Records and Lambda Abstractions

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

Introduction

- Records and lambda abstractions.
  - Crucial components of structured, parameterized language.
  - Basis of natural "model" and "logic".
- The Desugared Programming Language
  - E $\in$ Everything
    N $\in$ Numeral
    L $\in$ Location
    E ::= E_1:=E_2 | E_1;E_2 | **if** E_1 **then** E_2 **else** E_3 **fi** | **skip** | **while** E_1 **do** E_2 **od** | E_1+E_2 | E_1=E_2 | $\neg$E | @E | N | L
  - Only one syntax rule
  - Typing annotations preserve structure:
- Reasons for collapsing language:
  - Same structuring device for whole language instead of one instance for each syntax domain.
  - Additional language extensions more convenient.

Typing Rules

$$\frac{E_1\text{: } \mathit{intloc} \quad E_2\text{: } \mathit{intexp}}{E_1 := E_2\text{: } \mathit{comm}} \qquad \frac{E_1\text{: } \mathit{comm} \quad E_2\text{: } \mathit{comm}}{E_1 ; E_2\text{: } \mathit{comm}}$$

$$\frac{E_1\text{: } \mathit{boolexp} \quad E_2\text{: } \mathit{comm} \quad E_3\text{: } \mathit{comm}}{\mathbf{if}\ E_1\ \mathbf{then}\ E_2\ \mathbf{else}\ E_3\ \mathbf{fi}\text{: } \mathit{comm}}$$

$$\mathbf{skip}\text{: } \mathit{comm} \qquad \frac{E_1\text{: } \mathit{boolexp} \quad E_2\text{: } \mathit{comm}}{\mathbf{while}\ E_1\ \mathbf{do}\ E_2\ \mathbf{od}\text{: } \mathit{comm}}$$

$$\frac{E_1\text{: } \mathit{intexp} \quad E_2\text{: } \mathit{intexp}}{E_1 + E_2\text{: } \mathit{intexp}} \qquad \frac{E_1\text{: } \tau\mathit{exp} \quad E_2\text{: } \tau\mathit{exp}}{E_1 = E_2\text{: } \mathit{boolexp}}$$

$$\frac{E\text{: } \mathit{boolexp}}{\neg E\text{: } \mathit{boolexp}}$$

$$\frac{E\text{: } \mathit{intloc}}{@E\text{: } \mathit{intexp}} \qquad N\text{: } \mathit{intexp} \qquad L\text{: } \mathit{intloc}$$

Record Introduction

- Abstraction/qualification principle:
  - E ::= ... | I=E | E_1,E_2 | *with* E_1 *do* E_2 | I
  - $\dfrac{\pi \vdash \text{E: } \theta}{\pi \vdash \text{I=E: } \{\text{I:}\theta\}}$

    $\dfrac{\pi \vdash \text{E}_1\text{: } \pi_1 \quad \pi \vdash \text{E}_2\text{: } \pi_2}{\pi \vdash \text{E}_1\text{,E}_2\text{: } \pi_1 \ \dot{\cup} \ \pi_2}$

    $\dfrac{\pi \vdash \text{E}_1\text{: } \pi_1 \quad \pi \ \overline{\cup} \ \pi_1 \vdash \text{E}_2\text{: } \theta}{\pi \vdash \textit{with } \text{E}_1 \textit{ do } \text{E}_2\text{: } \theta}$

    $\pi \vdash \text{I: } \theta \quad \text{if } (\text{I:}\theta) \in \pi$
- *Record* I=E
  - Record union E_1,E_2.
  - Local record definitions.
  - Record field reference I.
- Enrichement of type system:
  - Type attribute for records.
  - $\{i{:}\theta_i\}_{i \in I}$
  - Type assignment attributes introduced.

