The Temporal Logic of Actions II

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC-Linz)
Johannes Kepler University, A-4040 Linz, Austria

Wolfgang.Schreiner@risc.uni-linz.ac.at
http://www.risc.uni-linz.ac.at/people/schreine

Proving Simple Program Properties

- Program P:
  - **var natural** x, $y = 0$
    do
    $\langle$**true** $\rightarrow x := x + 1\rangle$
    []
    $\langle$**true** $\rightarrow y := y + 1\rangle$
    od
- TLA Formula Φ:
  - $\mathit{Init}_\Phi \equiv (x = 0) \wedge (y = 0)$
  - $\mathsf{M}_1 \equiv (x' = x + 1) \wedge (y' = y)$
  - $\mathsf{M}_2 \equiv (y' = y + 1) \wedge (x' = x)$
  - $\mathsf{M} \equiv \mathsf{M}_1 \vee \mathsf{M}_2$
  - $\Phi \equiv \mathit{Init}_\Phi \wedge \Box[\mathsf{M}]_{\langle x,y\rangle}$
    $\wedge\ \mathsf{WF}_{\langle x,y\rangle}(\mathsf{M}_1) \wedge \mathsf{WF}_{\langle x,y\rangle}(\mathsf{M}_2)$
- Program P has property F:
  - $\Phi \Rightarrow F$

Invariance Properties

- TLA formula $\Box P$.
- *Partial correctness*
  - If program has terminated, answer is correct.
- *Deadlock freedom*
  - Program is not deadlocked.
- *Mutual exclusion*
  - At most one process is in critical section.
- Proofs based on rule INV1.
  - $\dfrac{I \land [\mathsf{N}]_f \Rightarrow I'}{I \land \Box[\mathsf{N}]_f \Rightarrow \Box I}$

Example: Type Correctness

- Type declarations in TLA:
  - Invariance property assuring that program variables are always from certain domain.
  - $\Phi \Rightarrow \Box T$
- **natural** x, y
  - $T \equiv (x \in \textbf{Nat}) \land (y \in \textbf{Nat})$.
- Must prove:
  - $\mathit{Init}_\Phi \Rightarrow T$
    $T \land [\mathsf{M}]_{\langle x,y\rangle} \Rightarrow T'$
- Then we know:
  - Φ
    $\Rightarrow \mathit{Init}_\Phi \land [\mathsf{M}]_{\langle x,y\rangle}$
    $\Rightarrow T \land \Box[\mathsf{M}]_{\langle x,y\rangle}$
    $\Rightarrow \Box T$

Proof

- Prove $T \wedge [\mathsf{M}]_{\langle x,y \rangle} \Rightarrow T'$
  - $T \wedge \mathsf{M}_1 \Rightarrow T'$
  - $T \wedge \mathsf{M}_2 \Rightarrow T'$
  - $T \wedge (\langle x,\ y\ \rangle' = \langle x,\ y\ \rangle) \Rightarrow T'$
- Prove $T \wedge \mathsf{M}_1 \Rightarrow T'$
  - $T' \equiv ((x \in \textbf{Nat}) \wedge (y \in \textbf{Nat}))'$
    $\equiv (x' \in \textbf{Nat}) \wedge (y' \in \textbf{Nat})$
  - $T \wedge \mathsf{M}_1 \Rightarrow x' \in \textbf{Nat}$ $T \wedge \mathsf{M}_1 \Rightarrow y' \in \textbf{Nat}$
- Prove $T \wedge \mathsf{M}_1 \Rightarrow x' \in \textbf{Nat}$
  - $T \wedge \mathsf{M}_1$
    $\Rightarrow (x \in \textbf{Nat}) \wedge (x' = x + 1)$
    $\Rightarrow x' \in \textbf{Nat}$

Proofs "mechanically" guided by the structure of formulas.

General Invariance Proofs

- Special case $\Phi \Rightarrow \Box T$
  - T was invariant of $[\mathsf{M}]_{\langle x,y \rangle}$
  - T could be used as I in INV1.
- Generally $\Phi \Rightarrow \Box P$
  - P need *not* be invariant.
  - Find invariant $I \Rightarrow P$
- Creativity is in finding I
  - Invariance proof itself mechanical.

INV1 reduces temporal reasoning to ordinary (non-temporal) reasoning!

