We’ve developed a type system for the \( \lambda \)-calculus and mathematical tools for proving its type soundness.

We also know how to extend the \( \lambda \)-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

\[ e ::= \cdots \mid (e_1, e_2) \mid \#1 e \mid \#2 e \]
\[
\nu ::= \cdots \mid (\nu_1, \nu_2)
\]
Products (Pairs)

Syntax

\[
e ::= \cdots | (e_1, e_2) | \#_1 e | \#_2 e
\]

\[
v ::= \cdots | (v_1, v_2)
\]

Semantics

\[
E ::= \cdots | (E, e) | (v, E) | \#_1 E | \#_2 E
\]

\[
\#_1 (v_1, v_2) \rightarrow v_1 \\
\#_2 (v_1, v_2) \rightarrow v_2
\]
Product Types

$\tau_1 \times \tau_2$
Product Types

\[ \tau_1 \times \tau_2 \]

\[ \Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2 \]

\[ \Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2 \]
Product Types

\[ \tau_1 \times \tau_2 \]

\[
\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2 \\
\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2
\]

\[
\Gamma \vdash e : \tau_1 \times \tau_2 \\
\Gamma \vdash \#1 e : \tau_1
\]

\[
\Gamma \vdash e : \tau_1 \times \tau_2 \\
\Gamma \vdash \#2 e : \tau_2
\]
Sums (Tagged Unions)

Syntax

\[
\begin{align*}
e & ::= \cdots | \text{inl}_{\tau_1+\tau_2} \ e | \text{inr}_{\tau_1+\tau_2} \ e | \text{(case } e_1 \text{ of } e_2 | e_3) \\
v & ::= \cdots | \text{inl}_{\tau_1+\tau_2} \ v | \text{inr}_{\tau_1+\tau_2} \ v
\end{align*}
\]
Sums (Tagged Unions)

Syntax

\[ e ::= \cdots \mid \text{inl}_{\tau_1+\tau_2} e \mid \text{inr}_{\tau_1+\tau_2} e \mid (\text{case } e_1 \text{ of } e_2 \mid e_3) \]

\[ v ::= \cdots \mid \text{inl}_{\tau_1+\tau_2} v \mid \text{inr}_{\tau_1+\tau_2} v \]

Semantics

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

\[ \text{case } \text{inl}_{\tau_1+\tau_2} v \text{ of } e_2 \mid e_3 \rightarrow e_2 v \]

\[ \text{case } \text{inr}_{\tau_1+\tau_2} v \text{ of } e_2 \mid e_3 \rightarrow e_3 v \]
Sum Types

\[ \tau ::= \cdots \mid \tau_1 + \tau_2 \]
Sum Types

\[ \tau ::= \cdots | \tau_1 + \tau_2 \]

\[
\begin{align*}
\Gamma \vdash e : \tau_1 \\
\Gamma \vdash \text{inl}_{\tau_1 + \tau_2} e : \tau_1 + \tau_2 \\
\Gamma \vdash e : \tau_2 \\
\Gamma \vdash \text{inr}_{\tau_1 + \tau_2} e : \tau_1 + \tau_2
\end{align*}
\]
Sum Types

\[ \tau ::= \cdots | \tau_1 + \tau_2 \]

\[ \Gamma \vdash e : \tau_1 \]
\[ \Gamma \vdash \text{inl}_{\tau_1 + \tau_2} e : \tau_1 + \tau_2 \]

\[ \Gamma \vdash e : \tau_2 \]
\[ \Gamma \vdash \text{inr}_{\tau_1 + \tau_2} e : \tau_1 + \tau_2 \]

\[ \Gamma \vdash e : \tau_1 + \tau_2 \quad \Gamma \vdash e_1 : \tau_1 \rightarrow \tau \quad \Gamma \vdash e_2 : \tau_2 \rightarrow \tau \]
\[ \Gamma \vdash \text{case } e \text{ of } e_1 | e_2 : \tau \]
Example

let $f = \lambda a : \texttt{int} + (\texttt{int} \rightarrow \texttt{int})$. 

    case $a$ of $(\lambda y : \texttt{int}. y + 1) \mid (\lambda g : \texttt{int} \rightarrow \texttt{int}. g \ 35)$ in 

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

$f (\text{inr} \cdot \text{int} + (\text{int} \rightarrow \text{int}) \ h)$
References