Example

with A=loc_1 *do*
  with $M = (P = A{:=}@A+1)$ *do*
    with M, F=0 *do*
      $A{:=}F$; P

begin alias A=loc_1 **in**
  begin module $M = \{$**proc** $P = A{:=}@A+1\}$ **in**
    begin import M, **fun** F=0 **in**
      $A{:=}F$; **call** P
    end
  end
end

Lambda Abstraction Introduction

- Parameterization principle:
  - E ::= ... | λI:θ. E | E_1 E_2 | I
  - $\dfrac{\pi \,\overline{\cup}\, \{\mathsf{I}:\theta_1\} \vdash \mathsf{E}:\theta_2}{\pi \vdash \lambda \mathsf{I}:\theta_1.\ \mathsf{E}:\ \theta_1 \to \theta_2}$

    $\dfrac{\pi \vdash \mathsf{E}_1:\ \theta_1 \to \theta_2 \quad \pi \vdash \mathsf{E}_2:\ \theta_1}{\pi \vdash \mathsf{E}_1\ \mathsf{E}_2:\ \theta_2}$

    $\pi \vdash \mathsf{I}:\ \theta \quad \text{if } (\mathsf{I}:\theta) \in \pi$
- *Lambda abstraction* $\lambda \mathsf{I}_2:\theta$. E
  - Abstraction invocation E_1 E_2
  - Parameter reference I.
- Correspondence principle
  - Semantics of definition binding and semantics of parameter binding is the same.
- Enrichement of type system
  - New attribute $\theta_1 \to \theta_2$

Example

$\mathit{with}\ M = \lambda X{:}\mathit{intloc}.$
$\quad (A{=}\mathit{loc}_1,\ P{=}\lambda Y{:}\mathit{intexp}.X{:=}Y)\ \mathit{do}$
$\quad \mathit{with}\ M\ \mathit{loc}_1\ \mathit{do}$
$\quad\quad P\ (@A{+}1)$

begin module $M(X\!: \mathit{intloc}) =$
$\quad$ {**alias** $A{=}\mathit{loc}_1$, **proc** $P(Y\!: \mathit{intexp}) = X{:=}Y$}
in begin import $M(\mathit{loc}_1)$ **in**
$\quad$ **call** $P(@A{+}1)$
end end

Full Language

E ::= E_1:=E_2 | E_1;E_2 | **if** E_1 **then** E_2 **else** E_3 **fi**
| **skip** | **while** E_1 **do** E_2 **od** | E_1+E_2 | E_1=E_2
| ¬E | @E | N | L
| I=E |E_1,E_2 | *with* E_1 *do* E_2 | I | λI:θ. E | E_1 E_2

θ ::= τ*exp* | *comm* | *intloc* | π | $\theta_1 \rightarrow$ | θ_2
τ ::= *int* | *bool*
π ::= $\{j{:}\theta_j\}_{j \in J}$

$$\frac{\pi \vdash \mathsf{E}_1\text{: } \mathit{intloc} \quad \pi \vdash \mathsf{E}_2\text{: } \mathit{intexp}}{\pi \vdash \mathsf{E}_1\text{:=}\mathsf{E}_2\text{: } \mathit{comm}}$$

$$\frac{\pi \vdash \mathsf{E}_1\text{: } \mathit{comm} \quad \pi \vdash \mathsf{E}_2\text{: } \mathit{comm}}{\pi \vdash \mathsf{E}_1\text{;}\mathsf{E}_2\text{: } \mathit{comm}}$$

$$\frac{\pi \vdash \mathsf{E}_1\text{: } \mathit{boolexp} \quad \pi \vdash \mathsf{E}_2\text{: } \mathit{comm} \quad \pi \vdash \mathsf{E}_3\text{: } \mathit{comm}}{\pi \vdash \textbf{if } \mathsf{E}_1 \textbf{ then } \mathsf{E}_2 \textbf{ else } \mathsf{E}_3 \textbf{ fi}\text{: } \mathit{comm}}$$

$$\pi \vdash \textbf{skip}\text{: } \mathit{comm}$$

$$\frac{\pi \vdash \mathsf{E}_1\text{: } \mathit{boolexp} \quad \pi \vdash \mathsf{E}_2\text{: } \mathit{comm}}{\pi \vdash \textbf{while } \mathsf{E}_1 \textbf{ do } \mathsf{E}_2 \textbf{ od}\text{: } \mathit{comm}}$$

$$\frac{\pi \vdash \mathsf{E}_1\text{: } \mathit{intexp} \quad \pi \vdash \mathsf{E}_2\text{: } \mathit{intexp}}{\pi \vdash \mathsf{E}_1\text{+}\mathsf{E}_2\text{: } \mathit{intexp}}$$

