The Abstraction Principle II

Wolfgang Schreiner
Research Institute for Symbolic Computation
Johannes Kepler University, Linz, Austria

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

Variable Declarations

- Alias abstraction.
  - Naming of an allocated location.
- Variable declaration.
  - Allocation of a fresh location.
  - (Potentially) initialization.
  - Binding to identifier.
- Part command and part abstraction.
  - Core language construct?
  - Eagerly evaluated abstraction!

Abstract Syntax

D ::= ...| **var** I
C ::= L:=E | C_1;C_2 | ...
E ::= N | @L | ...
L ::= I

Example

var X; **var** Y; **proc** P = Y:=@X+1
in X:=0; **call** P

- Copy rule of littel use
  - Location numbers are *anonymous*!
- Partial application of copy rule
  - **var** X; **var** Y **in** X:=0; Y:=@X+1

Semantics of Variables

$[[\pi \vdash$ **var** I: $\{$I:*intloc*$\}$*dec*$]]e\ s = (\{$I$=l\},\ s')$
where $(l,\ s') =$ *allocate*(s)
and *allocate*: *Store* $\rightarrow$
(*Location* $\times$ *Store*)
allocate $\langle n_1,\ n_2,\ \ldots,\ n_m\rangle =$
$($*loc*$_{m+1},\ \langle n_1,\ n_2,\ \ldots,\ n_m,$ *init*$\rangle)$
where *init* is some arbitrary integer

- Abstraction (I=*l*)
- Command ($s \rightarrow s'$)
- Declarations can alter store!

$[[\pi \vdash$ **define** I=U: $\{$I:$\theta\}$*dec*$]]e\ s = (\{$I$=f\},\ s)$
where $f\ s' = [[\pi \vdash$ U: $\theta]]e\ s'$
$[[$D **in** C: *comm*$]]s = [[\pi \vdash$C: *comm*$]]e_1\ s_1$,
where $(e_1,\ s_1) = [[\emptyset \vdash$ D: π*dec*$]]\emptyset\ s$

Type-Structures

D $\in$ Declaration
T $\in$ Type-structure
D ::= ... | **var** I:T
T ::= **newint**

- Type structure T
  - Allocates storage!
- Declaration **var** I:T
  - Activates T.
  - Binds result value to I.
- Contrast to procedures
  - **proc** I=C alters storage only when procedure is invoked (*lazy evaluation*).
  - **var** I:T alters storage when variable is declared (*eager evaluation*).

Type-Structure Abstractions

D ::= . . . | **var** I:T | **class** I=T
T ::= **newint** | I

- Class = type-structure abstraction.
  - **type** I=T

class *M* = **newint**
var *A*:*M*; **var** *B*:*M*;
var *C*:**newint**
in *A*:=0; *B*:=@*A*+@*C*

- *A* and *B* bind to distinct locations!
- Body of *M* is evaluated each time it is invoked.
- Lazy evaluation of type abstractions.
- Copy rule applies:
  - **var** *A*: **newint**; **var** *B*: **newint**;
    var *C*:**newint**
    in *A*:=0; *B*:=@*A*+@*C*

Classes and Variables

- **class** I=T
- **var** I:T
- Identical structure and components.
- Only semantics distinguishes between these two.

A class is a type-structure abstraction that evaluates lazily, and a variable declaration is a type-structure abstraction that evaluates eagerly.

Record Structures

D ::= ...| **var** I:T | **class** I=T
T ::= **newint** | I | **record** D **end**

- When activated, **record** D **end** allocates store and returns (declaration) value.
- Activation by variable name declaration.

Examples

class R = **record** **var** X: **newint**, **var** Y: **newint** **end**
var A: R;
var B: R

var A: **newint**;
class R = **record** **var** C: **newint**;
  proc $P = (C{:=}@A{+}1)$ **end**;
var F: R; **var** G: R
in ... $F.C$ ... **call** $G.P$ ...

