COMP3610/6361
Principles of Programming Languages

Peter Höfner

Aug 16, 2023

Section 8

Data

Recap and Missing Steps

- simple while language
- with functions
- **but no data structures**

Products – Syntax

$$T ::= \ldots \mid T * T$$

$$E ::= \ldots \mid (E, E) \mid \textbf{fst}\ E \mid \textbf{snd}\ E$$

Products – Typing

(pair) $$\frac{\Gamma \vdash E_1 : T_1 \qquad \Gamma \vdash E_2 : T_2}{\Gamma \vdash (E_1, E_2) : T_1 * T_2}$$

(proj1) $$\frac{\Gamma \vdash E : T_1 * T_2}{\Gamma \vdash \textbf{fst}\ E : T_1}$$

(proj2) $$\frac{\Gamma \vdash E : T_1 * T_2}{\Gamma \vdash \textbf{snd}\ E : T_2}$$

Products – Semantics

Values

$v ::= \ldots \mid (v, v)$

SOS rules

(pair1) $$\frac{\langle E_1 \,,\, s\rangle \longrightarrow \langle E_1' \,,\, s'\rangle}{\langle (E_1, E_2) \,,\, s\rangle \longrightarrow \langle (E_1', E_2) \,,\, s'\rangle}$$

(pair2) $$\frac{\langle E_2 \,,\, s\rangle \longrightarrow \langle E_2' \,,\, s'\rangle}{\langle (v, E_2) \,,\, s\rangle \longrightarrow \langle (v, E_2') \,,\, s'\rangle}$$

(proj1) $\langle \textbf{fst}\ (v_1, v_2) \,,\, s\rangle \longrightarrow \langle v_1 \,,\, s\rangle$

(proj2) $\langle \textbf{snd}\ (v_1, v_2) \,,\, s\rangle \longrightarrow \langle v_2 \,,\, s\rangle$

(proj3) $$\frac{\langle E \,,\, s\rangle \longrightarrow \langle E' \,,\, s'\rangle}{\langle \textbf{fst}\ E \,,\, s\rangle \longrightarrow \langle \textbf{fst}\ E' \,,\, s'\rangle}$$

(proj4) $$\frac{\langle E \,,\, s\rangle \longrightarrow \langle E' \,,\, s'\rangle}{\langle \textbf{snd}\ E \,,\, s\rangle \longrightarrow \langle \textbf{snd}\ E' \,,\, s'\rangle}$$

Sums (Variants, Tagged Unions) – Syntax

$$T ::= \ldots \mid T + T$$

$$E ::= \ldots \mid \textbf{inl}\ E : T \mid \textbf{inr}\ E : T \mid$$
$$\textbf{case}\ E\ \textbf{of inl}\ x_1 : T_1 \Rightarrow E \mid \textbf{inr}\ x_2 : T_2 \Rightarrow E$$

x_1 and x_2 are binders for E_1 and E_2, up to alpha-equivalence

Sums – Typing I

(inl)
$$\frac{\Gamma \vdash E : T_1}{\Gamma \vdash \textbf{inl}\ E : T_1 + T_2 : T_1 + T_2}$$

(inr)
$$\frac{\Gamma \vdash E : T_2}{\Gamma \vdash \textbf{inr}\ E : T_1 + T_2 : T_1 + T_2}$$

(case)
$$\frac{\Gamma \vdash E : T_1 + T_2 \qquad \Gamma, x : T_1 \vdash E_1 : T \qquad \Gamma, y : T_2 \vdash E_2 : T}{\Gamma \vdash \textbf{case}\ E\ \textbf{of inl}\ x : T_1 \Rightarrow E_1 \mid \textbf{inr}\ y : T_2 \Rightarrow E_2 : T}$$

Sums – Typing II

$$\textbf{case}\ E\ \textbf{of inl}\ x : T_1 \Rightarrow E_1 \mid \textbf{inr}\ y : T_2 \Rightarrow E_2$$

Why do we need to carry around type annotations?

