generates the original formula/term.

B.3 States with Control Data

Figures B.8 and B.9 summarize basic properties of states with control data.

Figures B.10, B.11, B.12, B.13, B.14, B.14, B.16, B.17, B.18, and B.19 generalize the properties of phrases given in Appendix B.2 by taking into account the new state control predicates introduced for the specification of programs operating on states with control data.

Reading and Writing Stores (with Control Data)

(CR)
$$\begin{array}{l}\forall s \in \mathit{State}, I \in \text{Identifier}, k \in \mathit{key}, v \in \mathit{Value} : \\ \quad \mathit{read}(s,I) = \mathit{read}(\mathit{execute}(s),I) \\ \quad \mathit{read}(s,I) = \mathit{read}(\mathit{continue}(s),I) \\ \quad \mathit{read}(s,I) = \mathit{read}(\mathit{break}(s),I) \\ \quad \mathit{read}(s,I) = \mathit{read}(\mathit{return}(s,v),I) \\ \quad \mathit{read}(s,I) = \mathit{read}(\mathit{throw}(s,k,v),I)\end{array}$$

(CW)
$$\begin{array}{l}\forall s \in \mathit{State}, I \in \text{Identifier}, v \in \mathit{Value} : \\ \quad \mathit{control}(s) = \mathit{control}(\mathit{write}(s,I,v))\end{array}$$

(CWE)
$$\begin{array}{l}\forall s \in \mathit{State}, I_1, \ldots, I_n \in \text{Identifier} : \\ \quad \mathit{control}(s) = \mathit{control}(\mathit{write}(s,I_1,v_1,\ldots,I_n,v_n))\end{array}$$

Figure B.9: Lemmas for States with Control Data

Basic Formula Lemmas (with State Control Predicates)

(ESF')
$$\begin{array}{l}\forall F \in \text{Formula}, e \in \mathit{Environment}, s_0, s_0', s_1, s_1' \in \mathit{State} : \\ \quad s_0 \text{ EQUALS } s_1 \wedge s_0' \text{ EQUALS } s_1' \wedge \\ \quad \mathit{control}(s_0) = \mathit{control}(s_1) \wedge \mathit{control}(s_0') = \mathit{control}(s_1') \Rightarrow \\ \quad\quad [\![F]\!](e)(s_0,s_0') \Leftrightarrow [\![F]\!](e)(s_1,s_1')\end{array}$$

(MVF')
$$\begin{array}{l}\forall F \in \text{Formula} : F \text{ has no free (mathematical or state) variables } \Rightarrow \\ \quad \forall e_1, e_2 \in \mathit{Environment}, s, s' \in \mathit{State} : \\ \quad\quad [\![F]\!](e_1)(s,s') \Leftrightarrow [\![F]\!](e_2)(s,s')\end{array}$$

(MVF0')
$$\begin{array}{l}\forall F \in \text{Formula}, I \in \text{Identifier} : \$I \text{ is not free in } F \Rightarrow \\ \quad \forall e \in \mathit{Environment}, s, s' \in \mathit{State}, v \in \mathit{Value} : \\ \quad\quad [\![F]\!](e)(s,s') \Leftrightarrow [\![F]\!](e[I \mapsto v])(s,s')\end{array}$$

(MVF1')
$$\begin{array}{l}\forall F \in \text{Formula}, I \in \text{Identifier} : \#I \text{ is not free in } F \Rightarrow \\ \quad \forall e \in \mathit{Environment}, s, s' \in \mathit{State}, c \in \mathit{State} : \\ \quad\quad [\![F]\!](e)(s,s') \Leftrightarrow [\![F]\!](e[I \mapsto c]_c)(s,s')\end{array}$$

Figure B.10: Basic Formula Lemmas

Basic Formula Lemmas (with State Control Predicates)

(PVF1')
$$\begin{array}{l}\forall F \in \text{Formula} : F \text{ has no plain program variables } \Rightarrow \\ \quad \forall e \in \mathit{Environment}, s_1, s_2, s' \in \mathit{State} : \\ \qquad \mathit{control}(s_1) = \mathit{control}(s_2) \Rightarrow \\ \qquad\quad [\![F]\!](e)(s_1, s') \Leftrightarrow [\![F]\!](e)(s_2, s')\end{array}$$