Full Language (Contd)

$$\frac{\pi \vdash \mathsf{E}_1\!:\ \tau\mathit{exp} \quad \pi \vdash \mathsf{E}_2\!:\ \tau\mathit{exp}}{\pi \vdash \mathsf{E}_1{=}\mathsf{E}_2\!:\ \mathit{boolexp}}$$

$$\frac{\pi \vdash \mathsf{E}\!:\ \mathit{boolexp}}{\pi \vdash \neg\mathsf{E}\!:\ \mathit{boolexp}}$$

$$\frac{\pi \vdash \mathsf{E}\!:\ \mathit{intloc}}{\pi \vdash @\mathsf{E}\!:\ \mathit{intexp}} \qquad \pi \vdash \mathsf{N}\!:\ \mathit{intexp}$$

$$\pi \vdash \mathsf{L}\!:\ \mathit{intloc}$$

$$\frac{\pi \vdash \mathsf{E}\!:\ \theta}{\pi \vdash \mathsf{I}{=}\mathsf{E}\!:\ \{\mathsf{I}{:}\theta\}}$$

$$\frac{\pi \vdash \mathsf{E}_1\!:\ \pi_1 \quad \pi \vdash \mathsf{E}_2\!:\ \pi_2}{\pi \vdash \mathsf{E}_1,\mathsf{E}_2\!:\ \pi_1 \ \dot{\cup}\ \ \pi_2}$$

$$\frac{\pi \vdash \mathsf{E}_1\!:\ \pi_1 \quad \pi \mathbin{\overline{\cup}} \pi_1 \vdash \mathsf{E}_2\!:\ \theta}{\pi \vdash \textit{with}\ \mathsf{E}_1\ \textit{do}\ \mathsf{E}_2\!:\ \theta}$$

$$\pi \vdash \mathsf{I}\!:\ \theta \qquad \text{if } (\mathsf{I}{:}\theta) \in \pi$$

$$\frac{\pi \mathbin{\overline{\cup}} \{\mathsf{I}{:}\theta_1\} \vdash \mathsf{E}{:}\theta_2}{\pi \vdash \lambda\mathsf{I}{:}\theta_1.\ \mathsf{E}\!:\ \theta_1 \rightarrow \theta_2}$$

$$\frac{\pi \vdash \mathsf{E}_1\!:\ \theta_1 \rightarrow \theta_2 \quad \pi \vdash \mathsf{E}_2\!:\ \theta_1}{\pi \vdash \mathsf{E}_1\ \mathsf{E}_2\!:\ \theta_2}$$

$$\pi \vdash \mathsf{I}\!:\ \theta \qquad \text{if } (\mathsf{I}{:}\theta) \in \pi$$

Higher-Order Programming Languages

- Records are values in own right:
  - **alias** A=loc_1 in
    call { **fun** F=@A+1, **proc** P=A:=F }.P
  - $R.I \equiv$ **with** R **do** I
- Parameterization in arbitrary places:
  - A:=((λX:*intexp*. X+1)@A)+2
  - **proc** $R(X$: *intloc*$)$ = λY:*intexp*. X:=Y
- *Higher-order languages*
  - Lambda abstractions and records are full-fledged values.
  - *First-order* languages do not allow this free use.