Identifier Expressions

X $\in$ Identifier-expr
X ::= I | X.I
...
C ::= L:=E | C_1;C_2 | ...| **call** X
E ::= N | @L | ...| X
L ::= X

- $\pi \vdash$ **newint**:*intloc class*
- $$\frac{\pi \vdash \text{D: } \pi_1 \textit{dec}}{\pi \vdash \textbf{record}\ \text{D}\ \textbf{end}\text{: } \pi_1 \textit{class}}$$
- $$\frac{\pi \vdash \text{T:}\delta \textit{class}}{\pi \vdash \textbf{var}\ \text{I:T: } \{\text{I:}\delta\}\textit{dec}}$$
- $$\frac{\pi \vdash \text{T:}\delta \textit{class}}{\pi \vdash \textbf{class}\ \text{I=T: } \{\text{I:}\delta \textit{class}\}\textit{dec}}$$
- $\pi \vdash$ I:θ, if (I:θ) $\in \pi$
- $$\frac{\pi \vdash \text{X:}\pi_1}{\pi \vdash \text{X.I: } \tau} \quad \text{if (I:}\tau) \in \pi_1$$

Type Attributes of Type Structures

- Type structures have command-like semantics:
  - When evaluated with current environment and store, it alters store and returns value that can be bound to identifier.
  - $[[\pi \vdash \mathsf{T}{:}\delta\mathit{class}]] \in \mathit{Env}_{\pi} \rightarrow \mathit{Store} \rightarrow ([[\delta]] \times \mathit{Store})$
- Range of type attributes:
  - $\delta ::= \mathit{intloc} \mid \pi$
  - $\pi ::= \{i{:}\tau_i\}_{i \in I}$, where $I \subseteq$ Identifier
  - $\theta ::= \tau \mid \tau\mathit{exp} \mid \mathit{comm} \mid \delta \mid \delta\mathit{class} \mid \pi\mathit{dec}$
  - $\tau ::= \mathit{int} \mid \mathit{bool}$
- $[[\delta\mathit{class}]] = \mathit{Store} \rightarrow ([[\delta]] \times \mathit{Store})$.

Semantics of Type Structures I

- Program
  - $[[$D **in** C: *comm*$]]s = [[\pi_1 \vdash$ C: *comm*$]]e_1\ s$,
    where $(e_1,\ s_1) = [[\emptyset \vdash$ D: π_1*dec*$]]\emptyset\ s$
- Declaration
  - $[[\pi \vdash$ **define** I=U: $\{$I:$\theta\}$*dec*$]]e\ s = (\{$I$=f\},\ s)$
    where $f\ s' = [[\pi \vdash$ U:$\theta]]e\ s'$ and U $\in\{$E, T, C$\}$
  - $[[\pi \vdash$ D$_1$;D$_2$: $(\pi_1 \dot\cup\ \pi_2)$*dec*$]]e\ s =$
    $(e_1 \cup e_2,\ s_2)$ where
    $(e_1,\ s_1) = [[\pi \vdash$ D$_1$: π_1*dec*$]]e\ s$
    $(e_2,\ s_2) = [[\pi \dot\cup\ \pi_1 \vdash$ D$_2$: π_2*dec*$]]\ (e \cup e_1)\ s_1$
  - $[[\pi \vdash$**var** I:T: $\{$I:$\delta\}$*dec*$]]e\ s = (\{$I$=v\},\ s')$
    where $(v,\ s') = [[\pi \vdash$ T: δ*class*$]]e\ s$

Semantics of Type Structures II

- Type-structure
  - $[[\pi \vdash$ **newint**: *intloc class*$]]e\ s = $ *allocate*(s)
  - $[[\pi \vdash$**record** D **end**: π*class*$]]e\ s =$
    $[[\pi \vdash$ D:π*dec*$]]e\ s$
  - $[[\pi \vdash$ X: δ*class*$]]e\ s = p(s)$
    where $p = [[\pi \vdash$ X: δ*class*$]]e$
- Identifier-expr
  - $[[\pi \vdash \mathsf{I}\text{: } \theta]]e = v$,
    where $(\mathsf{I}{=}v) \in e$
  - $[[\pi \vdash \mathsf{X.I}\text{: } \theta]]e = v$,
    where $(\mathsf{I}{=}v) \in [[\pi \vdash \mathsf{X}\text{: } \pi_1]]e$

Declaration Abstractions

D ::= … | **module** I={D} | **import** I

- Module = abstraction of declarations.
- Module invocation = *import*.

module *M* = { **class** *K* = **newint**; **var** *A*: *K*;
  proc *P* = *A*:=0 };
var *B*: **newint**;
import *M*
in call *P*; *B*:=@*A*

Declarations in M is not visible in program until M is imported.

