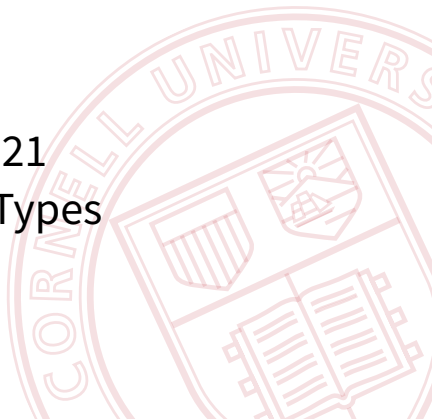


CS 4110

Programming Languages & Logics

Lecture 21
Advanced Types



Review

We've developed a type system for the λ -calculus and mathematical tools for proving its type soundness.

We also know how to extend the λ -calculus with new language features.

Today, we'll extend our *type system* with features commonly found in real-world languages: products, sums, references, and exceptions.

Products (Pairs)

Syntax

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

Products (Pairs)

Syntax

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

Semantics

$$E ::= \dots \mid (E, e) \mid (v, E) \mid \#1 E \mid \#2 E$$
$$\frac{}{\#1 (v_1, v_2) \rightarrow v_1}$$
$$\frac{}{\#2 (v_1, v_2) \rightarrow v_2}$$

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

$$\tau_1 \times \tau_2$$
$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2}$$
$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \#1 e : \tau_1}$$
$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \#2 e : \tau_2}$$

Sums (Tagged Unions)

Syntax

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

Sums (Tagged Unions)

Syntax

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

Semantics

$$E ::= \dots \mid \text{inl}_{\tau_1 + \tau_2} E \mid \text{inr}_{\tau_1 + \tau_2} E \mid (\text{case } E \text{ of } e_2 \mid e_3)$$
$$\frac{}{\text{case inl}_{\tau_1 + \tau_2} v \text{ of } e_2 \mid e_3 \rightarrow e_2 v}$$
$$\frac{}{\text{case inr}_{\tau_1 + \tau_2} v \text{ of } e_2 \mid e_3 \rightarrow e_3 v}$$

Sum Types

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

Sum Types

$$\tau ::= \dots \mid \tau_1 + \tau_2$$
$$\frac{\Gamma \vdash e : \tau_1}{\Gamma \vdash \text{inl}_{\tau_1 + \tau_2} e : \tau_1 + \tau_2}$$
$$\frac{\Gamma \vdash e : \tau_2}{\Gamma \vdash \text{inr}_{\tau_1 + \tau_2} e : \tau_1 + \tau_2}$$

Sum Types

$$\tau ::= \dots \mid \tau_1 + \tau_2$$
$$\frac{\Gamma \vdash e : \tau_1}{\Gamma \vdash \text{inl}_{\tau_1 + \tau_2} e : \tau_1 + \tau_2}$$
$$\frac{\Gamma \vdash e : \tau_2}{\Gamma \vdash \text{inr}_{\tau_1 + \tau_2} e : \tau_1 + \tau_2}$$
$$\frac{\Gamma \vdash e : \tau_1 + \tau_2 \quad \Gamma \vdash e_1 : \tau_1 \rightarrow \tau \quad \Gamma \vdash e_2 : \tau_2 \rightarrow \tau}{\Gamma \vdash \text{case } e \text{ of } e_1 \mid e_2 : \tau}$$

Example

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

References

Syntax

$$e ::= \dots \mid \text{ref } e \mid !e \mid e_1 := e_2 \mid \ell$$
$$v ::= \dots \mid \ell$$

References

Syntax

$$e ::= \dots \mid \text{ref } e \mid !e \mid e_1 := e_2 \mid \ell$$
$$v ::= \dots \mid \ell$$

Semantics

$$E ::= \dots \mid \text{ref } E \mid !E \mid E := e \mid v := E$$

$$\frac{\ell \notin \text{dom}(\sigma)}{\langle \sigma, \text{ref } v \rangle \rightarrow \langle \sigma[\ell \mapsto v], \ell \rangle}$$

$$\frac{\sigma(\ell) = v}{\langle \sigma, !\ell \rangle \rightarrow \langle \sigma, v \rangle}$$