More About Invariance Proofs

- Use one invariance property to prove another.
  - Know $\Phi \Rightarrow \Box T$.
  - Prove $\Phi \Rightarrow \Box P$.
- Application of rule INV2.
  - $\vdash \Box I \Rightarrow (\Box[\mathsf{N}]_f \equiv \Box[\mathsf{N} \land I \land I']_f)$
  - $\Phi \equiv \mathit{Init}_\Phi \land \Box[\mathsf{M} \land T \land T']_{\langle x,y\rangle}$
    $\land\ \mathsf{WF}_{\langle x,y\rangle}(\mathsf{M}_1) \land \mathsf{WF}_{\langle x,y\rangle}(\mathsf{M}_2)$
  - Can substitute $\mathsf{M} \land T \land T'$ instead of M for N in INV1.

Eventuality Properties

- Something *eventually* happens.
- *Termination*
  - $\Diamond$*terminated*.
- *Service*
  - If process has requested service, it is eventually served.
  - *requested* $\leadsto$ *served*.
- *Message delivery*
  - If a message is sent often enough, it is eventually delivered.
  - $(\Box\Diamond \mathit{sent}) \Rightarrow \Diamond \mathit{delivered}$.
- $P \leadsto Q$.
  - $\Phi \wedge (n \in \mathbf{Nat}) \Rightarrow \Diamond(x > n)$
  - $\Phi \Rightarrow ((n \in \mathbf{Nat} \wedge x = n) \leadsto \Diamond(x = n + 1))$

Must be derived from fairness condition!

Example

- Prove WF1
  - $P \leftarrow n \in \textbf{Nat} \wedge x = n$ $Q \leftarrow x = n{+}1$
    $\mathsf{N} \leftarrow \mathsf{M}$, $\mathsf{A} \leftarrow \mathsf{M}_1$, $f \leftarrow \langle x, y \rangle$
- Hypotheses:
  - $(n \in \textbf{Nat} \wedge x = n) \wedge [\mathsf{M}]_{\langle x,y \rangle}$
    $\Rightarrow ((n \in \textbf{Nat} \wedge x' = n) \vee (x' = n{+}1))$
    $(n \in \textbf{Nat} \wedge x = n) \wedge \langle \mathsf{M}_1 \rangle_{\langle x,y \rangle}$
    $\Rightarrow (x' = n{+}1))$
    $(n \in \textbf{Nat} \wedge x = n) \wedge \langle \mathsf{M}_1 \rangle_{\langle x,y \rangle}$
    $\Rightarrow \textit{Enabled}\ \langle \mathsf{M}_1 \rangle_{\langle x,y \rangle}$
  - From definitions of M_1 and M.
- Conclusion:
  - $\Box[\mathsf{M}]_{\langle x,y \rangle} \wedge \mathrm{WF}_{\langle x,y \rangle}(\mathsf{M}_1)$
    $\Rightarrow ((n \in \textbf{Nat} \wedge x = n) \mapsto (x = n{+}1))$

Other Properties

- What about more complicated properties"
  - A behavior begins with x and y both zero, and repeatedly increments either x or y (in a single operation), choosing non-deterministically between them, but choosing each infinitely many times.
- Exactly our formula Φ!
  - No distinction between program and property.
  - View Φ as *description* of program.
  - View Φ as *specification* of program.
- Consider a program Ψ.
  - Show that $\Psi \Rightarrow \Phi$.

Another Example

var integer x, $y = 0$;
var semaphore *sem* $= 1$;
cobegin
  loop
    α_1: $\langle P(sem)\rangle$;
    β_1: $\langle x := x + 1\rangle$
    γ_1: $\langle V(sem)\rangle$;
  endloop
[]
  loop
    α_2: $\langle P(sem)\rangle$;
    β_2: $\langle y := y + 1\rangle$
    γ_2: $\langle V(sem)\rangle$;
  endloop
coend

- Program is *informal* description.
- *Real* definition is formula Ψ.