(PVF2')
$$\begin{array}{l}\forall F \in \text{Formula} : F \text{ has no primed program variables } \Rightarrow \\ \quad \forall e \in \mathit{Environment}, s, s'_1, s'_2 \in \mathit{State} : \\ \qquad \mathit{control}(s'_1) = \mathit{control}(s'_2) \Rightarrow \\ \qquad\quad [\![F]\!](e)(s, s'_1) \Leftrightarrow [\![F]\!](e)(s, s'_2)\end{array}$$

(PVF3')
$$\begin{array}{l}\forall F \in \text{Formula}, I_1, \ldots, I_n \in \text{Identifier} : I_1, \ldots, I_n \text{ do not occur in } F \ \Rightarrow \\ \quad \forall e \in \mathit{Environment}, s_1, s_2, s' \in \mathit{State} : \\ \qquad s_1 = s_2 \text{ EXCEPT } I_1, \ldots, I_n \wedge \mathit{control}(s_1) = \mathit{control}(s_2) \Rightarrow \\ \qquad\quad [\![F]\!](e)(s_1, s') \Leftrightarrow [\![F]\!](e)(s_2, s')\end{array}$$

(PVF4')
$$\begin{array}{l}\forall F \in \text{Formula}, I_1, \ldots, I_n \in \text{Identifier} : I_1\texttt{'}, \ldots, I_n\texttt{'} \text{ do not occur in } F \Rightarrow \\ \quad \forall e \in \mathit{Environment}, s, s'_1, s'_2 \in \mathit{State} : \\ \qquad s'_1 = s'_2 \text{ EXCEPT } I_1, \ldots, I_n \wedge \mathit{control}(s'_1) = \mathit{control}(s'_2) \Rightarrow \\ \qquad\quad [\![F]\!](e)(s, s'_1) \Leftrightarrow [\![F]\!](e)(s, s'_2)\end{array}$$

Figure B.11: Basic Formula Lemmas

Basic Formula Substitution Lemmas (with State Control Predicates)

(CNOF0) $\forall F \in \text{Formula}, I \in \textit{Identifier}$:
`now` does not occur in $F[\#I/\texttt{now}]$

(CNEF0) $\forall F \in \text{Formula}, I \in \textit{Identifier}$:
`next` does not occur in $F[\#I/\texttt{next}]$

(CNOF1) $\forall F \in \text{Formula}, I \in \textit{Identifier}$:
`now` does not occur in $F[\texttt{next}/\texttt{now}]$

(CNEF1) $\forall F \in \text{Formula}, I \in \textit{Identifier}$:
`next` does not occur in $F[\texttt{now}/\texttt{next}]$

(CNOF2) $\forall F \in \text{Formula}, J \in \text{Identifier} : \#J$ does not occur in $F \Rightarrow$
$\forall e \in \textit{Environment}, s, s' \in \textit{State}$:
$[\![F]\!](e)(s,s') \Leftrightarrow$
$[\![F[\#J/\texttt{now}]]\!](e[J \mapsto \textit{control}(s)]_c)(s,s')$

(CNEF2) $\forall F \in \text{Formula}, J \in \text{Identifier} : \#J$ does not occur in $F \Rightarrow$
$\forall e \in \textit{Environment}, s, s' \in \textit{State}$:
$[\![F]\!](e)(s,s') \Leftrightarrow$
$[\![F[\#J/\texttt{next}]]\!](e[J \mapsto \textit{control}(s')]_c)(s,s')$

(PVFNO) $\forall F \in \text{Formula} :$ `now` does not occur in $F \Rightarrow$
$\forall e \in \textit{Environment}, s, s' \in \textit{State}, c \in \textit{Control}$:
$[\![F]\!](e)(s,s') \Leftrightarrow [\![F]\!](e)((\textit{store}(s), c), s')$

(PVFNE) $\forall F \in \text{Formula} :$ `next` does not occur in $F \Rightarrow$
$\forall e \in \textit{Environment}, s, s' \in \textit{State}, c \in \textit{Control}$:
$[\![F]\!](e)(s,s') \Leftrightarrow [\![F]\!](e)(s, (\textit{store}(s'), c))$

