# The Abstraction Principle 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

# 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: $\pi$*dec*]]$\emptyset\ s$

# Type-Structures

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

# 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.
  - $[[\pi \vdash \mathsf{T}{:}\delta\,\mathit{class}]] \in \mathit{Env}_\pi \to \mathit{Store} \to ([[\delta]] \times \mathit{Store})$
- Range of type attributes:
  - $\delta ::= \mathit{intloc} \mid \pi$
  - $\pi ::= \{i{:}\tau_i\}_{i \in I}$, where $I \subseteq$ Identifier
  - $\theta ::= \tau \mid \tau\,\mathit{exp} \mid \mathit{comm} \mid \delta \mid \delta\,\mathit{class} \mid \pi\,\mathit{dec}$
  - $\tau ::= \mathit{int} \mid \mathit{bool}$
- $[[\delta\,\mathit{class}]] = \mathit{Store} \to ([[\delta]] \times \mathit{Store})$.

# 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: $\pi_1$*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
    $(e_1,\ s_1)$ = [[$\pi \vdash$ $D_1$: $\pi_1$*dec*]]$e\ s$
    $(e_2,\ s_2)$ = [[$\pi \dot\cup\ \pi_1 \vdash$ $D_2$: $\pi_2$*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: $\delta$*class*]]$e\ s$

# Semantics of Type Structures II

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

# 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

- $$\frac{\pi \vdash \mathsf{D}\text{: } \pi_1 dec}{\textbf{module } \mathsf{I}=\{\mathsf{D}\}\text{: } \{\mathsf{I}\text{:}\pi_1 dec\}dec}$$
- $\pi \vdash \textbf{import } \mathsf{I}\text{: } \pi_1 dec$ if $(\mathsf{I}\text{:}\pi_1 dec) \in \pi$
- $[[\pi \vdash \textbf{module } \mathsf{I}=\{\mathsf{D}\}\text{: } \{\mathsf{I}\text{:}\pi_1 dec\}dec]]e\ s =$
  $(\{\mathsf{I}=f\},\ s)$
  where $f(s') = [[\pi \vdash \mathsf{D}\text{: } \pi_1 dec]]e\ s'$
- $[[\pi \vdash \textbf{import } \mathsf{I}\text{: } \pi_1 dec]]e\ s = f(s)$
  where $(\mathsf{I}=f) \in e$

- 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:$\pi_1 dec\}dec]]e\ s =$
  $(\{$I=$e_1\}, s_1)$
  where $(e_1, s_1) = [[\pi \vdash$ D: $\pi_1 dec]]e\ s$
- $[[\pi \vdash$ **import** I: $\pi_1 dec]]e\ s = (e_1, s)$
  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 $\theta$.
  - $\pi \vdash$ U: $\theta$
  - $\pi \vdash$ **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
  - $\theta ::= \tau \mid \tau\,exp \mid comm \mid \delta \mid \delta\,class \mid \pi\,dec$
  - $\tau ::= int \mid bool$
  - $\delta ::= intloc \mid \pi$
  - $\pi ::= \{i{:}\tau_i\}_{i \in I}$, where $I \subseteq$ Identifier
- Meaning of type attributes:
  - $[[int]] = Int$.
  - $[[bool]] = Bool$.
  - $[[\tau\,exp]] = Store \to [[\tau]]$.
  - $[[comm]] = Store \to Store_\perp$.
  - $[[intloc]] = Location$.
  - $[[\delta\,class]] = Store \to ([[\delta]] \times Store)$.
  - $[[\pi\,dec]] = Store \to [[\pi]]$.
  - $[[\{i{:}\theta_i\}_{i \in I}]] = \{i{:}[[\theta_i]]\}_{i \in I}$

# Summary (Typing Rules I)

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

# Summary (Typing Rules II)

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

# Summary (Typing Rules III)

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

# Summary (Typing Rules IV)

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

# Summary (Typing Rules V)

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

# Summary (Semantics I)

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

# 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 $(e_1, s_1) = [[\pi \vdash$ $D_1$: $\pi_1$*dec*$]]e\ s$
    and $(e_2, s_2) = [[\pi \dot\cup\ \pi_1 \vdash$ $D_2$: $\pi_2$*dec*$]]\ (e \cup e_1)\ s_1$
  - $[[\pi \vdash$ **fun** I=E: $\{$I:$\tau$*exp*$\}]]e\ s = \{$I=$f\}$
    where $f(s') = [[\pi \vdash$ E: $\tau$*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: $\delta$*class*$]]e\ s$.
  - $[[\pi \vdash$**class** I=T: $\{$I:$\delta$*class*$\}$*dec*$]]e\ s = \{$I=$r\}$,
    where $r(s') = [[\pi \vdash$ T: $\delta$*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: $\pi_1$*dec*]]$e\ s$.
  - [[$\pi \vdash$ **import** I: $\pi_1$*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**: $\pi$*class*]]$e\ s =$
    [[$\pi \vdash$ D:$\pi$*dec*]]$e\ s$
  - [[$\pi \vdash$ X: $\delta$*class*]]$e\ s = p(s)$
    where $p =$ [[$\pi \vdash$ X: $\delta$*class*]]$e$

# Summary (Semantics IV)

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

# Summary (Semantics V)

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

# Summary (Semantics VI)

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