CS 4110

Programming Languages & Logics

Lecture 21
Advanced Types

19 October 2016

Announcements

- HW #5 due today at 11:59pm
- HW #6 out

Review

So far we've seen how to develop a type system for λ-calculus, and have developed mathematical tools for proving type soundness.

Today we'll extend our type system with a number of other additional features commonly found in real-world languages, including products, sums, references, and exceptions.

Products

Syntax

$$
\begin{aligned}
e &::= \cdots \mid (e_1, e_2) \mid \#1\, e \mid \#2\, e \\
v &::= \cdots \mid (v_1, v_2)
\end{aligned}
$$

Products

Syntax

$$e ::= \cdots \mid (e_1, e_2) \mid \#1\, e \mid \#2\, e$$
$$v ::= \cdots \mid (v_1, v_2)$$

Semantics

$$E ::= \cdots \mid (E, e) \mid (v, E) \mid \#1\, E \mid \#2\, E$$

$$\frac{}{\#1\,(v_1, v_2) \to v_1} \qquad \frac{}{\#2\,(v_1, v_2) \to v_2}$$

Product Types

$$\tau_1 \times \tau_2$$

Product Types

$$\tau_1 \times \tau_2$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2}$$

Product Types

$$\tau_1 \times \tau_2$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2}$$

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \#1\, e : \tau_1}$$

Product Types

$$\tau_1 \times \tau_2$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2}$$

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \#1\, e : \tau_1}$$

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \#2\, e : \tau_2}$$

Note the similarities to the natural deduction rules for conjunction.

Sums

Syntax

$$e ::= \cdots \mid \mathsf{inl}_{\tau_1+\tau_2}\ e \mid \mathsf{inr}_{\tau_1+\tau_2}\ e \mid (\mathsf{case}\ e_1\ \mathsf{of}\ e_2 \mid e_3)$$
$$v ::= \cdots \mid \mathsf{inl}_{\tau_1+\tau_2}\ v \mid \mathsf{inr}_{\tau_1+\tau_2}\ v$$

Sums

Syntax

$$
\begin{aligned}
e &::= \cdots \mid \mathsf{inl}_{\tau_1+\tau_2}\ e \mid \mathsf{inr}_{\tau_1+\tau_2}\ e \mid (\mathsf{case}\ e_1\ \mathsf{of}\ e_2 \mid e_3) \\
v &::= \cdots \mid \mathsf{inl}_{\tau_1+\tau_2}\ v \mid \mathsf{inr}_{\tau_1+\tau_2}\ v
\end{aligned}
$$

Semantics

$$E ::= \cdots \mid \mathsf{inl}_{\tau_1+\tau_2}\ E \mid \mathsf{inr}_{\tau_1+\tau_2}\ E \mid (\mathsf{case}\ E\ \mathsf{of}\ e_2 \mid e_3)$$

$$\frac{}{\mathsf{case}\ \mathsf{inl}_{\tau_1+\tau_2}\ v\ \mathsf{of}\ e_2 \mid e_3 \rightarrow e_2\ v}$$

$$\frac{}{\mathsf{case}\ \mathsf{inr}_{\tau_1+\tau_2}\ v\ \mathsf{of}\ e_2 \mid e_3 \rightarrow e_3\ v}$$

Sum Types

$$\tau ::= \cdots \mid \tau_1 + \tau_2$$

Sum Types

$$\tau ::= \cdots \mid \tau_1 + \tau_2$$

$$\frac{\Gamma \vdash e : \tau_1}{\Gamma \vdash \mathsf{inl}_{\tau_1+\tau_2}\, e : \tau_1 + \tau_2}$$

Sum Types

$$\tau ::= \cdots \mid \tau_1 + \tau_2$$

$$\frac{\Gamma \vdash e : \tau_1}{\Gamma \vdash \mathsf{inl}_{\tau_1 + \tau_2}\ e : \tau_1 + \tau_2}$$

$$\frac{\Gamma \vdash e : \tau_2}{\Gamma \vdash \mathsf{inr}_{\tau_1 + \tau_2}\ e : \tau_1 + \tau_2}$$

Sum Types

$$\tau ::= \cdots \mid \tau_1 + \tau_2$$

$$\frac{\Gamma \vdash e : \tau_1}{\Gamma \vdash \mathsf{inl}_{\tau_1+\tau_2}\ e : \tau_1 + \tau_2}$$

$$\frac{\Gamma \vdash e : \tau_2}{\Gamma \vdash \mathsf{inr}_{\tau_1+\tau_2}\ e : \tau_1 + \tau_2}$$

$$\frac{\Gamma \vdash e : \tau_1 + \tau_2 \quad \Gamma \vdash e_1 : \tau_1 \to \tau \quad \Gamma \vdash e_2 : \tau_2 \to \tau}{\Gamma \vdash \mathsf{case}\ e\ \mathsf{of}\ e_1 \mid e_2 : \tau}$$