Figure B.12: Basic Formula Substitution Lemmas

Basic Formula Substitution Lemmas (with State Control Predicates)

(PMVF0') $\forall F \in \text{Formula}, I_1,\ldots,I_n,J_1,\ldots,J_n \in \text{Identifier}:$
$\$J_1,\ldots,\$J_n$ do not occur in $F[I_1/\$J_1,\ldots,I_n/\$J_n]$

(MPVF0') $\forall F \in \text{Formula}, I_1,\ldots,I_n,J_1,\ldots,J_n \in \text{Identifier}:$
$I_1,\ldots,I_n$ do not occur in $F[\,\$J_1/I_1,\ldots,\$J_n/I_n]$

(MPVF1') $\forall F \in \text{Formula}, I_1,\ldots,I_n,J_1,\ldots,J_n \in \text{Identifier}:$
$I_1\texttt{'},\ldots,I_n\texttt{'}$ do not occur in $F[\,\$J_1/I_1\texttt{'},\ldots,\$J_n/I_n\texttt{'}\,]$

(PPVF0') $\forall F \in \text{Formula}, I_1,\ldots,I_n \in \text{Identifier}:$
$I_1\texttt{'},\ldots,I_n\texttt{'}$ do not occur in $F[I_1/I_1\texttt{'},\ldots,I_n/I_n\texttt{'}\,]$

(PPVG0') $\forall F \in \text{Formula}, I_1,\ldots,I_n \in \text{Identifier}:$
$I_1,\ldots,I_n$ do not occur in $F[I_1\texttt{'}/I_1,\ldots,I_n\texttt{'}/I_n]$

(MPVF2') $\forall F \in \text{Formula}, I_1,\ldots,I_n,J_1,\ldots,J_n \in \text{Identifier}:$
$J_1,\ldots,J_n$ is a renaming of $I_1,\ldots,I_n \wedge$
$I_1,\ldots,I_n$ do not occur in $F \Rightarrow$
$F[I_1/\$J_1,\ldots,I_n/\$J_n][\,\$J_1/I_1,\ldots,\$J_n/I_n] = F$

Figure B.13: Basic Formula Substitution Lemmas

Formula Substitution Lemmas (with State Control Predicates)

(PNNF1)
$$\forall F \in \text{Formula}, e \in \mathit{Environment}, s, s' \in \mathit{State}: \\ [\![F[\texttt{next/now}]]\!](e)(s,s') \Leftrightarrow [\![F]\!](e)((\mathit{store}(s), \mathit{control}(s')), s')$$

(PNNF2)
$$\forall F \in \text{Formula}, e \in \mathit{Environment}, s, s' \in \mathit{State}: \\ [\![F[\texttt{now/next}]]\!](e)(s,s') \Leftrightarrow [\![F]\!](e)(s, (\mathit{store}(s'), \mathit{control}(s)))$$

(PPVF1')
$$\forall F \in \text{Formula}, e \in \mathit{Environment}, s, s' \in \mathit{State}, I_1, \ldots, I_n \in \text{Identifier}: \\ [\![F[I_1\texttt{'}/I_1, \ldots, I_n\texttt{'}/I_n]]\!](e)(s,s') \Leftrightarrow \\ [\![F]\!](e)(\mathit{writes}(s, I_1, \mathit{read}(s', I_1), \ldots, I_n, \mathit{read}(s', I_n)), s')$$

(PPVF2')
$$\forall F \in \text{Formula}, e \in \mathit{Environment}, s, s' \in \mathit{State}, I_1, \ldots, I_n \in \text{Identifier}: \\ [\![F[I_1/I_1\texttt{'}, \ldots, I_n/I_n\texttt{'}]]\!](e)(s,s') \Leftrightarrow \\ [\![F]\!](e)(s, \mathit{writes}(s', I_1, \mathit{read}(s, I_1), \ldots, I_n, \mathit{read}(s, I_n)))$$