Semantics of Records and Lambda Abstractions

- Core part of language:
  - $[[\pi \vdash \text{L:=E: } \mathit{comm}]]e\ s =$
    $\mathit{update}([[\pi \vdash \text{L: } \mathit{intloc}]]e,$
    $[[\pi \vdash \text{E: } \mathit{intexp}]]e\ s,\ s)$
  - $[[\pi \vdash \text{E}_1\text{;E}_2\text{: } \mathit{comm}]]e\ s =$
    $[[\pi \vdash \text{E}_2\text{: } \mathit{comm}]]e\ ([[\pi \vdash \text{E}_1\text{: } \mathit{comm}]]e\ s)$
  - $[[\pi \vdash \text{E}_1\text{+E}_2\text{: } \mathit{intexp}]]e\ s =$
    $\mathit{plus}([[\pi \vdash \text{E}_1\text{: } \mathit{intexp}]]e\ s,$
    $[[\pi \vdash \text{E}_2\text{: } \mathit{intexp}]]e\ s)$
  - $[[\pi \vdash \text{N: } \mathit{intexp}]]e\ s = [[\text{N: } \mathit{int}]]e$
  - $[[\pi \vdash \mathit{loc}_i\text{: } \mathit{intloc}]]e = \mathit{loc}_i$
- Type semantics:
  - $[[\mathit{comm}]] = \mathit{Store} \to \mathit{Store}_\perp$
  - $[[\tau\,\mathit{exp}]] = \mathit{Store} \to [[\tau]]$

Addition of environment does not change meaning of core language programs.

Lazy Evaluation Semantics

- Lazily evaluated abstractions/records:
  - $[[\pi \vdash \mathsf{l{=}E}\text{: } \{\mathsf{l}{:}\theta\}]]e = \{\mathsf{l} = [[\pi \vdash \mathsf{E}\text{: } \theta]]e\}$
  - $[[\pi \vdash \mathsf{E}_1,\mathsf{E}_2\text{: } \pi_1 \dot\cup\ \pi_2]]e =$
    $([[\pi \vdash \mathsf{E}_1\text{: } \pi_1]]e) \cup ([[\pi \vdash \mathsf{E}_2\text{: } \pi_2]]e)$
  - $[[\pi \vdash$ *with* E_1 *do* E_2: $\theta]]e =$
    $[[\pi \mathbin{\overline{\cup}} \pi_1 \vdash \mathsf{E}_2\text{: } \theta]](e \mathbin{\overline{\cup}} [[\pi \vdash \mathsf{E}_1\text{: } \pi_1]]e)$
  - $[[\pi \vdash \mathsf{l}\text{: } \theta]]e = v$, where $(\mathsf{l}{=}v) \in e$
  - $[[\pi \vdash \lambda\mathsf{l}{:}\theta_1.\ \mathsf{E}\text{: } \theta_1 \rightarrow \theta_2]]e = f$
    where $f\ u = [[\pi \mathbin{\overline{\cup}} \{\mathsf{l}{:}\theta_1\} \vdash \mathsf{E}\text{: } \theta_2]](e \mathbin{\overline{\cup}} \{\mathsf{l}{=}u\})$
  - $[[\pi \vdash \mathsf{E}_1\ \mathsf{E}_2\text{: } \theta_2]]e =$
    $([[\pi \vdash \mathsf{E}_1\text{: } \theta_1 \rightarrow \theta_2]]e)([[\pi \vdash \mathsf{E}_2\text{: } \theta_1]]e)$
- Type semantics:
  - $[[\{i{:}\theta_i\}_{i\in I}]] = \{\ i{:}[[\theta_i]]\ \}_{i\in I}$
  - $[[\theta_1 \rightarrow \theta_2]] = [[\theta_1]] \rightarrow [[\theta_2]]$

Lazy Evaluation Semantics

- *No* store argument mentioned:
  - Lazy evaluation never suppplies a store to a record binding, a lambda abstraction, or the argument of a lambda abstraction.
  - Records and lambda abstractions constitute a "functional sublanguage" overlaid over imperative core.
- Typing rules are *sound*.
- Copy rules:
  - *with* $(i{=}\mathsf{E}_i)_{i \in I}$ *do* $\mathsf{E} \Rightarrow [\mathsf{E}_i/i]_{i \in I} E$
  - $(\lambda \mathsf{I}{:}\theta.\ \mathsf{E}_1)\mathsf{E}_2 \Rightarrow [\mathsf{E}_2/\mathsf{I}]\mathsf{E}_1$
- Calculation of program semantics:
  - Apply copy rules to make all bindings.
  - Use semantics of core language.