Semantics of Declaration Abstractions

- $$\frac{\pi \vdash \text{D: } \pi_1 \textit{dec}}{\textbf{module } \text{I=}\{\text{D}\}\text{: } \{\text{I:}\pi_1 \textit{dec}\}\textit{dec}}$$
- $\pi \vdash \textbf{import } \text{I: } \pi_1 \textit{dec} \quad \text{if } (\text{I:}\pi_1 \textit{dec}) \in \pi$
- $[[\pi \vdash \textbf{module } \text{I=}\{\text{D}\}\text{: } \{\text{I:}\pi_1 \textit{dec}\}\textit{dec}]]e\ s =$
  $(\{\text{I=}f\}, s)$
  where $f(s') = [[\pi \vdash \text{D: } \pi_1 \textit{dec}]]e\ s'$
- $[[\pi \vdash \textbf{import } \text{I: } \pi_1 \textit{dec}]]e\ s = f(s)$
  where $(\text{I=}f) \in e$
- Lazy evaluation semantics of abstraction.
- Storage cell for module's variables not allocated until module imported.

module M = {**var** A: **newint**};
module N = {**var** B: **newint**};
import M **in** A:=0

Only one location is allocated!

Alternative Semantics

- $[[\pi \vdash$ **module** I=$\{$D$\}$: $\{$I:π_1*dec*$\}$*dec*$]]e\ s =$
  $(\{$I$=e_1\},\ s_1)$
  where $(e_1,\ s_1) = [[\pi \vdash$ D: π_1*dec*$]]e\ s$
- $[[\pi \vdash$ **import** I: π_1*dec*$]]e\ s = (e_1,\ s)$
  where (I$=e_1) \in e$

Eager evaluation allocates storage when module is defined.

Indexing of Module Components

module M = { **class** K = **newint**; **var** A: K;
  proc P = A:=0 };
var B: **newint**;
in call $M.P$; B:=@$M.A$

- Eager evaluation of module definition is essential.
- Ensures that storage for $M.A$ is allocated when $M.P$ is called.

Sharing of Modules

module $S = \{$ **var** A: K;
  fun F=@A+1 $\}$;
module $M = \{$ **import** S; **proc** $INIT$=A:=0 $\}$;
module $N = \{$ **import** S; **proc** $SUCC$=A:=F $\}$;
in call $M.INIT$; **call** $N.SUCC$

- S is *shared* by M and N.
- *Same* A is altered by M and N.
- Eager evaluation of S essential.

Redeclaration of Modules

module S = { **var** A: K; **fun** F=@A+1 };
module M = { **import** S; **proc** $INIT$=A:=0 };
module N = { **import** S; **proc** $SUCC$=A:=F };
import M; **import** N; **in call** $INIT$; **call** $SUCC$

- Prohibited by typing rules.
- S would be imported *twice*!
- However, makes sense if S is evaluated *eagerly*!

Alternative typing rules required for eagerly evaluated modules and multiple import of same module.

External Modules

var *A*: **newint**, **import** *M*
in *A*:=@*B*+1

- Modules separately coded and compiled.
- Type-checking of programs with external modules?
- Above program well typed if *M* contains a declaration for *B*.

var *A*: **newint**,
import external *M*: { B: *intloc*, C: *comm* }
in *A*:=@*B*+1

- *M* is "parameter" to program.
- Additional information gives type of parameter.

Idea is explored later.