(PMVF1')
$$\forall F \in \text{Formula}, J_1, \ldots, J_n \in \text{Identifier}: \\ \texttt{now} \text{ does not occur in } F \wedge \$J_1, \ldots, \$J_n \text{ do not occur in } F \Rightarrow \\ \forall I_1, \ldots, I_n \in \text{Identifier}, e \in \mathit{Environment}, s, s', s_0 \in \mathit{State}: \\ J_1, \ldots, J_n \text{ is a renaming of } I_1, \ldots, I_n \wedge \\ s = s_0 \text{ EXCEPT } I_1, \ldots, I_n \Rightarrow \\ [\![F]\!](e)(s_0, s') \Leftrightarrow \\ [\![F[\$J_1/I_1, \ldots, \$J_n/I_n]]\!] \\ (e[J_1 \mapsto \mathit{read}(s_0, I_1), \ldots, J_n \mapsto \mathit{read}(s_0, I_n)])(s,s')$$

(PMVF2')
$$\forall F \in \text{Formula}, J_1, \ldots, J_n \in \text{Identifier}: \\ \texttt{next} \text{ does not occur in } F \wedge \$J_1, \ldots, \$J_n \text{ do not occur in } F \Rightarrow \\ \forall I_1, \ldots, I_n \in \text{Identifier}, e \in \mathit{Environment}, s, s', s_1 \in \mathit{State}: \\ J_1, \ldots, J_n \text{ is a renaming of } I_1, \ldots, I_n \wedge \\ s_1 = s' \text{ EXCEPT } I_1, \ldots, I_n \Rightarrow \\ [\![F]\!](e)(s, s_1) \Leftrightarrow \\ [\![F[\$J_1/I_1\texttt{'}, \ldots, \$J_n/I_n\texttt{'}]]\!] \\ (e[J_1 \mapsto \mathit{read}(s_1, I_1), \ldots, J_n \mapsto \mathit{read}(s_1, I_n)])(s,s')$$

Figure B.14: Formula Substitution Lemmas

Formula Substitution Lemmas (with State Control Predicates)

$$\begin{array}{ll} & \forall F \in \text{Formula}, J_1, \ldots, J_n \in \text{Identifier} : \$J_1, \ldots, \$J_n \text{ not in } F \Rightarrow \\ & \quad \forall I_1, \ldots, I_n \in \text{Identifier}, e \in \mathit{Environment}, s, s', s_0 \in \mathit{State} : \\ & \quad\quad J_1, \ldots, J_n \text{ is a renaming of } I_1, \ldots, I_n \wedge \\ \text{(PMVF1")} & \quad\quad s = s_0 \text{ EXCEPT } I_1, \ldots, I_n \wedge \mathit{control}(s) = \mathit{control}(s_0) \Rightarrow \\ & \quad\quad\quad [\![F]\!](e)(s_0, s') \Leftrightarrow \\ & \quad\quad\quad [\![F[\$J_1/I_1, \ldots, \$J_n/I_n]]\!] \\ & \quad\quad\quad\quad (e[J_1 \mapsto \mathit{read}(s_0, I_1), \ldots, J_n \mapsto \mathit{read}(s_0, I_n)])(s, s') \end{array}$$

$$\begin{array}{ll} & \forall F \in \text{Formula}, J_1, \ldots, J_n \in \text{Identifier} : \$J_1, \ldots, \$J_n \text{ not in } F \Rightarrow \\ & \quad \forall I_1, \ldots, I_n \in \text{Identifier}, e \in \mathit{Environment}, s, s', s_1 \in \mathit{State} : \\ & \quad\quad J_1, \ldots, J_n \text{ is a renaming of } I_1, \ldots, I_n \wedge \\ \text{(PMVF2")} & \quad\quad s_1 = s' \text{ EXCEPT } I_1, \ldots, I_n \wedge \mathit{control}(s_1) = \mathit{control}(s') \Rightarrow \\ & \quad\quad\quad [\![F]\!](e)(s, s_1) \Leftrightarrow \\ & \quad\quad\quad [\![F[\$J_1/I_1\,', \ldots, \$J_n/I_n\,']]\!] \\ & \quad\quad\quad\quad (e[J_1 \mapsto \mathit{read}(s_1, I_1), \ldots, J_n \mapsto \mathit{read}(s_1, I_n)])(s, s') \end{array}$$

Figure B.15: Formula Substitution Lemmas

B.4 Contexts and Global Variables

Figures B.20, B.21, and B.22, depict properties of contexts respectively the behavior of states and formulas under certain context transformations. Figure B.23 shows the behavior of formulas under substitutions of global variables.