The Formula Ψ

- $\Psi \equiv \mathit{Init}_\Psi \wedge \Box[\mathsf{N}]_w \wedge \mathsf{SF}_w(\mathsf{N}_1) \wedge \mathsf{SF}_w(\mathsf{N}_2)$
- $\mathit{Init}_\Psi \equiv (\mathit{pc}_1 = \text{“a”}) \wedge (\mathit{pc}_2 = \text{“a”})$
  $\wedge\ (x = 0) \wedge (y = 0) \wedge (\mathit{sem}{=}1)$
- $w \equiv \langle x, y, \mathit{sem}, \mathit{pc}_1, \mathit{pc}_2 \rangle$
- $\mathsf{N} \equiv \mathsf{N}_1 \vee \mathsf{N}_2$
- $\mathsf{N}_1 \equiv \alpha_1 \vee \beta_1 \vee \gamma_1$
- $\mathsf{N}_2 \equiv \alpha_2 \vee \beta_2 \vee \gamma_2$
- $\alpha_1 \equiv (\mathit{pc}_1 = \text{“a”}) \wedge (0 < \mathit{sem})$
  $\wedge\ \mathit{pc}_1' = \text{“b”} \wedge \mathit{sem}' = \mathit{sem}\text{-}1$
  $\wedge\ \mathit{Unchanged}\ \langle x, y, \mathit{pc}_2 \rangle$
- $\beta_1 \equiv \mathit{pc}_1 = \text{“b”}$
  $\wedge\ \mathit{pc}_1' = \text{“g”} \wedge x' = x + 1$
  $\wedge\ \mathit{Unchanged}\ \langle x, y, \mathit{pc}_2 \rangle$
- $\gamma_1 \equiv \mathit{pc}_1 = \text{“g”}$
  $\wedge\ \mathit{pc}_1' = \text{“a”} \wedge \mathit{sem}' = \mathit{sem}{+}1$
  $\wedge\ \mathit{Unchanged}\ \langle x, y, \mathit{pc}_2 \rangle$
- $\alpha_2 \equiv \ldots, \beta_2 \equiv \ldots, \gamma_2 \equiv \ldots$

The Next-State Relation

- α_1 step:
  - Starts in state with *pc*$_1$ = "a" (first process is at control point α_1) and $0 <$ *sem* (no process in critical section).
  - Ends in staet with *pc*$_1$ = "b" (first process is at control point β_1).
  - Decrements *sem* and does not change x, y, *pc*$_2$.
- N_1 step:
  - α_1 step or β_1 step or γ_1 step.
  - Execution of atomic operation by first process.
- N step:
  - Step of either process.
  - The program's next-state relation.

The Fairness Requirement

- Ψ shall implement Φ.
  - x and y must be incremented infinitely often.
  - Infinitely many N_1 and N_2 steps must occur.
- Assume only N_2 steps occur.
- Does $WF_w(N_1)$ rule out this?
  - *Enabled* $\alpha_1 \equiv$ (*pc*$_1$ = “a”) $\wedge$ (0 ¡ *sem*).
  - α_1 is enabled and disabled infinitely often.
  - $\langle N_1 \rangle_w$ is disabled infinitely often.
  - $WF_w(N_1)$ still holds for this behavior!
- Does $SF_w(N_1)$ rule out this?
  - Either $\langle N_1 \rangle_w$ is eventually disabled forever, or infinitely many $\langle N_1 \rangle_w$ steps occur.
  - $\langle N_1 \rangle_w$ is enabled infinitely often.
  - $SF_w(N_1)$ does not hold for this behavior!

Need strong fairness condition!

Proving Ψ Implements Φ

- Prove $\Psi \Rightarrow \Phi$
  - $\mathit{Init}_\Psi \Rightarrow \mathit{Init}_\Phi$
  - $\Box[\mathsf{N}]_w \Rightarrow \Box[\mathsf{M}]_{\langle x,y \rangle}$
  - $\Psi \Rightarrow \mathsf{WF}_{\langle x,y \rangle}(\mathsf{M}_1) \land \mathsf{WF}_{\langle x,y \rangle}(\mathsf{M}_2)$
- Proof of Step Simulation:
  - $[\mathsf{N}]_w \Rightarrow [\mathsf{M}]_{\langle x,y \rangle}$
  - $[\mathsf{N}]_w \equiv \alpha_1 \lor \ldots \lor \gamma_2 \lor (w' = w)$
  - $\beta_1 \Rightarrow \mathsf{M}_1$
  - $\beta_2 \Rightarrow \mathsf{M}_2$
  - $(\langle x, y \rangle' = \langle x, y \rangle)$ for all others.