Example

$$\text{let } f = \lambda a : \mathbf{int} + (\mathbf{int} \to \mathbf{int}).\ \text{case } a \text{ of } (\lambda y.\, y + 1) \mid (\lambda g.\, g\ 35) \text{ in}$$
$$\text{let } h = \lambda x : \mathbf{int}.\, x + 7 \text{ in}$$
$$f\,(\text{inr}_{\mathbf{int}+(\mathbf{int}\to\mathbf{int})}\ h)$$

Example

$\text{let } f = \lambda a : \textbf{int} + (\textbf{int} \rightarrow \textbf{int}).\ \text{case } a \text{ of } (\lambda y.\, y + 1) \mid (\lambda g.\, g\ 35) \text{ in}$

$\text{let } h = \lambda x : \textbf{int}.\ x + 7 \text{ in}$

$f\ (\text{inr}_{\textbf{int}+(\textbf{int}\rightarrow\textbf{int})}\ h)$

Question: what does this evaluate to?

References

Syntax

$$
\begin{aligned}
e &::= \cdots \mid \mathsf{ref}\, e \mid !e \mid e_1 := e_2 \mid \ell \\
v &::= \cdots \mid \ell
\end{aligned}
$$

References

Syntax

$$e ::= \cdots \mid \mathsf{ref}\, e \mid\, !e \mid e_1 := e_2 \mid \ell$$
$$v ::= \cdots \mid \ell$$

Semantics

$$E ::= \cdots \mid \mathsf{ref}\, E \mid\, !E \mid E := e \mid v := E$$

$$\frac{\ell \notin dom(\sigma)}{\langle \sigma, \mathsf{ref}\, v \rangle \to \langle \sigma[\ell \mapsto v], \ell \rangle} \qquad \frac{\sigma(\ell) = v}{\langle \sigma,\ !\ell \rangle \to \langle \sigma, v \rangle}$$

$$\frac{}{\langle \sigma, \ell := v \rangle \to \langle \sigma[\ell \mapsto v], v \rangle}$$

Reference Types

$$\tau ::= \cdots \mid \tau\ \textbf{ref}$$

Reference Types

$$\tau ::= \cdots \mid \tau\ \textbf{ref}$$

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \mathsf{ref}\ e : \tau\ \textbf{ref}}$$

Reference Types

$$\tau ::= \cdots \mid \tau\ \textbf{ref}$$

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \mathsf{ref}\ e : \tau\ \textbf{ref}}$$

$$\frac{\Gamma \vdash e : \tau\ \textbf{ref}}{\Gamma \vdash\ !e : \tau}$$

Reference Types

$$\tau ::= \cdots \mid \tau\ \textbf{ref}$$

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \mathsf{ref}\ e : \tau\ \textbf{ref}}$$

$$\frac{\Gamma \vdash e : \tau\ \textbf{ref}}{\Gamma \vdash\ !e : \tau}$$

$$\frac{\Gamma \vdash e_1 : \tau\ \textbf{ref} \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1 := e_2 : \tau}$$

Question

Is this type system sound?

Question

Is this type system sound?

Well... what is the type of a location ℓ?

Question

Is this type system sound?

Well... what is the type of a location ℓ?

Oops!

Store Typings

Let Σ range over maps from locations to types

Store Typings

Let Σ range over maps from locations to types

$$\frac{\Gamma, \Sigma \vdash e : \tau}{\Gamma, \Sigma \vdash \mathrm{ref}\, e : \tau\ \textbf{ref}}$$

Store Typings

Let Σ range over maps from locations to types

$$\frac{\Gamma, \Sigma \vdash e : \tau}{\Gamma, \Sigma \vdash \mathrm{ref}\, e : \tau\ \mathbf{ref}}$$

$$\frac{\Gamma, \Sigma \vdash e : \tau\ \mathbf{ref}}{\Gamma, \Sigma \vdash !e : \tau}$$