Modules and Classes

- Modules and classes are named compound declarations.
  - Module is *eagerly* evaluated compound declaration.
  - Class is *lazily* evaluated compound declaration.
- Simulation of **module** M=D
  - **class** K=D
    var M: K
  - Module *sharing* is more difficult.
- Simulation of **class** K=D
  - D is record of variables V_i and procedures P_i.
  - Make module M of procedures P_i.
  - Pass record V_i as additional argument to $M.P_i$.
  - Very unpleasant!

Deal with this issue later.

Record Introduction Principle

Abstraction Principle

> Phrases of any "semantically meaningful syntactic class" may be named.

- Names to well-formed phrases of any θ.
  - $\pi \vdash$ U: θ
  - $\pi \vdash$ **define** I=U: $\{$I:$\theta\}$*dec*
- Declaration biuilds phrase that is binding of a name to a value.
- Denotational semantics builds singleton set $\{(\mathsf{I}{=}v)\}$.
- Compound declaration builds set $\{\mathsf{I}_1{=}v_1,\ldots,\mathsf{I}_n{=}v_n\}$.
- Semantics of a declaration is a *record*.

Phrases of any semantically meaningful syntactic class may be components of records.

Summary (Abstract Syntax)

P ∈ Program	L ∈ Variable
D ∈ Declaration	N ∈ Numeral
T ∈ Type-structure	X ∈ Identifier-expr
C ∈ Command	I ∈ Identifier
E ∈ Expression	

P ::= D **in** C
D ::= D_1,D_2 | D_1;D_2 | **fun** I=E | **const** I=N
  | **proc** I=C | **var** I:T | **class** I=T
  | **module** I={D} | **import** X
T ::= **newint** | **record** D **end** | X
C ::= L:=E | C_1;C_2 | **if** E **then** C_1 **else** C_2 **fi**
  | **while** E **do** C **od** | **skip** | **call** X
E ::= N | @L | E_1+E_2 | E_1=E_2 | ¬E | X
L ::= X
N ::= n | X
X ::= I | X.I

Summary (Type Attributes)

- Type attributes
  - $\theta ::= \tau \mid \tau\mathit{exp} \mid \mathit{comm} \mid \delta \mid \delta\mathit{class} \mid \pi\mathit{dec}$
  - $\tau ::= \mathit{int} \mid \mathit{bool}$
  - $\delta ::= \mathit{intloc} \mid \pi$
  - $\pi ::= \{i{:}\tau_i\}_{i \in I}$, where $I \subseteq$ Identifier
- Meaning of type attributes:
  - $[[\mathit{int}]] = \mathit{Int}$.
  - $[[\mathit{bool}]] = \mathit{Bool}$.
  - $[[\tau\mathit{exp}]] = \mathit{Store} \to [[\tau]]$.
  - $[[\mathit{comm}]] = \mathit{Store} \to \mathit{Store}_\perp$.
  - $[[\mathit{intloc}]] = \mathit{Location}$.
  - $[[\delta\mathit{class}]] = \mathit{Store} \to ([[\delta]] \times \mathit{Store})$.
  - $[[\pi\mathit{dec}]] = \mathit{Store} \to [[\pi]]$.
  - $[[\{i{:}\theta_i\}_{i\in I}]] = \{i{:}[[\theta_i]]\}_{i\in I}$

Summary (Typing Rules I)