Figure B.23 depicts the effect of substitutions of global variable references in phrases.

Term Lemmas (with State Control Predicates)

(EST')
$$\begin{array}{l}\forall T \in \text{Term}, e \in \mathit{Environment}, s_0, s_0', s_1, s_1' \in \mathit{State} : \\ \quad s_0 \text{ EQUALS } s_1 \wedge s_0' \text{ EQUALS } s_1' \wedge \\ \quad \mathit{control}(s_0) = \mathit{control}(s_1) \wedge \mathit{control}(s_0') = \mathit{control}(s_1') \Rightarrow \\ \quad\quad [\![T]\!](e)(s_0, s_0') = [\![T]\!](e)(s_1, s_1')\end{array}$$

(MVT')
$$\begin{array}{l}\forall T \in \text{Term} : T \text{ has no free variables } \Rightarrow \\ \quad \forall e_1, e_2 \in \mathit{Environment}, s, s' \in \mathit{State} : \\ \quad\quad [\![T]\!](e_1)(s, s') = [\![T]\!](e_2)(s, s')\end{array}$$

(PVT1')
$$\begin{array}{l}\forall T \in \text{Term} : T \text{ has no plain program variables } \Rightarrow \\ \quad \forall e \in \mathit{Environment}, s_1, s_2, s' \in \mathit{State} : \\ \quad\quad \mathit{control}(s_1) = \mathit{control}(s_2) \Rightarrow \\ \quad\quad\quad [\![T]\!](e)(s_1, s') \Leftrightarrow [\![T]\!](e)(s_2, s')\end{array}$$

(PVT2')
$$\begin{array}{l}\forall T \in \text{Term} : T \text{ has no primed program variables } \Rightarrow \\ \quad \forall e \in \mathit{Environment}, s, s_1', s_2' \in \mathit{State} : \\ \quad\quad \mathit{control}(s_1') = \mathit{control}(s_2') \Rightarrow \\ \quad\quad\quad [\![T]\!](e)(s, s_1') \Leftrightarrow [\![T]\!](e)(s, s_2')\end{array}$$

Figure B.16: Term Lemmas

Term Substitution Lemmas (with State Control Predicates)

(PPVT1') $\forall T \in \text{Term}, e \in \mathit{Environment}, s, s' \in \mathit{State}, I_1, \ldots, I_n \in \text{Identifier}:$
$[\![T[I_1\prime/I_1, \ldots, I_n\prime/I_n]]\!](e)(s,s') =$
$[\![T]\!](e)(\mathit{writes}(s, I_1, \mathit{read}(s', I_1), \ldots, I_n, \mathit{read}(s', I_n)), s')$

(PPVT2') $\forall T \in \text{Term}, e \in \mathit{Environment}, s, s' \in \mathit{State}, I_1, \ldots, I_n \in \text{Identifier}:$
$[\![T[I_1/I_1\prime, \ldots, I_n/I_n\prime]]\!](e)(s,s') =$
$[\![T]\!](e)(s, \mathit{writes}(s', I_1, \mathit{read}(s, I_1), \ldots, I_n, \mathit{read}(s, I_n)))$

(CNOT1) $\forall T \in \text{Term}, I \in \mathit{Identifier}:$
`now` does not occur in $T[\texttt{next/now}]$

(CNET1) $\forall T \in \text{Term}, I \in \mathit{Identifier}:$
`next` does not occur in $T[\texttt{now/next}]$

(PVTNO) $\forall T \in \text{Term}:$ `now` does not occur in $T \Rightarrow$
$\forall e \in \mathit{Environment}, s, s' \in \mathit{State}, c \in \mathit{Control}:$
$[\![T]\!](e)(s,s') = [\![T]\!](e)((\mathit{store}(s), c), s')$

(PVTNE) $\forall T \in \text{Term}:$ `next` does not occur in $T \Rightarrow$
$\forall e \in \mathit{Environment}, s, s' \in \mathit{State}, c \in \mathit{Control}:$
$[\![T]\!](e)(s,s') = [\![T]\!](e)(s, (\mathit{store}(s'), c))$