Variable Declarations

- New block construct for variables:
  - Allocate location, bind it to identifier V and let V be visible to command E.
  - $$\frac{\pi \cup \{\ \text{V: } \textit{intloc}\ \} \vdash \text{E: } \textit{comm}}{\pi \vdash \textit{new}\ \text{V}\ \textit{in}\ \text{E: } \textit{comm}}$$
  - $[[\pi \vdash \textit{new}\ \text{V}\ \textit{in}\ E\text{: } \textit{comm}]]e\ s =$
    free (*size-of* s) s_2
    where $(l,\ s_1) =$ *allocate* s
    and $s_2 = [[\pi \cup \{\ \text{V: } \textit{intloc}\ \} \vdash \text{E: } \textit{comm}]]$
    $(e \cup \{\ \text{V}=l\ \})s_1$
- Lazy evaluation semantics is preserved.

Eager Evaluation Semantics

- *Without* variable declarations:
  - $[[\pi \vdash \mathsf{I{=}E}\colon \{\mathsf{I}{:}\theta\}]]e\ s = \{\mathsf{I} = [[\pi \vdash \mathsf{E}\colon \theta]]e\ s\}$
  - $[[\pi \vdash \mathsf{E}_1,\mathsf{E}_2\colon \pi_1 \dot{\cup}\ \pi_2]]e\ s =$
    $([[\pi \vdash \mathsf{E}_1\colon \pi_1]]e\ s) \cup ([[\pi \vdash \mathsf{E}_2\colon \pi_2]]e\ s)$
  - $[[\pi \vdash$ *with* E_1 *do* $E_2\colon \theta]]e\ s =$
    $[[\pi \mathbin{\overline{\cup}} \pi_1 \vdash \mathsf{E}_2\colon \theta]](e \mathbin{\overline{\cup}} ([[\pi \vdash \mathsf{E}_1\colon \pi_1]]e\ s))\ s$
  - $[[\pi \vdash \mathsf{I}\colon \theta]]e\ s = u$, where $(\mathsf{I}{=}u) \in e$
  - $[[\pi \vdash \lambda\mathsf{I}{:}\theta_1.\ \mathsf{E}\colon \theta_1 \to \theta_2]]e\ s = f$
    where $f\ u\ s' =$
    $[[\pi \mathbin{\overline{\cup}} \{\mathsf{I}{:}\theta_1\} \vdash \mathsf{E}\colon \theta_2]](e \mathbin{\overline{\cup}} \{\mathsf{I}{=}u\})\ s'$
  - $[[\pi \vdash \mathsf{E}_1\ \mathsf{E}_2\colon \theta_2]]e\ s =$
    $([[\pi \vdash \mathsf{E}_1\colon \theta_1 \to \theta_2]]e)([[\pi \vdash \mathsf{E}_2\colon \theta_1]]e\ s)\ s$
- Type semantics:
  - $[[\{i{:}\theta_i\}_{i\in I}]] = (\{\ i{:}\mathit{eval}(\theta_i)\ \}_{i\in I})_\perp$
  - $[[\theta_1 \to \theta_2]] = (\mathit{eval}(\theta_1) \to [[\theta_2]])_\perp$
  - $[[\mathit{comm}]] = (\mathit{Store} \to \mathit{Store})_\perp$
  - $[[\tau\,\mathit{exp}]] = (\mathit{Store} \to [[\tau]])_\perp$

Eager Evaluation Semantics

- Type semantics (contd)
  - $\textit{eval}(\{i{:}\theta_i\}_{i \in I}) = \{\ i{:}\ \textit{eval}(\theta_i)\ \}_{i \in I}$
  - $\textit{eval}(\theta_1 \rightarrow \theta_2) = \textit{eval}(\theta_1) \rightarrow [[\theta_2]]$
  - $\textit{eval}(\textit{comm}) = \textit{Store}$
  - $\textit{eval}(\tau\,\textit{exp}) = [[\tau]]$
- Properties:
  - Use store argument to force evaluation of phrases that bind to identifiers.
  - Soundness of typing rules preserved.
  - Copy rule does *not* hold.
- Strictness:
  - $\{\ \mathsf{I} = \bot\ \} = \bot$
  - $f\ \bot = \bot$
- Non-termination arises in language!