- U-abstraction definition and invocation:
  - $$\frac{\pi \vdash \mathsf{U}{:}\theta}{\pi \vdash \textbf{define } \mathsf{I}{=}\mathsf{U}{:}\ \{\mathsf{I}{:}\theta\}\mathit{dec}}$$
  - $\pi \vdash \textbf{invoke } \mathsf{I}{:}\ \theta \quad$ if $(\mathsf{I}{:}\theta) \in \pi$
  - $$\frac{\pi \dot{\cup} \{\mathsf{I}{:}\theta\} \vdash \mathsf{U}{:}\theta}{\pi \vdash \textbf{rec-define } \mathsf{I}{=}\mathsf{U}{:}\ \{\mathsf{I}{:}\theta\}\mathit{dec}}$$
- Program:
  - $$\frac{\emptyset \vdash \mathsf{D}{:}\ \pi\mathit{dec} \quad \pi \vdash \mathsf{C}{:}\ \mathit{comm}}{\vdash \mathsf{D} \textbf{ in } \mathsf{C}{:}\ \mathit{comm}}$$
- Type-Structure
  - $\pi \vdash \textbf{newint}{:}\mathit{intloc\ class}$
  - $$\frac{\pi \vdash \mathsf{D}{:}\ \pi_1\mathit{dec}}{\pi \vdash \textbf{record } \mathsf{D} \textbf{ end}{:}\ \pi_1\mathit{class}}$$
  - $$\frac{\pi \vdash \mathsf{X}{:}\delta\mathit{class}}{\pi \vdash \mathsf{X}{:}\delta\mathit{class}}$$

Summary (Typing Rules II)

- Declaration:
  - $\dfrac{\pi \vdash \mathsf{D}_1\text{: } \pi_1 \textit{dec} \quad \pi \vdash \mathsf{D}_2\text{: } \pi_2 \textit{dec}}{\pi \vdash \mathsf{D}_1,\mathsf{D}_2\text{: } (\pi_1 \dot{\cup} \ \pi_2)\textit{dec}}$
  - $\dfrac{\pi \vdash \mathsf{D}_1\text{: } \pi_1 \textit{dec} \quad \pi \dot{\cup}\, \pi \vdash \mathsf{D}_2\text{: } \pi_2 \textit{dec}}{\pi \vdash \mathsf{D}_1;\mathsf{D}_2\text{: } (\pi_1 \dot{\cup} \ \pi_2)\textit{dec}}$
  - $\dfrac{\pi \vdash \mathsf{N}\text{: } \tau \textit{int}}{\pi \vdash \textbf{const } \mathsf{I}{=}\mathsf{N}\text{: } \{\mathsf{I}{:}\textit{int}\}\textit{dec}}$
  - $\dfrac{\pi \vdash \mathsf{E}\text{: } \tau \textit{exp}}{\pi \vdash \textbf{fun } \mathsf{I}{=}\mathsf{E}\text{: } \{\mathsf{I}{:}\tau \textit{exp}\}\textit{dec}}$
  - $\dfrac{\pi \vdash \mathsf{C}\text{: } \tau \textit{comm}}{\pi \vdash \textbf{proc } \mathsf{I}{=}\mathsf{C}\text{: } \{\mathsf{I}{:}\textit{comm}\}\textit{dec}}$
  - $\dfrac{\pi \vdash \mathsf{T}\text{: } \delta \textit{class}}{\pi \vdash \textbf{var } \mathsf{I}{:}\mathsf{T}\text{: } \{\mathsf{I}{:}\delta\}\textit{dec}}$
  - $\dfrac{\pi \vdash \mathsf{T}\text{: } \delta \textit{clas}}{\pi \vdash \textbf{class } \mathsf{I}{=}\mathsf{T}\text{: } \{\mathsf{I}{:}\delta \textit{class}\}\textit{dec}}$

Summary (Typing Rules III)