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

Reference Types

$$\tau ::= \dots \mid \tau \mathbf{ref}$$

Reference Types

$$\tau ::= \dots \mid \tau \mathbf{ref}$$
$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \mathbf{ref} \ e : \tau \mathbf{ref}}$$

Reference Types

$$\tau ::= \dots \mid \tau \mathbf{ref}$$
$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \mathbf{ref} e : \tau \mathbf{ref}}$$
$$\frac{\Gamma \vdash e : \tau \mathbf{ref}}{\Gamma \vdash !e : \tau}$$

Reference Types

$$\tau ::= \dots \mid \tau \mathbf{ref}$$
$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \mathbf{ref} \ e : \tau \mathbf{ref}}$$
$$\frac{\Gamma \vdash e : \tau \mathbf{ref}}{\Gamma \vdash !e : \tau}$$
$$\frac{\Gamma \vdash e_1 : \tau \mathbf{ref} \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1 := e_2 : \tau}$$

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 partial functions from locations to types.

Store Typings

Let Σ range over partial functions from locations to types.

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

Store Typings

Let Σ range over partial functions from locations to types.

$$\frac{\Gamma, \Sigma \vdash e : \tau}{\Gamma, \Sigma \vdash \text{ref } e : \tau \text{ **ref**}}$$

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

Store Typings

Let Σ range over partial functions from locations to types.

$$\frac{\Gamma, \Sigma \vdash e : \tau}{\Gamma, \Sigma \vdash \text{ref } e : \tau \text{ **ref**}}$$

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

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

Store Typings

Let Σ range over partial functions from locations to types.

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

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

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

$$\frac{\Sigma(l) = \tau}{\Gamma, \Sigma \vdash l : \tau \mathbf{ref}}$$

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 $\text{dom}(\sigma) = \text{dom}(\Sigma)$ and for all $\ell \in \text{dom}(\sigma)$ 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 $\text{dom}(\sigma) = \text{dom}(\Sigma)$ and for all $\ell \in \text{dom}(\sigma)$ we have $\Gamma, \Sigma \vdash \sigma(\ell) : \Sigma(\ell)$.

Theorem (Type soundness)

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

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 $\text{dom}(\sigma) = \text{dom}(\Sigma)$ and for all $\ell \in \text{dom}(\sigma)$ we have $\Gamma, \Sigma \vdash \sigma(\ell) : \Sigma(\ell)$.

Theorem (Type soundness)

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

Lemma (Preservation)

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

Landin's Knot

Using references, we (re)gain the ability define recursive functions!

```
let  $r = \text{ref } \lambda x:\text{int}. 0$  in
```

Landin's Knot

Using references, we (re)gain the ability define recursive functions!

```
let  $r = \text{ref } \lambda x: \text{int}. 0$  in  
let  $f = (\lambda x: \text{int}. \text{if } x = 0 \text{ then } 1 \text{ else } x \times (!r) (x - 1))$  in
```


Landin's Knot

Using references, we (re)gain the ability define recursive functions!

```
let  $r = \text{ref } \lambda x:\text{int}. 0$  in  
let  $f = (\lambda x:\text{int}. \text{if } x = 0 \text{ then } 1 \text{ else } x \times (!r) (x - 1))$  in  
let  $a = (r := f)$  in
```

Landin's Knot

Using references, we (re)gain the ability define recursive functions!

```
let  $r = \text{ref } \lambda x:\text{int}. 0$  in  
let  $f = (\lambda x:\text{int}. \text{if } x = 0 \text{ then } 1 \text{ else } x \times (!r) (x - 1))$  in  
let  $a = (r := f)$  in  
 $f\ 5$ 
```

Fixed Points

Syntax

$$e ::= \dots \mid \text{fix } e$$

Fixed Points

Syntax

$$e ::= \dots \mid \text{fix } e$$

Semantics

$$E ::= \dots \mid \text{fix } E$$
$$\frac{}{\text{fix } \lambda x:\tau. e \rightarrow e\{(\text{fix } \lambda x:\tau. e)/x\}}$$

Fixed Points

Syntax

$$e ::= \dots \mid \text{fix } e$$

Semantics

$$E ::= \dots \mid \text{fix } E$$
$$\frac{}{\text{fix } \lambda x:\tau. e \rightarrow e\{(\text{fix } \lambda x:\tau. e)/x\}}$$

The typing rule for fix is on the homework...