Lazy and Eager Evaluation Combined

- Introduction of **newint**
- Type attributes:
  - $\theta ::= \tau \mid \tau\,\mathit{exp}$
  - $\tau ::= \mathit{int} \mid \mathit{bool} \mid \mathit{intloc} \mid \mathit{store} \mid \pi \mid \theta_1 \to \theta_2$
  - $\pi ::= \{i{:}\theta_i\}_{i \in I}$
- Evaluation:
  - τ is type of *evaluated phrase*
  - $\tau\,\mathit{exp}$ is type of *unevaluated phrase*
- Examples:
  - $\mathit{comm} \equiv \mathit{store}\ \mathit{exp}$
  - $\delta\mathit{class} \equiv \delta\mathit{exp}$
  - $\pi\mathit{dec} \equiv \pi\mathit{exp}$

Lazy and Eager Bindings

- Extended syntax:
  - E ::= E_1:=E_2 | . . . | **newint**
    | *lazy* I=E | *eager* I=E
    | E_1,E_2 | *with* E_1 *do* E_2 | I | λI:θ. E | E_1 E_2
- Examples:
  - **var** X: **newint**
    $\equiv$ *eager* X = **newint**
  - **class** K =
    record var Y: **newint**, **proc** P = **skip end**
    $\equiv$ *lazy* K =
    (*eager* Y = **newint**, *lazy* P = **skip**)
  - **module** M =
    { **var** Y: **newint**, **proc** P = **skip** }
    $\equiv$ *eager* K =
    (*eager* Y = **newint**, *lazy* P = **skip**)
- Lambda abstractions:
  - λI:τ*exp*. E takes unevaluated phrases.
  - λI:τ. E takes only evaluated phrases.

Typing Rules

- Basic Rules
  - $\pi \vdash$ **newint**: *intloc exp*
  - $$\frac{\pi \vdash \mathsf{E}\colon \theta}{\pi \vdash \textit{lazy}\ \mathsf{I}{=}\mathsf{E}\colon \{\ \mathsf{I}{:}\theta\}\textit{exp}}$$
  - $$\frac{\pi \vdash \mathsf{E}\colon \tau\textit{exp}}{\pi \vdash \textit{eager}\ \mathsf{I}{=}\mathsf{E}\colon \{\ \mathsf{I}{:}\tau\}\textit{exp}}$$
- *with* E_1 *do* E_2
  - $$\frac{\pi \vdash \mathsf{E}_1\colon\ \pi_1 \quad \pi \cup \pi_1 \vdash \mathsf{E}_2\colon\ \theta}{\pi \vdash \textit{with}\ \mathsf{E}_1\ \textit{do}\ \mathsf{E}_2\colon\ \theta}$$
  - $$\frac{\pi \vdash \mathsf{E}_1\colon\ \pi_1\textit{exp} \quad \pi \cup \pi_1 \vdash \mathsf{E}_2\colon\ \tau\textit{exp}}{\pi \vdash \textit{with}\ \mathsf{E}_1\ \textit{do}\ \mathsf{E}_2\colon\ \tau\textit{exp}}$$

    $\tau \in \{\ \pi_2,\ \textit{store}\ \}$
  - Second rule considers that evaluation of E_1 may cause side effects that must be managed by E_2.