- Declaration
  - $$\frac{\pi \vdash \mathsf{D}\colon \pi_1 \mathit{dec}}{\pi \vdash \textbf{module } \mathsf{I}{=}\{\mathsf{D}\}\colon \{\mathsf{I}{:}\pi_1\}\mathit{dec}}$$
  - $$\frac{\pi \vdash \mathsf{X}\colon \pi_1 \mathit{dec}}{\pi \vdash \textbf{import } \mathsf{X}\colon \{\mathsf{I}{:}\pi_1\}\mathit{dec}}$$
- Command
  - $$\frac{\pi \vdash \mathsf{L}\colon \mathit{intloc} \quad \pi \vdash \mathsf{E}\colon \mathit{intexp}}{\pi \vdash \mathsf{L}{:=}\mathsf{E}\colon \mathit{comm}}$$
  - $$\frac{\pi \vdash \mathsf{C}_1\colon \mathit{comm} \quad \pi \vdash \mathsf{C}_2\colon \mathit{comm}}{\pi \vdash \mathsf{C}_1;\mathsf{C}_2\colon \mathit{comm}}$$
  - $$\frac{\pi \vdash \mathsf{E}\colon \mathit{boolexp} \quad \begin{array}{l}\pi \vdash \mathsf{C}_1\colon \mathit{comm}\\ \pi \vdash \mathsf{C}_2\colon \mathit{comm}\end{array}}{\pi \vdash \textbf{if } \mathsf{E} \textbf{ then } \mathsf{C}_1 \textbf{ else } \mathsf{C}_2 \textbf{ fi}\colon \mathit{comm}}$$
  - $$\frac{\pi \vdash \mathsf{E}\colon \mathit{boolexp} \quad \pi \vdash \mathsf{C}\colon \mathit{comm}}{\pi \vdash \textbf{while } \mathsf{E} \textbf{ do } \mathsf{C} \textbf{ od}\colon \mathit{comm}}$$
  - $\pi \vdash \textbf{skip}\colon \mathit{comm}$
  - $$\frac{\pi \vdash X\colon \mathit{comm}}{\pi \vdash \textbf{call } X\colon \mathit{comm}}$$

Summary (Typing Rules IV)

- Expression
  - $\dfrac{\pi \vdash \mathsf{N}\colon \textit{int}}{\pi \vdash \mathsf{N}\colon \textit{intexp}}$
  - $\dfrac{\pi \vdash \mathsf{L}\colon \textit{intloc}}{\pi \vdash @\mathsf{L}\colon \textit{intexp}}$
  - $\dfrac{\pi \vdash \mathsf{E}_1\colon\ \textit{intexp} \quad \pi \vdash \mathsf{E}_2\colon\ \textit{intexp}}{\pi \vdash \mathsf{E}_1 + \mathsf{E}_2\colon\ \textit{intexp}}$
  - $\dfrac{\pi \vdash \mathsf{E}_1\colon\ \textit{intexp} \quad \pi \vdash \mathsf{E}_2\colon\ \textit{intexp}}{\pi \vdash \mathsf{E}_1 = \mathsf{E}_2\colon\ \textit{boolexp}}$
  - $\dfrac{\pi \vdash \mathsf{E}\colon \textit{boolexp}}{\pi \vdash \neg\mathsf{E}\colon \textit{boolexp}}$
  - $\pi \vdash \mathsf{I}\colon \tau\textit{exp}$, if $(\mathsf{I}{:}\tau\textit{exp}) \in \pi$
  - $\dfrac{\pi \vdash \mathsf{X}\colon \tau\textit{exp}}{\pi \vdash \mathsf{X}\colon \tau\textit{exp}}$

Summary (Typing Rules V)

- Variable:
  - $$\frac{\pi \vdash \text{X: } \mathit{intloc}}{\pi \vdash \text{X: } \mathit{intloc}}$$
- Numeral
  - $\pi \vdash n\text{: } \mathit{int}$, for $n \in \mathit{Int}$
  - $$\frac{\pi \vdash \text{X: } \mathit{int}}{\pi \vdash \text{X: } \mathit{int}}$$
- Identifier-expr:
  - $\pi \vdash \text{I:}\theta$, if $(\text{I:}\theta) \in \pi$
  - $$\frac{\pi \vdash \text{X:}\pi_1}{\pi \vdash \text{X.I: } \theta}$$, if $(\text{I:}\theta) \in \pi_1$

Summary (Semantics I)