Store Typings

Let Σ range over maps from locations to types

$$\frac{\Gamma, \Sigma \vdash e : \tau}{\Gamma, \Sigma \vdash \mathrm{ref}\, e : \tau\ \mathbf{ref}}$$

$$\frac{\Gamma, \Sigma \vdash e : \tau\ \mathbf{ref}}{\Gamma, \Sigma \vdash\ !e : \tau}$$

$$\frac{\Gamma, \Sigma \vdash e_1 : \tau\ \mathbf{ref} \quad \Gamma, \Sigma \vdash e_2 : \tau}{\Gamma, \Sigma \vdash e_1 := e_2 : \tau}$$

Store Typings

Let Σ range over maps from locations to types

$$\frac{\Gamma, \Sigma \vdash e : \tau}{\Gamma, \Sigma \vdash \mathrm{ref}\, e : \tau\ \textbf{ref}}$$

$$\frac{\Gamma, \Sigma \vdash e : \tau\ \textbf{ref}}{\Gamma, \Sigma \vdash\ !e : \tau}$$

$$\frac{\Gamma, \Sigma \vdash e_1 : \tau\ \textbf{ref} \quad \Gamma, \Sigma \vdash e_2 : \tau}{\Gamma, \Sigma \vdash e_1 := e_2 : \tau}$$

$$\frac{\Sigma(\ell) = \tau}{\Gamma, \Sigma \vdash \ell : \tau\ \textbf{ref}}$$

Reference Types Metatheory

Definition

Store σ is *well-typed* with respect to typing context Γ and store typing Σ, written $\Gamma, \Sigma \vdash \sigma$, if $dom(\sigma) = dom(\Sigma)$ and for all $\ell \in dom(\sigma)$ we have $\Gamma, \Sigma \vdash \sigma(\ell) : \Sigma(\ell)$.

Reference Types Metatheory

Definition

Store σ is *well-typed* with respect to typing context Γ and store typing Σ, written $\Gamma, \Sigma \vdash \sigma$, if $\mathit{dom}(\sigma) = \mathit{dom}(\Sigma)$ and for all $\ell \in \mathit{dom}(\sigma)$ we have $\Gamma, \Sigma \vdash \sigma(\ell) : \Sigma(\ell)$.

Theorem (Type soundness)

If $\cdot, \Sigma \vdash e : \tau$ and $\cdot, \Sigma \vdash \sigma$ and $\langle e, \sigma \rangle \to^* \langle e', \sigma' \rangle$ then either e' is a value, or there exists e'' and σ'' such that $\langle e', \sigma' \rangle \to \langle e'', \sigma'' \rangle$.

Reference Types Metatheory

Definition

Store σ is *well-typed* with respect to typing context Γ and store typing Σ, written $\Gamma, \Sigma \vdash \sigma$, if $dom(\sigma) = dom(\Sigma)$ and for all $\ell \in dom(\sigma)$ we have $\Gamma, \Sigma \vdash \sigma(\ell) : \Sigma(\ell)$.

Theorem (Type soundness)

If $\cdot, \Sigma \vdash e : \tau$ *and* $\cdot, \Sigma \vdash \sigma$ *and* $\langle e, \sigma \rangle \to^* \langle e', \sigma' \rangle$ *then either* e' *is a value, or there exists* e'' *and* σ'' *such that* $\langle e', \sigma' \rangle \to \langle e'', \sigma'' \rangle$.

Lemma (Preservation)

If $\Gamma, \Sigma \vdash e : \tau$ *and* $\Gamma, \Sigma \vdash \sigma$ *and* $\langle e, \sigma \rangle \to \langle e', \sigma' \rangle$ *then there exists some* $\Sigma' \supseteq \Sigma$ *such that* $\Gamma, \Sigma' \vdash e' : \tau$ *and* $\Gamma, \Sigma' \vdash \sigma'$.

Landin's Knot

It turns out that using references we (re)-gain the ability define arbitrary recursive functions!

Landin's Knot

It turns out that using references we (re)-gain the ability define arbitrary recursive functions!

$$\begin{array}{l}\textbf{let}\ r = \mathsf{ref}\ \lambda x.\, 0\ \textbf{in} \\ r := \lambda x\!:\!\textbf{int}.\ \textbf{if}\ x = 0\ \textbf{then}\ 1\ \textbf{else}\ x \times\ !r\,(x-1)\end{array}$$

Landin's Knot

It turns out that using references we (re)-gain the ability define arbitrary recursive functions!

$$\begin{array}{l} \textbf{let}\ r = \mathrm{ref}\ \lambda x.\, 0\ \textbf{in} \\ r := \lambda x\!:\!\textbf{int}.\ \textbf{if}\ x = 0\ \textbf{then}\ 1\ \textbf{else}\ x \times\ !r\,(x-1) \end{array}$$

This trick is called "Landin's knot" after its creator.