Proof of Fairness

- $\Psi \Rightarrow \mathsf{WF}_{\langle x,y \rangle}(\mathsf{M}_1)$
  - x is incremented infinitely often.
  - Application of SF_2.
  - Use β_1 for B.
  - Strengthen N by invariant *I* through application of INV2.
  - $I \equiv x \in \mathbf{Nat}$

$$
\begin{array}{l}
\land\ (((sem=1) \land (pc_1 = pc_2 = \text{“a”})) \\
\quad \lor\ ((sem=0) \\
\qquad \land\ (((pc_1 = \text{“a”}) \land (pc_2 \in \{\text{“b”}, \text{“g”}\})) \\
\qquad\quad \lor\ ((pc_2 = \text{“a”}) \\
\qquad\qquad \land\ (pc_1 \in \{\text{“b”}, \text{“g”}\})))))
\end{array}
$$

For details, see the paper.

Hiding Variables

- A simple processor/memory interface:
  - Processor issues *read* and *write* operations executed by memory.
- Three interface registers:
  - *op*: set by processor to indicate operation, reset by memory after operation.
  - *adr* set by processor to indicate memory address to be read or written.
  - *val* set by processor to indicate value to be written, set by memory to return result of *read*.
- Specification Φ:
  - *memory*(n) current value of location n.
  - **Address** set of legal address.
  - **MemVal** set of possible memory values.
  - Action S(m,v) assignment *memory*(m):=v.
  - Processor actions R_{proc}, W_{proc}.
  - Memory responses R_{mem}, W_{mem}.

Formal Specification

- $\Phi \equiv \mathit{Init}_\Phi \land \Box[\mathsf{N}]_w \land \mathsf{WF}_w(\mathsf{N}_{mem})$
- $\mathit{Init}_\Phi \equiv \mathit{op} =$ "ready"
  $\land\ \forall n \in$ **Address**: $\mathit{memory}(n) \in$ **MemVal**
- $\mathsf{N} \equiv \mathsf{N}_{mem} \lor \mathsf{R}_{proc} \lor \mathsf{W}_{proc}$
- $\mathsf{N}_{mem} \equiv \mathsf{R}_{mem} \lor \mathsf{W}_{mem}$
- $w \equiv \langle \mathit{op}, \mathit{adr}, \mathit{val}, \mathit{memory} \rangle$
- $\mathsf{S}(m,v) \equiv \forall n \in$ **Address**:
  $(n = m) \Rightarrow (\mathit{memory}(n)' = v)$
  $\land (n \neq m) \Rightarrow (\mathit{memory}(n)' = \mathit{memory}(n))$
- Fairness condition:
  - Memory eventually responds to each request.
  - Processor need not issue requests.

Formal Specification (Contd)

- $\mathsf{R}_{proc} \equiv op = \text{“ready”}$
  $\land\ op' = \text{“read”} \land adr' \in \mathbf{Address}$
  $\land\ memory' = memory$
- $\mathsf{W}_{proc} \equiv op = \text{“ready”}$
  $\land\ op' = \text{“write”} \land adr' \in \mathbf{Address}$
  $\land\ val' \in \mathbf{MemVal}$
  $\land\ memory' = memory$
- $\mathsf{R}_{mem} \equiv op = \text{“read”}$
  $\land\ op' = \text{“ready”} \land val' = memory(adr)$
  $\land\ memory' = memory$
- $\mathsf{W}_{mem} \equiv op = \text{“write”}$
  $\land\ op' = \text{“ready”}$
  $\land\ \mathsf{S}(adr, val)$

- Only interested in memory *interface*:
  - Behavior of *op*, *adr*, *val*.
  - Behavior of *memory* should be hidden.
  - $\exists memory : \Phi$.

Quantification over Flexible Variables

- $\exists x$: F
  - Flexible variable x.
  - There exists values for x such that F holds.
- Auxiliary definitions:
  - $s =_x t$: states s and t assign same values to all variables other than x.
  - $s =_x t \equiv \forall 'v' \neq 'x'\ s[[v]] = t[[v]]$
  - $\langle s_0, s_1, \ldots \rangle =_x \langle t_0, t_1, \ldots \rangle \equiv \forall n \in \textbf{Nat}$: $s_n =_x t_n$