Typing Rules

- $\mathsf{E}_1\ \mathsf{E}_2$
  - $$\frac{\pi \vdash \mathsf{E}_1\colon\ \theta_1 \rightarrow \theta_2 \quad \pi \vdash \mathsf{E}_2\colon\ \theta_1}{\pi \vdash \mathsf{E}_1\ \mathsf{E}_2\colon\ \theta_2}$$
  - $$\frac{\pi \vdash \mathsf{E}_1\colon\ \tau \rightarrow \theta \quad \pi \vdash \mathsf{E}_2\colon\ \tau\, \mathit{exp}}{\pi \vdash \mathsf{E}_1\ \mathsf{E}_2\colon\ \theta}$$
    $\tau \in \{\ \mathit{int}, \mathit{bool}\ \}$
  - Second rule forces evaluation of actual parameter.
- Various possibilities!

Denotational Semantics

$[[\pi \vdash$ **newint**: *intloc exp*$]]e\ s =$ *allocate* s
$[[\pi \vdash$ *lazy* I=E: $\{$ I:θ $\}$*exp*$]]e\ s =$
$(\{$ I $= [[\pi \vdash$ E:$\theta]]e$ $\}$, $s)$
$[[\pi \vdash$ *eager* I=E: $\{$ I:τ $\}$*exp*$]]e\ s = (\{$ I $= v$ $\}$, $s')$
where $(v, s') = [[\pi \vdash$ E:τ*exp*$]]e\ s$
$[[\pi \vdash$ I: τ*exp*$]]e\ s = v\ s$, where (I=v) $\in e$
$[[\pi \vdash$ I: $\tau]]e\ s = v$, where (I=v) $\in e$
$[[\pi \vdash$ *with* E_1 *do* E_2: $\theta]]e\ s =$
$[[\pi \uplus \pi_1 \vdash E_2\text{: } \theta]](e \uplus [[\pi \vdash E_1\text{: } \pi_1]]e)\ s$
$[[\pi \vdash$ *with* E_1 *do* E_2: π_2*exp*$]]e\ s =$
$[[\pi \uplus \pi_1 \vdash E_2$: π_2*exp*$]](e \uplus e_1)\ s_1$
where $(e_1, s_1) = [[\pi \vdash E_1$: π_1*exp*$]]e\ s$
$[[\pi \vdash$ *with* E_1 *do* E_2: *store exp*$]]e\ s =$
free (*size-of* s) s_2
where $(e_1, s_1) = [[\pi \vdash E_1$: π_1*exp*$]]e\ s$
and $s_2 = [[\pi \uplus \pi_1 \vdash E_2$: *store exp*$]](e \uplus e_1)\ s_1$

Denotational Semantics

$[\![\pi \vdash \mathsf{E}_1\ \mathsf{E}_2\colon \theta_2]\!] e\ s =$
$\quad ([\![\pi \vdash \mathsf{E}_1\colon \theta_1 \rightarrow \theta_2]\!] e)\ ([\![\pi \vdash \mathsf{E}_2\colon \theta_1]\!] e)\ s$
$[\![\pi \vdash \mathsf{E}_1\ \mathsf{E}_2\colon \theta]\!] e\ s =$
$\quad ([\![\pi \vdash \mathsf{E}_1\colon \tau \rightarrow \theta]\!] e)\ ([\![\pi \vdash \mathsf{E}_2\colon \tau\, \mathit{exp}]\!] e\ s)\ s$

$[\![\tau\, \mathit{exp}]\!] = \mathit{Store} \rightarrow [\![\tau]\!]$
$[\![\mathit{int}]\!] = \mathit{Int}_\perp$
$[\![\mathit{bool}]\!] = \mathit{Bool}_\perp$
$[\![\theta_1 \rightarrow \theta_2]\!] = (\mathit{proper}\ [\![\theta_1]\!] \rightarrow [\![\theta_2]\!])_\perp$
$[\![\{i{:}\theta_i\}_{i \in I}]\!] = (\{\ i{:}\mathit{proper}\ [\![\ \theta_i\ ]\!]\}_{i \in I})_\perp$
$[\![\mathit{intloc}]\!] = \mathit{Location}_\perp$
$[\![\mathit{store}]\!] = \mathit{Store}_\perp$
$\mathit{proper}\ D = D - \{\ \perp\}$