- maintain the unique typing property
  Otherwise **inl** 3: could be of type int $+$ int or int $+$ bool
- many programming languages allow type polymorphism

Sums – Semantics

Values

$v ::= \ldots \mid \textbf{inl}\ v : T \mid \textbf{inr}\ v : T$

SOS rules

(inl) $$\frac{\langle E\ ,\ s\rangle \longrightarrow \langle E'\ ,\ s'\rangle}{\langle \textbf{inl}\ E : T\ ,\ s\rangle \longrightarrow \langle \textbf{inl}\ E' : T\ ,\ s'\rangle}$$

(inr) $$\frac{\langle E\ ,\ s\rangle \longrightarrow \langle E'\ ,\ s'\rangle}{\langle \textbf{inr}\ E : T\ ,\ s\rangle \longrightarrow \langle \textbf{inr}\ E' : T\ ,\ s'\rangle}$$

(case1) $$\frac{\langle E\ ,\ s\rangle \longrightarrow \langle E'\ ,\ s'\rangle}{\begin{array}{l}\langle \textbf{case}\ E\ \textbf{of inl}\ x : T_1 \Rightarrow E_1 \mid \textbf{inr}\ y : T_2 \Rightarrow E_2\ ,\ s\rangle \\ \longrightarrow \langle \textbf{case}\ E'\ \textbf{of inl}\ x : T_1 \Rightarrow E_1 \mid \textbf{inr}\ y : T_2 \Rightarrow E_2\ ,\ s'\rangle\end{array}}$$

(case2) $$\begin{array}{l}\langle \textbf{case inl}\ v : T\ \textbf{of inl}\ x : T_1 \Rightarrow E_1 \mid \textbf{inr}\ y : T_2 \Rightarrow E_2\ ,\ s\rangle \\ \longrightarrow \langle \{v/x\}\, E_1\ ,\ s\rangle\end{array}$$

(case3) $$\begin{array}{l}\langle \textbf{case inr}\ v : T\ \textbf{of inl}\ x : T_1 \Rightarrow E_1 \mid \textbf{inr}\ y : T_2 \Rightarrow E_2\ ,\ s\rangle \\ \longrightarrow \langle \{v/y\}\, E_2\ ,\ s\rangle\end{array}$$

Constructors and Destructors

type	constructors	destructors
$T \to T$	$(\textbf{fn}\ x : T \Rightarrow \_)$	$\_\ E$
$T * T$	$(\_, \_)$	**fst** _ **snd** _
$T + T$	**inl** $\_ : T$ **inr** $\_ : T$	**case**
bool	`true` `false`	**if** _ **then** _ **else** _

Proofs as Programs

The Curry-Howard correspondence

(var)	$\Gamma, x : T \vdash x : T$	$\Gamma, P \vdash P$
(fn)	$\dfrac{\Gamma, x : T \vdash E : T'}{\Gamma \vdash (\textbf{fn}\ x : T \Rightarrow E) : T \to T'}$	$\dfrac{\Gamma, P \vdash P'}{\Gamma \vdash P \to P'}$
(app)	$\dfrac{\Gamma \vdash E_1 : T \to T' \qquad \Gamma \vdash E_2 : T}{\Gamma \vdash E_1\ E_2 : T'}$	$\dfrac{\Gamma \vdash P \to P' \qquad \Gamma \vdash P}{\Gamma \vdash P'}$ (modus ponens)

...

Proofs as Programs: The Curry-Howard correspondence

(var) $\Gamma, x{:}T \vdash x : T$ $\qquad$ $\Gamma, P \vdash P$

(fn) $\dfrac{\Gamma, x{:}T \vdash E : T'}{\Gamma \vdash (\textbf{fn}\ x : T \Rightarrow E) : T \to T'}$ $\qquad$ $\dfrac{\Gamma, P \vdash P'}{\Gamma \vdash P \to P'}$