(PNNT1) $\forall T \in \text{Term}, e \in \mathit{Environment}, s, s' \in \mathit{State}:$
$[\![T[\texttt{next/now}]]\!](e)(s,s') = [\![T]\!](e)((\mathit{store}(s), \mathit{control}(s')), s')$

(PNNT2) $\forall T \in \text{Term}, e \in \mathit{Environment}, s, s' \in \mathit{State}:$
$[\![T[\texttt{now/next}]]\!](e)(s,s') = [\![T]\!](e)(s, (\mathit{store}(s'), \mathit{control}(s)))$

Figure B.17: Term Substitution Lemmas

Term Substitution Lemmas (with State Control Predicates)

(PMVT1')

$$\begin{array}{l}\forall T \in \text{Term}, J_1, \ldots, J_n \in \text{Identifier}: \\ \quad \texttt{now} \text{ does not occur in } T \wedge \$J_1, \ldots, \$J_n \text{ do not occur in } T \Rightarrow \\ \quad \forall I_1, \ldots, I_n \in \text{Identifier}, e \in \mathit{Environment}, s, s', s_0 \in \mathit{State}: \\ \qquad J_1, \ldots, J_n \text{ is a renaming of } I_1, \ldots, I_n \wedge \\ \qquad s = s_0 \text{ EXCEPT } I_1, \ldots, I_n \Rightarrow \\ \qquad\quad [\![T]\!](e)(s_0, s') = \\ \qquad\quad [\![T[\$J_1/I_1, \ldots, \$J_n/I_n]]\!] \\ \qquad\qquad (e[J_1 \mapsto \mathit{read}(s_0, I_1), \ldots, J_n \mapsto \mathit{read}(s_0, I_n)])(s, s')\end{array}$$

(PMVT2')

$$\begin{array}{l}\forall T \in \text{Term}, J_1, \ldots, J_n \in \text{Identifier}: \\ \quad \texttt{next} \text{ does not occur in } T \wedge \$J_1, \ldots, \$J_n \text{ do not occur in } T \Rightarrow \\ \quad \forall I_1, \ldots, I_n \in \text{Identifier}, e \in \mathit{Environment}, s, s', s_1 \in \mathit{State}: \\ \qquad J_1, \ldots, J_n \text{ is a renaming of } I_1, \ldots, I_n \wedge \\ \qquad s_1 = s' \text{ EXCEPT } I_1, \ldots, I_n \Rightarrow \\ \qquad\quad [\![T]\!](e)(s, s_1) = \\ \qquad\quad [\![T[\$J_1/I_1', \ldots, \$J_n/I_n']]\!] \\ \qquad\qquad (e[J_1 \mapsto \mathit{read}(s_1, I_1), \ldots, J_n \mapsto \mathit{read}(s_1, I_n)])(s, s')\end{array}$$

Figure B.18: Term Substitution Lemmas

Term Substitution Lemmas (with State Control Predicates)

$$\begin{array}{ll} & \forall T \in \text{Term}, J_1, \ldots, J_n \in \text{Identifier} : \$J_1, \ldots, \$J_n \text{ not in } T \Rightarrow \\ & \quad \forall I_1, \ldots, I_n \in \text{Identifier}, e \in \mathit{Environment}, s, s', s_0 \in \mathit{State} : \\ & \qquad J_1, \ldots, J_n \text{ is a renaming of } I_1, \ldots, I_n \wedge \\ \text{(PMVT1")} & \qquad s = s_0 \text{ EXCEPT } I_1, \ldots, I_n \wedge \mathit{control}(s) = \mathit{control}(s_0) \Rightarrow \\ & \qquad\quad [\![T]\!](e)(s_0, s') = \\ & \qquad\quad [\![T[\$J_1/I_1, \ldots, \$J_n/I_n]]\!] \\ & \qquad\qquad (e[J_1 \mapsto \mathit{read}(s_0, I_1), \ldots, J_n \mapsto \mathit{read}(s_0, I_n)])(s, s') \end{array}$$