Lambda Abstractions Alone

- Language with naming and parameters.
  - Do not care about record structures, classes, modules.
  - Can use lambda abstractions as sole foundation.
- Translation technique:
  - **define** I(I_1:θ_1, I_2:θ_2, ..., I_n:θ_n) = U
    $\Rightarrow$ **define** I = ($\lambda\mathsf{I}_1$:θ_1. (($\lambda\mathsf{I}_2$:θ_2. ...($\lambda\mathsf{I}_n$:θ_n. U) ...)))
  - **begin define** I_1=U_1, **define** I_2=U_2, ..., **define** I_n=U_n **in** W **end**
    $\Rightarrow$ ((...((($\lambda\mathsf{I}_1$:θ_1. ($\lambda\mathsf{I}_2$:θ_2. ...($\lambda\mathsf{I}_n$:θ_n. W) ...)) U_1) U_2) ...) U_n)
  - **invoke** I(V_1, V_2, ..., V_n)
    $\Rightarrow$ ((...((I V_1) V_2) ...) V_n)

Orthogonality

- Records and lambda abstractions can be added *independently*.
  - Concepts are orthogonal to each other and to core language.
- *Orthogonality* of new construction:
  - Semantics of original constructs unchanged.
  - If new construction uses component phrases of original language, the semantics of new construction is defined uniformly with respect to the component phrases (no special cases).
- Consequences:
  - Typing rules and semantic equations can be added without affecting existing language.
  - Records, lambda abstractions, *new* V *in* E
- No orthogonality:
  - Revision of language required.
  - **newint**

Model of the Programming Language

- Are our constructions of records and lambda abstractions the “right” ones?
  - Justification by category theory.
  - *Categorical product*.
  - *Categorical exponentiation*.
- Definitions:
  - $\langle \mathsf{E}_1, \mathsf{E}_2 \rangle := (\mathit{fst}{=}E_1,\ \mathit{snd}{=}E_2)$
  - $e{\downarrow}1 :=$ *with* e *do fst*
  - $e{\downarrow}2 :=$ *with* e *do snd*
  - $(\mathsf{E}_1 \equiv \mathsf{E}_2) := [[\pi \vdash \mathsf{E}_1\!: \theta]]e = [[\pi \vdash \mathsf{E}_2\!: \theta]]e$,
    $e \in \mathit{Env}_\pi$

Categorical Product

- Categorical product $\theta_1 \times \theta_2$
  - $\langle \mathsf{E}_1, \mathsf{E}_2 \rangle \downarrow 1 \equiv \mathsf{E}_1$
  - $\langle \mathsf{E}_1, \mathsf{E}_2 \rangle \downarrow 2 \equiv \mathsf{E}_2$
  - $\langle \mathsf{E} \downarrow 1, \mathsf{E}_2 \downarrow 2 \rangle \equiv \mathsf{E}$, if $\pi \vdash \mathsf{E}$: $\theta_1 \times \theta_2$
- Theorem:
  - For the lazy evaluation semantics, $\theta_1 \times \theta_2 :=$ $\{$*fst*:θ_1, *snd*:$\theta_2\}$ is the categorical product.
- Theorem does *not* hold for eager semantics!
  - $\langle \perp, \mathsf{E} \rangle = \perp$
  - "Tensor product"

Categorical Exponentiation

- Categorical exponentiation $\theta_1 \Rightarrow \theta_2$
  - *apply*$\langle$(λI:θ_1. E), U$\rangle$ $\equiv$ *with* I=U *do* E
  - (λI:θ_1. *apply*$\langle$E, I$\rangle$) $\equiv$ E, if $\pi \vdash$ E: $\theta_1 \Rightarrow \theta_2$
- Theorem:
  - For the lazy evaluation semantics, $\theta_1 \Rightarrow \theta_2 := \theta_1 \to \theta_2$ is the categorical exponentiation.
  - *apply* = λI: $(\theta_1 \Rightarrow \theta_2)\times\theta_1$. (I$\downarrow$1)(I$\downarrow$2)
- Theorem does *not* hold for eager semantics!
  - (λX:*comm*. λY:*comm*. Y)*loop*
  - "Weak exponentiation"