Quantification over Flexible Variables

- "Obvious" definition:
  - $\sigma[[\exists x\colon F]] \equiv \exists \tau \in \textbf{St}^\infty\colon (\sigma =_x \tau) \wedge \tau[[\mathrm{F}]]$
  - Not correct since not necessarily invaraint under stuttering!
- Remove stuttering steps:
  - $\natural\langle s_0, s_1, \ldots\rangle \equiv$
    $\quad\textbf{if } \forall n \in \textbf{Nat}\colon s_n = s_0$
    $\quad\quad\textbf{then } \langle s_0, s_0, \ldots\rangle$
    $\quad\quad\textbf{else if } s_1 = s_0 \textbf{ then } \natural\langle s_1, s_2, \ldots\rangle$
    $\textbf{else } \langle s_0\rangle \circ \natural\langle s_1, \ldots\rangle$
- TLA = STLA + quantification.
  - Existential quantifier over flexible and rigid variables.
  - All TLA formulas are invariant under stuttering:
    $\natural\sigma = \natural\tau \Rightarrow \sigma[[F]] = \tau[[F]]$

Quantification in TLA

- Syntax:
  - $\langle$*general formula*$\rangle \equiv \langle$*STLA formula*$\rangle$
    $\mid \exists\langle$*flexible variable*$\rangle$: $\langle$*general formula*$\rangle$
    $\mid \exists\langle$*rigid variable*$\rangle$: $\langle$*general formula*$\rangle$
    $\mid \langle$*general formula*$\rangle\wedge \langle$*general formula*$\rangle$
    $\mid \neg\langle$*general formula*$\rangle$
- Semantics:
  - $\sigma[[\exists x\colon F]] \equiv \exists \rho, \tau \in \mathbf{St}^{\infty}$:
    $(\sharp\sigma = \sharp\rho) \wedge (\rho =_x \tau) \wedge \tau[[\mathrm{F}]]$
  - $\sigma[[\exists c\colon F]] \equiv \exists c \in \mathbf{Val}\colon \sigma[[F]]$
- Proof rules:
  - E1. $\vdash F(f/x) \Rightarrow \exists\ x\colon F$
  - E2. $\dfrac{F \Rightarrow G}{(\exists x\colon F) \Rightarrow G}$, x not free in G.
  - F1. $\vdash F(e/c) \Rightarrow \exists\ c\colon F$
  - F2. $\dfrac{F \Rightarrow G}{(\exists c\colon F) \Rightarrow G}$, c not free in G.

Refinement Mappings

- Implementation of memory interface.
  - $\exists$*memory*: Φ.
  - Main memory *main* and cache memory *cache*.
  - *cache*(m) cache value for location m or $\perp$.
- Actions:
  - T(a, m, v) assignment $a(m) := v$.
  - R$_{pro}$, W$_{pro}$ processor *read* and *write* request.
  - R$_{cch}$, W$_{cch}$ response to processor requests serviced by the cache.
  - C$_{get}(m)$, C$_{fl}(m)$ moving value from memory to cache and flushing value from cache to memory.
  - P next-state relation (disjunctions of all actions).
  - F disjunction of memory actions.

A Simple Cached Memory

- $\Phi \equiv \mathit{Init}_\Phi \wedge \Box[\mathsf{P}]_u \wedge \mathsf{WF}_u(\mathsf{F})$.
- $\mathit{Init}_\Phi \equiv \mathit{op} =$ "ready"
  $\wedge\ \forall n \in$ **Address**:
  $(\mathit{main}(n) \in$ **MemVal**$) \wedge (\mathit{cache}(n) = \bot)$
- $u \equiv \langle \mathit{op}, \mathit{adr}, \mathit{val}, \mathit{main}, \mathit{cache} \rangle$
- $\mathsf{P} \equiv \mathsf{R}_{pro} \vee \mathsf{W}_{pro} \vee \mathsf{R}_{cch} \vee \mathsf{W}_{cch}$
  $\vee\ (\exists m \in$ **Address**: $\mathsf{C}_{get}(m) \vee \mathsf{C}_{fl}(m))$
- $\mathsf{F} \equiv \mathsf{R}_{pro} \vee \mathsf{W}_{pro} \vee (\mathsf{C}_{get}(\mathit{adr}) \wedge (\mathit{op} =$ "read"$))$
- $\mathsf{T}(a, m, v) \equiv \forall n \in$ **Address**:
  $(n = m) \Rightarrow (a'(n) = v)$
  $\wedge\ (n' \neq m) \Rightarrow (a'(n) = a(n))$
- $\mathsf{R}_{pro} \equiv \mathit{op} =$ "ready"
  $\wedge\ \mathit{op}' =$ "read" $\wedge\ \mathit{adr}' \in$ **Address**
  $\wedge\ \mathit{Unchanged}\ \langle \mathit{main}, \mathit{cache} \rangle$
- $\mathsf{W}_{pro} \equiv \mathit{op} =$ "ready"
  $\wedge \mathit{op}' =$ "write" $\wedge\ \mathit{adr}' \in$ **Address**
  $\wedge\ \mathit{val}' \in$ **MemVal**
  $\wedge\ \mathit{Unchanged}\ \langle \mathit{main}, \mathit{cache} \rangle$