Fixpoints

Syntax

$$e ::= \cdots \mid \mathsf{fix}\; e$$

Fixpoints

Syntax

$$e ::= \cdots \mid \mathsf{fix}\, e$$

Semantics

$$E ::= \cdots \mid \mathsf{fix}\, E$$

$$\frac{}{\mathsf{fix}\, \lambda x\!:\!\tau.\, e \to e\{(\mathsf{fix}\, \lambda x\!:\!\tau.\, e)/x\}}$$

Fixpoints

Syntax

$$e ::= \cdots \mid \mathsf{fix}\, e$$

Semantics

$$E ::= \cdots \mid \mathsf{fix}\, E$$

$$\frac{}{\mathsf{fix}\; \lambda x\!:\!\tau.\, e \rightarrow e\{(\mathsf{fix}\; \lambda x\!:\!\tau.\, e)/x\}}$$

The typing rule for fix is left as an exercise...

Fixpoint Examples

With fix, we can define letrec $x\!:\!\tau = e_1$ in e_2 as syntactic sugar:

$$\text{letrec}\, x\!:\!\tau = e_1 \text{ in } e_2 \triangleq \text{let}\, x = \text{fix}\ \lambda x\!:\!\tau.\, e_1 \text{ in } e_2$$

Fixpoint Examples

With fix, we can define letrec $x : \tau = e_1$ in e_2 as syntactic sugar:

$$\text{letrec } x : \tau = e_1 \text{ in } e_2 \triangleq \text{let } x = \text{fix } \lambda x : \tau.\, e_1 \text{ in } e_2$$

We can also take fixpoints at other types. For example, consider the following expression,

$$\begin{aligned}&\text{fix } \lambda x : (\mathbf{int} \to \mathbf{int}) \times (\mathbf{int} \to \mathbf{int}).\\ &\qquad (\lambda n : \mathbf{int}.\ \text{if } n = 0 \text{ then true else } (\#2\, x)\ (n - 1),\\ &\qquad\ \lambda n : \mathbf{int}.\ \text{if } n = 0 \text{ then false else } (\#1\, x)\ (n - 1))\end{aligned}$$

which defines a pair of mutually recursive functions; the first returns true if and only if its argument is even; the second returns true if and only if its argument is odd.

Exceptions

Syntax

$$e ::= \ldots \,\textbf{error} \mid \textbf{try}\ e\ \textbf{with}\ e$$

Exceptions

Syntax

$$e ::= \ldots \,\textbf{error} \mid \textbf{try}\ e\ \textbf{with}\ e$$

Semantics

$$E ::= \cdots \mid \textbf{try}\ E\ \textbf{with}\ e$$

$$\frac{}{E[\textbf{error}] \to \textbf{error}}$$

Exceptions

Syntax

$$e ::= \ldots\ \textbf{error} \mid \textbf{try}\ e\ \textbf{with}\ e$$

Semantics

$$E ::= \cdots \mid \textbf{try}\ E\ \textbf{with}\ e$$

$$\frac{}{E[\textbf{error}] \rightarrow \textbf{error}} \qquad \frac{}{\textbf{try error with}\ e \rightarrow e}$$

Exceptions

Syntax

$$e ::= \ldots \ \mathbf{error} \mid \mathbf{try}\ e\ \mathbf{with}\ e$$

Semantics

$$E ::= \cdots \mid \mathbf{try}\ E\ \mathbf{with}\ e$$

$$\frac{}{E[\mathbf{error}] \to \mathbf{error}} \qquad \frac{}{\mathbf{try\ error\ with}\ e \to e} \qquad \mathbf{try}\ v\ \mathbf{with}\ e \to v$$

Exception Types

We don't need to add any new types...

$$\frac{}{\Gamma \vdash \textbf{error} : \tau}$$

$$\frac{\Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \textbf{try}\ e_1\ \textbf{with}\ e_2 : \tau}$$

Exception Metatheory

Lemma (Progress)

If $\vdash e : \tau$ *then either*

- *e is a value or*
- *e is* ***error*** *or*
- *there exists e' such that $e \to e'$.*

Exception Metatheory

Lemma (Progress)

If $\vdash e : \tau$ *then either*

- *e is a value or*
- *e is* ***error*** *or*
- *there exists* e' *such that* $e \rightarrow e'$.

Theorem (Soundness)

If $\vdash e : \tau$ *and* $e \rightarrow^* e'$ *and* $e' \not\rightarrow$ *then either*

- *e is a value or*
- *e is* ***error***