- Lazy evaluation semantics of abstractions:
  - $[[\pi \vdash$ **define** I=U: $\{$I:$\theta\}$*dec*$]]e\ s = \{$I$=f\}$,
    where $f s' = [[\pi \vdash$ U:$\theta]]e\ s'$.
  - $[[\pi \vdash$ **invoke** I:$\theta]]e\ s = f s$,
    where (I$=f$) $\in e$.
- Recursively defined abstractions (example)
  - $[[\pi \vdash$ **rec-proc** I=C: $\{$I:*comm*$\}$*dec*$]]e\ s =$ ($\{$I$=r\}$, s)
    where $r(s') = [[\pi\ \dot\cup\ \{$I:*comm*$\} \vdash$ C:*comm*$]]$ (e $\cup\{$I$=r\}$)s'.
- Soundness of typing rules:
  - $[[\pi \vdash$ U:$\theta]] \in$ *Env*$_\pi \to [[\theta]]$.
- Program
  - $[[$D **in** C: *comm*$]]s = [[\pi_1 \vdash$ C: *comm*$]]e_1\ s$,
    where $(e_1, s_1) = [[\emptyset \vdash$ D: π_1*dec*$]]\emptyset\ s$

Summary (Semantics II)

- Declaration
  - $[[\pi \vdash$ **define** I=U: $\{$I:$\theta\}$*dec*$]]e\ s = (\{$I$=f\}, s)$
    where $f\ s' = [[\pi \vdash$ U:$\theta]]e\ s'$ and U $\in\{$E, T, C$\}$
  - $[[\pi \vdash$ D$_1$;D$_2$: $(\pi_1 \dot\cup\ \pi_2)$*dec*$]]e\ s =$
    $(e_1 \cup e_2, s_2)$
    where $(e_1, s_1) = [[\pi \vdash$ D$_1$: π_1*dec*$]]e\ s$
    and $(e_2, s_2) = [[\pi \dot\cup\ \pi_1 \vdash$ D$_2$: π_2*dec*$]]\ (e \cup e_1)\ s_1$
  - $[[\pi \vdash$ **fun** I=E: $\{$I:τ*exp*$\}]]e\ s = \{$I$=f\}$
    where $f(s') = [[\pi \vdash$ E: τ*exp*$]]e\ s'$
  - $[[\pi \vdash$ **const** I=N: $\{$I:*int*$\}]]e\ s =$
    $\{$I$=[[\pi \vdash$ N: *int*$]]e\ s\}$.
  - $[[\pi \vdash$ **proc** I=C: $\{$I:*comm*$\}$*dec*$]]e\ s = \{$I$=r\}$,
    where $r(s') = [[\pi \vdash$ C: *comm*$]]e\ s'$.
  - $[[\pi \vdash$**var** I:T: $\{$I:$\delta\}$*dec*$]]e\ s = (\{$I$=v\}, s')$
    where $(v, s') = [[\pi \vdash$ T: δ*class*$]]e\ s$.
  - $[[\pi \vdash$**class** I=T: $\{$I:δ*class*$\}$*dec*$]]e\ s = \{$I$=r\}$,
    where $r(s') = [[\pi \vdash$ T: δ*class*$]]e\ s'$.

Summary (Semantics III)

- Declaration
  - $[[\pi \vdash$ **module** I=$\{$D$\}$: $\{$I:$\pi_1\}$*dec*$]]e\ s =$
    $(\{$I=$e_1\}$, $s_1)$
    where $(e_1, s_1) = [[\pi \vdash$ D: π_1*dec*$]]e\ s$.
  - $[[\pi \vdash$ **import** I: π_1*dec*$]]e\ s = (e_1, s)$
    where $e_1 = [[\pi \vdash$X: $\pi_1]]e$
- Type-structure
  - $[[\pi \vdash$ **newint**: *intloc class*$]]e\ s =$ *allocate*(s)
  - $[[\pi \vdash$**record** D **end**: π*class*$]]e\ s =$
    $[[\pi \vdash$ D:π*dec*$]]e\ s$
  - $[[\pi \vdash$ X: δ*class*$]]e\ s = p(s)$
    where $p = [[\pi \vdash$ X: δ*class*$]]e$

Summary (Semantics IV)