A Simple Cached Memory (Contd)

- $\mathsf{C}_{get}(m) \equiv \textit{cache}(m) = \bot$
  $\wedge\ \mathsf{T}(\textit{cache}, m, \textit{main}(m))$
  $\wedge\ \textit{Unchanged}\ \langle \textit{op}, \textit{adr}, \textit{val}, \textit{main} \rangle$
- $\mathsf{R}_{cch} \equiv \textit{op} = \text{“read”} \wedge \textit{cache}(\textit{adr}) \neq \bot$
  $\wedge\ \textit{op}' = \text{“ready”} \wedge \textit{val}' = \textit{cache}(\textit{adr})$
  $\wedge\ \textit{Unchanged}\ \langle \textit{main}, \textit{cache} \rangle$
- $\mathsf{W}_{cch} \equiv \textit{op} = \text{“write”}$
  $\wedge\ \textit{op}' = \text{“ready”} \wedge \mathsf{T}(\textit{cache}, \textit{adr}, \textit{val})$
  $\wedge\ \textit{Unchanged}\ \textit{main}$
- $\mathsf{C}_{fl}(m) \equiv \textit{cache}(m) \neq \bot$
  $\wedge\ (\textit{op} \neq \text{“read”} \vee m \neq \textit{adr})$
  $\wedge\ \mathsf{T}(\textit{main}, m, \mathrm{cache}(m))$
  $\wedge\ \mathsf{T}(\textit{cache}, m, \bot)$
  $\wedge\ \textit{Unchanged}\ \langle \textit{op}, \textit{adr}, \textit{val} \rangle$

Formal Specification

- Correctness statement:
  - $(\exists \mathit{main}, \mathit{cache}\colon \Psi) \Rightarrow (\exists \mathit{memory}\colon \Phi)$
- Proof:
  - $\overline{\mathit{memory}}(m) \equiv$ **if** $\mathit{cache}(m) = \perp$ **then** $\mathit{main}(m)$ **else** $\mathit{cache}(m)$
  - $\Psi \Rightarrow \Phi(\overline{\mathit{memory}}/\mathit{memory})$
  - "Concrete" state function $\overline{\mathit{memory}}$ implements "abstract" variable *memory*.
- Cached memory still abstract:
  - No particular cache maintenance policy is specified.
  - Given a concrete caching algorithm, it has to be proved that it implements the simple cached memory.

Refinement Mappings

- *Refinement Mappings*
  - Prove: $(\exists x_1, \ldots, x_m \colon \Psi) \Rightarrow (\exists y_1, \ldots, y_n \colon \Phi)$
  - Define state functions $\overline{y_1}, \ldots, \overline{y_n}$ in terms of the variables occuring in Ψ.
  - Prove $\Psi \Rightarrow \overline{\Phi}$.
  - $\overline{\Phi} := \Phi(\overline{y_1}/y_1, \ldots, \overline{y_n}/y_n)$.
- Mapping need not exist:
  - Can prove: $(\exists \mathit{sem}, \mathit{pc}_1, \mathit{pc}_2 \colon \Psi) \Rightarrow \Phi$.
  - Cannot prove: $\Phi \Rightarrow (\exists \mathit{sem}, \mathit{pc}_1, \mathit{pc}_2 \colon \Psi)$
  - Cannot define state functions $\overline{\mathit{sem}}, \overline{\mathit{pc}_1}, \overline{\mathit{pc}_2}$ in terms of x and y.
- Addition of auxiliary variables:
  - $(\exists h, p \colon \Phi^{hp}) \Rightarrow (\exists \mathit{sem}, \mathit{pc}_1, \mathit{pc}_2 \colon \Psi)$
  - Using auxiliary variables, refinement mappings can be always found.

Summary

- TLA formulas describe algorithms:
  - Effects of all statements.
  - Control flow.
  - Liveness properties.
- Advantages:
  - Independent of language.
  - *All* information is explicitly specified in mathematical formulas.
- Problems:
  - TLA formulas may get very large.
  - Good structure and abstractions required to manage complexity.