(app) $\dfrac{\Gamma \vdash E_1 : T \to T' \qquad \Gamma \vdash E_2 : T}{\Gamma \vdash E_1\ E_2 : T'}$ $\qquad$ $\dfrac{\Gamma \vdash P \to P' \qquad \Gamma \vdash P}{\Gamma \vdash P'}$ (modus ponens)

(pair) $\dfrac{\Gamma \vdash E_1 : T_1 \qquad \Gamma \vdash E_2 : T_2}{\Gamma \vdash (E_1, E_2) : T_1 * T_2}$ $\qquad$ $\dfrac{\Gamma \vdash P_1 \qquad \Gamma \vdash P_2}{\Gamma \vdash P_1 \wedge P_2}$

(proj1) $\dfrac{\Gamma \vdash E : T_1 * T_2}{\Gamma \vdash \textbf{fst}\ E : T_1}$ (proj2) $\dfrac{\Gamma \vdash E : T_1 * T_2}{\Gamma \vdash \textbf{snd}\ E : T_2}$ $\qquad$ $\dfrac{\Gamma \vdash P_1 \wedge P_2}{\Gamma \vdash P_1}$ $\dfrac{\Gamma \vdash P_1 \wedge P_2}{\Gamma \vdash P_2}$

(inl) $\dfrac{\Gamma \vdash E : T_1}{\Gamma \vdash \textbf{inl}\ E : T_1 + T_2 : T_1 + T_2}$ (inr) $\dfrac{\Gamma \vdash E : T_2}{\Gamma \vdash \textbf{inr}\ E : T_1 + T_2 : T_1 + T_2}$ $\qquad$ $\dfrac{\Gamma \vdash P_1}{\Gamma \vdash P_1 \vee P_2}$ $\dfrac{\Gamma \vdash P_2}{\Gamma \vdash P_1 \vee P_2}$

(case) $\dfrac{\Gamma \vdash E : T_1 + T_2 \qquad \Gamma, x : T_1 \vdash E_1 : T \qquad \Gamma, y : T_2 \vdash E_2 : T}{\Gamma \vdash \textbf{case}\ E\ \textbf{of inl}\ x : T_1 \Rightarrow E_1 \mid \textbf{inr}\ y : T_2 \Rightarrow E_2 : T}$ $\qquad$ $\dfrac{\Gamma \vdash P_1 \vee P_2 \quad \Gamma, P_1 \vdash P \quad \Gamma, P_2 \vdash P}{\Gamma \vdash P}$

(unit), (zero), . . . ; **but not (letrec)**

Curry-Howard correspondence (abstract)

Programming side	**Logic side**
bottom type	false formula
unit type	true formula
sum type	disjunction
product type	conjunction
function type	implication
generalised sum type (Σ type)	existential quantification
generalised product type (Π type)	universal quantification

Datatypes in Haskell

Datatypes in Haskell generalise both sums and products

```
data Pair = P Int Double
data Either = I Int | D Double
```

The expression

```
data Expr = IntVal Int
          | BoolVal Bool
          | PairVal Int Bool
```

is (roughly) like saying

$$\text{Expr} = \text{int} + \text{bool} + (\text{int} * \text{bool})$$

More Datatypes - Records

A generalisation of products.
Labels $lab \in \mathbb{LAB}$ for a set $\mathbb{LAB} = \{p, q, ...\}$

$$T ::= \ldots \mid \{lab_1 : T_1, \ldots, lab_k : T_k\}$$
$$E ::= \ldots \mid \{lab_1 = E_1, \ldots, lab_k = E_k\} \mid \#lab\ E$$

(where in each record (type or expression) no lab occurs more than once)

Records – Typing

(record)

$$\frac{\Gamma \vdash E_1 : T_1 \qquad \ldots \qquad \Gamma \vdash E_k : T_k}{\Gamma \vdash \{lab_1 = E_1, \ldots, lab_k = E_k\} : \{lab_1 : T_1, \ldots, lab_k : T_k\}}$$

(recordproj)

$$\frac{\Gamma \vdash E : \{lab_1 : T_1, \ldots, lab_k : T_k\}}{\Gamma \vdash \#lab_i \ E : T_i}$$