- Command
  - $[[\pi \vdash \text{L:=E: } \textit{comm}]]e\ s =$
    update$([[\pi \vdash \text{L: } \textit{intloc}]]e,$
    $[[\pi \vdash \text{E: } \textit{intexp}]]e\ s,\ s)$
  - $[[\pi \vdash \text{C}_1\text{;C}_2\text{: } \textit{comm}]]e\ s = [[\pi \vdash \text{C}_2\text{: } \textit{comm}]]e$
    $([[\pi \vdash \text{C}_1\text{: } \textit{comm}]]e\ s)$
  - $[[\pi \vdash \textbf{if}\ \text{E}\ \textbf{then}\ \text{C}_1\ \textbf{else}\ \text{C}_2\ \textbf{fi}\text{: } \textit{comm}]]e\ s =$
    if$([[\pi \vdash \text{E: } \textit{boolexp}]]e\ s,$
    $[[\pi \vdash \text{C}_1\text{: } \textit{comm}]]e\ s,$
    $[[\pi \vdash \text{C}_2\text{: } \textit{comm}]]e\ s)$
  - $[[\pi \vdash \textbf{while}\ \text{E}\ \textbf{do}\ \text{C}\ \textbf{od}\text{: } \textit{comm}]]e\ s = w(s)$
    where $w(s) =$
    if$([[\pi \vdash \text{E: } \textit{boolexp}]]e\ s,$
    $w([[\pi \vdash \text{C: } \textit{comm}]]e\ s),\ s)$
  - $[[\pi \vdash \textbf{skip}\text{: } \textit{comm}]]e\ s = s$
  - $[[\pi \vdash \textbf{call}\ \text{X: } \textit{comm}]]e\ s = r(s)$
    where $r = [[\pi \vdash \text{X: } \textit{comm}]]e$

Summary (Semantics V)

- Expression
  - $[\![\pi \vdash \mathsf{N}\text{: } \textit{intexp}]\!] e\ s = [\![\pi \vdash \mathsf{N}\text{: } \textit{int}]\!] e$
  - $[\![\pi \vdash @\mathsf{L}\text{: } \textit{intexp}]\!] e\ s =$
    $\textit{lookup}([\![\pi \vdash \mathsf{L}\text{: } \textit{intloc}]\!] e,\ s)$
  - $[\![\pi \vdash \mathsf{E}_1 + \mathsf{E}_2\text{: } \textit{intexp}]\!] e\ s = \textit{plus}($
    $[\![\pi \vdash \mathsf{E}_1\text{: } \textit{intexp}]\!] e\ s,\ [\![\pi \vdash \mathsf{E}_2\text{: } \textit{intexp}]\!] e\ s)$
  - $[\![\pi \vdash \mathsf{E}_1 = \mathsf{E}_2\text{: } \textit{boolexp}]\!] e\ s = \textit{equal}($
    $[\![\pi \vdash \mathsf{E}_1\text{: } \textit{intexp}]\!] e\ s,\ [\![\pi \vdash \mathsf{E}_2\text{: } \textit{intexp}]\!] e\ s)$
  - $[\![\pi \vdash \neg\mathsf{E}\text{: } \textit{boolexp}]\!] e\ s =$
    $\textit{not}([\![\pi \vdash \mathsf{E}\text{: } \textit{boolexp}]\!] e\ s)$
  - $[\![\pi \vdash \mathsf{X}\text{: } \tau\textit{exp}]\!] e\ s = f(s)$
    where $f = [\![\pi \vdash \mathsf{X}\text{: } \tau\textit{expm}]\!] e$

Summary (Semantics VI)

- Variable
  - $[[\pi \vdash \text{X: } \textit{intloc}]]e = l$
    where $l = [[\pi \vdash \text{X: } \textit{intloc}]]e$
- Numeral
  - $[[\pi \vdash n\text{: } \textit{int}]]e = n$
  - $[[\pi \vdash \text{X: } \textit{int}]]e = v$
    where $v = [[\pi \vdash \text{X: } \textit{int}]]e$
- Identifier-expr
  - $[[\pi \vdash \text{I: } \theta]]e = v$, where $(\text{I}=v) \in e$
  - $[[\pi \vdash \text{X.I: } \theta]]e = v$,
    where $r = [[\pi \vdash \text{X: } \pi_1]]e$ and $(\text{I}=v) \in r$