$$\begin{array}{ll} & \forall T \in \text{Term}, J_1, \ldots, J_n \in \text{Identifier} : \$J_1, \ldots, \$J_n \text{ not in } T \Rightarrow \\ & \quad \forall I_1, \ldots, I_n \in \text{Identifier}, e \in \mathit{Environment}, s, s', s_1 \in \mathit{State} : \\ & \qquad J_1, \ldots, J_n \text{ is a renaming of } I_1, \ldots, I_n \wedge \\ \text{(PMVT2")} & \qquad s_1 = s' \text{ EXCEPT } I_1, \ldots, I_n \wedge \mathit{control}(s_1) = \mathit{control}(s') \Rightarrow \\ & \qquad\quad [\![T]\!](e)(s, s_1) = \\ & \qquad\quad [\![T[\$J_1/I_1\mathit{'}, \ldots, \$J_n/I_n\mathit{'}]]\!] \\ & \qquad\qquad (e[J_1 \mapsto \mathit{read}(s_1, I_1), \ldots, J_n \mapsto \mathit{read}(s_1, I_n)])(s, s') \end{array}$$

Figure B.19: Term Substitution Lemmas

Contexts

$$\begin{array}{ll} & \forall c_0, c_1 \in \mathit{Context} : \\ \text{(COV)} & \quad \mathit{view}(c_0) = \mathit{view}(c_1) \Leftrightarrow c_0 \text{ EQUALS } c_1 \end{array}$$

$$\begin{array}{ll} & \forall c \in \mathit{Context}, I_1, \ldots, I_n \in \text{Identifier} : \\ \text{(COP)} & \quad \mathit{DifferentVariables}(c) \Rightarrow \\ & \qquad \mathit{pushes}(c, \mathit{push}(c, I_1, \ldots, I_n), \{I_1, \ldots, I_n\}) \end{array}$$

$$\begin{array}{ll} & \forall c, c' \in \mathit{Context}, I_1, \ldots, I_n \in \text{Identifier} : \\ \text{(COC)} & \quad \mathit{pushes}(c, c', \{I_1, \ldots, I_n\}) \Rightarrow \\ & \qquad \mathit{DifferentVariables}(\mathit{call}(\mathit{view}(c), c')) \end{array}$$

Figure B.20: Contexts

Contexts and States

(COS)
$$\begin{array}{l}\forall c, c' \in \mathit{Context}, I_1, \ldots, I_n \in \text{Identifier}, s, s' \in \mathit{State} : \\ \quad \mathit{pushes}(c, c', \{I_1, \ldots, I_n\}) \wedge \\ \quad s = s' \text{ EXCEPT}^{\mathit{call}(\mathit{view}(c), c')} \; I_1, \ldots, I_n \; \Rightarrow \\ \quad\quad s \text{ EQUALS}^{c'} \; s'\end{array}$$

Figure B.21: Contexts and States

Contexts and Formulas

(COF1)
$$\begin{array}{l}\forall F \in \text{Formula}, I_1, \ldots, I_n \in \text{Identifier} : \\ \quad I_1, \ldots, I_n \text{ do not occur in } F \wedge \\ \quad I_1\texttt{'}, \ldots, I_n\texttt{'} \text{ do not occur in } F \; \Rightarrow \\ \quad\quad \forall s, s' \in \mathit{Store}, e \in \mathit{Environment}, c, c', c'' \in \mathit{Context} : \\ \quad\quad\quad c' = c'' \text{ EXCEPT } I_1, \ldots, I_n \; \Rightarrow \\ \quad\quad\quad\quad ([\![F]\!]^{c,c'}(e)(s, s') \Leftrightarrow [\![F]\!]^{c,c''}(e)(s, s'))\end{array}$$