Records – Semantics

Values

$v ::= \ldots \mid \{lab_1 = v_1, \ldots, lab_k = v_k\}$

SOS rules

(record1)
$$\frac{\langle E_i \, , \, s\rangle \longrightarrow \langle E_i' \, , \, s'\rangle}{\langle \{lab_1 = v_1, \ldots, lab_{i-1} = v_{i-1}, lab_i = E_i, \ldots, lab_k = E_k\} \, , \, s\rangle \longrightarrow \langle \{lab_1 = v_1, \ldots, lab_{i-1} = v_{i-1}, lab_i = E_i', \ldots, lab_k = E_k\} \, , \, s'\rangle}$$

(record2)
$$\langle \#lab_i \, \{lab_1 = v_1, \ldots, lab_k = v_k\} \, , \, s\rangle \longrightarrow \langle v_i \, , \, s\rangle$$

(record3)
$$\frac{\langle E \, , \, s\rangle \longrightarrow \langle E' \, , \, s'\rangle}{\langle \#lab_i \; E \, , \, s\rangle \longrightarrow \langle \#lab_i \; E' \, , \, s'\rangle}$$

Mutable Store I

Most languages have some kind of mutable store.
Two main choices:

1. our approach

$$E ::= \ldots \mid l := E \mid\ !l \mid x$$

   - locations store mutable values
   - variables refer to a previously calculated value – immutable
   - explicit dereferencing and assignment
     (**fn** x : int $\Rightarrow l := (!l) + x$)

Mutable Store II

Most languages have some kind of mutable store.
Two main choices:

2. languages as C or Java
   - variables can refer to a previously calculated value and overwrite that value
   - implicit dereferencing
   - some limited type machinery to limit mutability

```
void foo(x:int) {
  l = l + x
  ...
}
```

References

$$
\begin{array}{rcl}
T & ::= & \ldots \mid T \text{ ref} \\
T_{loc} & ::= & \cancel{\text{intref}} \;\; T \text{ ref} \\
E & ::= & \cdots \mid \cancel{ll := E} \mid \cancel{!l} \\
 & & \mid E_1 := E_2 \mid !E \mid \text{ref } E \mid l
\end{array}
$$

References – Typing

(ref)
$$\frac{\Gamma \vdash E : T}{\Gamma \vdash \mathsf{ref}\ E : T\ \mathsf{ref}}$$

(assign)
$$\frac{\Gamma \vdash E_1 : T\ \mathsf{ref} \qquad \Gamma \vdash E_2 : T}{\Gamma \vdash E_1 := E_2 : \mathsf{unit}}$$

(deref)
$$\frac{\Gamma \vdash E : T\ \mathsf{ref}}{\Gamma \vdash\ !E : T}$$

(loc)
$$\frac{\Gamma(l) = T\ \mathsf{ref}}{\Gamma \vdash l : T\ \mathsf{ref}}$$

References – Semantics I

Values

A location is a value $v ::= \ldots \mid l$

Stores s were finite partial functions $\mathbb{L} \rightharpoonup \mathbb{Z}$.
We now take them to be finite partial functions from $\mathbb{L}$ to all values.

SOS rules

(ref1) $\langle \text{ref } v\,,\, s\rangle \longrightarrow \langle l\,,\, s + \{l \mapsto v\}\rangle \quad \text{if } l \notin \text{dom}(s)$

(ref2) $$\frac{\langle E\,,\, s\rangle \longrightarrow \langle E'\,,\, s'\rangle}{\langle \text{ref } E\,,\, s\rangle \longrightarrow \langle \text{ref } E'\,,\, s'\rangle}$$

References – Semantics II

(deref1) $\langle !l\,,\, s\rangle \longrightarrow \langle v\,,\, s\rangle$ if $l \in \mathrm{dom}(s)$ and $s(l) = v$

(deref2) $$\frac{\langle E\,,\, s\rangle \longrightarrow \langle E'\,,\, s'\rangle}{\langle !E\,,\, s\rangle \longrightarrow \langle !E'\,,\, s'\rangle}$$

(assign1) $\langle l := v\,,\, s\rangle \longrightarrow \langle \textbf{skip}\,,\, s + \{l \mapsto v\}\rangle$ if $l \in \mathrm{dom}(s)$

(assign2) $$\frac{\langle E\,,\, s\rangle \longrightarrow \langle E'\,,\, s'\rangle}{\langle l := E\,,\, s\rangle \longrightarrow \langle l := E'\,,\, s'\rangle}$$

(assign3) $$\frac{\langle E\,,\, s\rangle \longrightarrow \langle E'\,,\, s'\rangle}{\langle E := E_2\,,\, s\rangle \longrightarrow \langle E' := E_2\,,\, s'\rangle}$$

Type Checking the Store

- so far we used $\text{dom}(\Gamma) \subseteq \text{dom}(s)$ in theorems such as progress and type preservation
- expressed 'all locations in Γ exist in store s'
- we need more
- for each $l \in \text{dom}(s)$ we require that $s(l)$ is typable
- moreover, $s(l)$ might contain some other locations ...

Type Checking – Example

Example

$$
\begin{aligned}
E \;=\; & \textbf{let val } x : (\text{int} \to \text{int})\ \text{ref} = \text{ref}\ (\textbf{fn } z : \text{int} \Rightarrow z)\ \textbf{in} \\
& (x := (\textbf{fn } z : \text{int} \Rightarrow \textbf{if } z \geq 1 \textbf{ then } z + ((!x)\ (z + -1)) \textbf{ else } 0); \\
& (!x)\ 3)\ \textbf{end}
\end{aligned}
$$

which has reductions

$$
\begin{aligned}
& \langle E\,,\,\{\}\rangle \\
\longrightarrow^* \;& \langle E_1\,,\,\{l_1 \mapsto (\textbf{fn } z : \text{int} \Rightarrow z)\rangle \\
\longrightarrow^* \;& \langle E_2\,,\,\{l_1 \mapsto (\textbf{fn } z : \text{int} \Rightarrow \textbf{if } z \geq 1 \textbf{ then } z + ((!l_1)(z + -1)) \textbf{ else } 0)\}\rangle \\
\longrightarrow^* \;& \langle 6\,,\,\ldots\rangle
\end{aligned}
$$

Progress and Type Preservation

Definition (Well-type store)

Let $\Gamma \vdash s$ if $\mathsf{dom}(\Gamma) = \mathsf{dom}(s)$ and if $\forall l \in \mathsf{dom}(s).\ \Gamma(l) = T\ \mathsf{ref} \Longrightarrow \Gamma \vdash s(l) : T$.

Theorem (Progress)

If E closed, $\Gamma \vdash E : T$ and $\Gamma \vdash s$ then either E is a value or there exist E' and s' such that $\langle E\ ,\ s\rangle \longrightarrow \langle E'\ ,\ s'\rangle$.

Theorem (Type Preservation)

If E closed, $\Gamma \vdash E : T$, $\Gamma \vdash s$ and $\langle E\ ,\ s\rangle \longrightarrow \langle E'\ ,\ s'\rangle$ then E' is closed and for some Γ' (with disjoint domain to Γ) $\Gamma, \Gamma' \vdash E' : T$ and $\Gamma, \Gamma' \vdash s'$.

Type Safety

Theorem (Type Safety)

If E *closed,* $\Gamma \vdash E : T$*,* $\Gamma \vdash s$*, and* $\langle E\ ,\ s\rangle \longrightarrow^* \langle E'\ ,\ s'\rangle$ *then either* E' *is a value with* $\Gamma \vdash E' : T$*, or there exist* E''*,* s'' *such that* $\langle E'\ ,\ s'\rangle \longrightarrow \langle E''\ ,\ s''\rangle$*, and there exists a* Γ' *s.t.* $\Gamma, \Gamma' \vdash E'' : T$ *and* $\Gamma, \Gamma' \vdash s''$*.*