(COF2)
$$\begin{array}{l}\forall F \in \text{Formula}, I_1, \ldots, I_n, J_1, \ldots, J_n \in \text{Identifier} : \\ \quad J_1, \ldots, J_n \text{ is a renaming of } I_1, \ldots, I_n \wedge \\ \quad \$I_1, \ldots, \$I_n, \$J_1, \ldots, \$J_n \text{ do not occur in } F \; \Rightarrow \\ \quad\quad \forall s, s' \in \mathit{Store}, e \in \mathit{Environment}, c, c', c'' \in \mathit{Context} : \\ \quad\quad\quad c' = c'' \text{ EXCEPT } I_1, \ldots, I_n \; \Rightarrow \\ \quad\quad\quad\quad ([\![F]\!]^{c,c'}(e)(s, s')) \; \Leftrightarrow \\ \quad\quad\quad\quad [\![\texttt{EXISTS } \$I_1, \ldots, \$I_n, \$J_1, \ldots, \$J_n : \\ \quad\quad\quad\quad\quad F[\$I_1/I_1, \ldots, \$I_n/I_n, \$J_1/I_1\texttt{'}, \ldots, \$J_n/I_n\texttt{'}\,]]\!]^{c,c''} \\ \quad\quad\quad\quad\quad\quad (e)(s, s'))\end{array}$$

Figure B.22: Contexts and Formulas

Global Variable Substitutions

(PMGF)
$$\begin{array}{l}\forall F \in \text{Formula}, I_1, \ldots, I_n \in \text{Identifier} : \\ \quad \forall s, s' \in \mathit{State}, e \in \mathit{Environment}, c, c', c'' \in \mathit{Context} : \\ \quad\quad c = c'' \text{ AT } I_1, \ldots, I_n \wedge c' = c'' \text{ EXCEPT } I_1, \ldots, I_n \; \Rightarrow \\ \quad\quad\quad [\![F]\!]^{c,c''}(e)(s, s') \Leftrightarrow \\ \quad\quad\quad [\![F[\texttt{?}I_1/I_1, \ldots, \texttt{?}I_n/I_n, \texttt{?}I_1\texttt{'}/I_1\texttt{'}, \ldots, \texttt{?}I_n\texttt{'}/I_n\texttt{'}\,]]\!]^{c,c'}(e)(s, s')\end{array}$$

Figure B.23: Global Variable Subsitutions

References

[1] R.-J. Back and J. von Wright. *Refinement Calculus: A Systematic Introduction*. Springer, New York, 1998.

[2] Bernhard Beckert, Reiner Hähnle, and Peter H. Schmitt, editors. *Verification of Object-Oriented Software: The KeY Approach*. Springer, Berlin, Germany, 2007.

[3] Raymond T. Boute. Calculational Semantics: Deriving Programming Theories from Equations by Functional Predicate Calculus. *ACM Transactions on Programming Languages and Systems*, 28(4):747–793, July 2006.

[4] Mike Gordon. Specification and Verification I. Lecture Notes, http:// www.cl.cam.ac.uk/ mjcg/Teaching/SpecVer1/SpecVer1.html.

[5] Eric C.R. Hehner. *A Practical Theory of Programming*. Springer, New York, 2006. http://www.cs.utoronto.ca/˜hehner/aPToP.

[6] C.A.R. Hoare and He Jifeng. *Unifying Theories of Programming*. Prentice Hall, London, UK, 1998.

[7] The Java Modeling Language (JML), 2008. http://www.cs.ucf.edu/ leavens/JML.

[8] Cliff B. Jones. *Systematic Software Devleopment Using VDM*. Prentice Hall, 2nd edition, 1990.

[9] K. Rustan M. Leino and James B. Saxe and Raymie Stata. Checking Java Programs via Guarded Commands. Compaq SRC Technical Note 1999-002, Compaq, 1999. http://gatekeeper.dec.com/pub/DEC/SRC/technical-notes/abstracts/src-tn-1999-002.html.

[10] Leslie Lamport. *Specifying Systems; The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002. http:// research.microsoft.com/users/lamport/tla/book.html.

[11] Carroll Morgan. *Programming from Specifications*. Prentice Hall, London, UK, 2nd edition, 1998.

[12] J. M. Morris. A Theoretical Basis for Stepwise Refinement and the Programming Calculus. *Science of Computer Programming*, 9(3):287–306, December 1987.

[13] David A. Schmidt. *Denotational Semantics: A Methodology for Language Development*. Allyn and Bacon, 1986. http://people.cis.ksu.edu/~schmidt/text/densem.html.

[14] Spec#, 2008. http://research.microsoft.com/SpecSharp.