Syntax

\[ e ::= \cdots \mid \text{ref } e \mid ! e \mid e_1 ::= e_2 \mid \ell \]

\[ v ::= \cdots \mid \ell \]
References

Syntax

\[
e ::= \cdots \mid \text{ref } e \mid !e \mid e_1 ::= e_2 \mid \ell
\]

\[
v ::= \cdots \mid \ell
\]

Semantics

\[
E ::= \cdots \mid \text{ref } E \mid !E \mid E ::= e \mid v ::= E
\]

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

\[
\sigma(\ell) = v \quad \frac{}{\langle \sigma, !\ell \rangle \rightarrow \langle \sigma, v \rangle}
\]

\[
\langle \sigma, \ell ::= v \rangle \rightarrow \langle \sigma[\ell \mapsto v], v \rangle
\]
Reference Types

\[ \tau ::= \cdots \mid \tau \text{ref} \]
Reference Types

\[ \tau ::= \cdots \mid \tau\text{ ref} \]

\[
\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \text{ref } e : \tau\text{ ref}}
\]
Reference Types

\[ \tau ::= \cdots \mid \tau \text{ ref} \]

\[
\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \text{ref } e : \tau \text{ ref}}
\]

\[
\frac{\Gamma \vdash e : \tau \text{ ref}}{\Gamma \vdash !e : \tau}
\]
Reference Types

\[ \tau ::= \cdots \mid \tau \text{ref} \]

\[ \Gamma \vdash e : \tau \quad \Gamma \vdash \text{ref} e : \tau \text{ref} \]

\[ \Gamma \vdash e \text{ref} \quad \Gamma \vdash !e : \tau \]

\[ \Gamma \vdash e_1 \text{ref} \quad \Gamma \vdash e_2 : \tau \quad \Gamma \vdash e_1 := e_2 : \tau \]
Is this type system sound?
Question

Is this type system sound?

Well... what is the type of a location $\ell$?
Is this type system sound?

Well... what is the type of a location $\ell$? (Oops!)
Let $\Sigma$ range over partial functions from locations to types.
Store Typings

Let \( \Sigma \) range over partial functions from locations to types.

\[
\Gamma, \Sigma \vdash e : \tau \\
\Gamma, \Sigma \vdash \text{ref } e : \tau \text{ ref}
\]
Store Typings

Let $\Sigma$ range over partial functions from locations to types.

\[
\Gamma, \Sigma \vdash e : \tau \\
\Gamma, \Sigma \vdash \text{ref } e : \tau \text{ ref}
\]

\[
\Gamma, \Sigma \vdash e : \tau \text{ ref} \\
\Gamma, \Sigma \vdash !e : \tau
\]
Store Typings

Let \( \Sigma \) range over partial functions from locations to types.

\[
\begin{align*}
\Gamma, \Sigma \vdash e : \tau \\
\frac{}{\Gamma, \Sigma \vdash \text{ref} \; e : \tau \text{ ref}} \\
\Gamma, \Sigma \vdash e : \tau \text{ ref} \\
\frac{}{\Gamma, \Sigma \vdash !e : \tau} \\
\Gamma, \Sigma \vdash e_1 : \tau \text{ ref} \quad \Gamma, \Sigma \vdash e_2 : \tau \\
\frac{}{\Gamma, \Sigma \vdash e_1 := e_2 : \tau}
\end{align*}
\]
Let $\Sigma$ range over partial functions from locations to types.

\[
\Gamma, \Sigma \vdash e : \tau \\
\Gamma, \Sigma \vdash \text{ref } e : \tau \text{ ref}
\]

\[
\Gamma, \Sigma \vdash e : \tau \text{ ref} \\
\Gamma, \Sigma \vdash !e : \tau
\]

\[
\Gamma, \Sigma \vdash e_1 : \tau \text{ ref} \quad \Gamma, \Sigma \vdash e_2 : \tau \\
\Gamma, \Sigma \vdash e_1 := e_2 : \tau
\]

\[
\Sigma(\ell) = \tau \\
\Gamma, \Sigma \vdash \ell : \tau \text{ ref}
\]
Reference Types Metatheory

**Definition**

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

Store $\sigma$ is **well-typed** with respect to typing context $\Gamma$ and store typing $\Sigma$, 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 $\langle e, \sigma \rangle \rightarrow^* \langle e', \sigma' \rangle$ and $\langle e', \sigma' \rangle \not\rightarrow$, then $e'$ is a value.
**Definition**

Store $\sigma$ is *well-typed* with respect to typing context $\Gamma$ and store typing $\Sigma$, 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 $\langle e, \sigma \rangle \rightarrow^* \langle e', \sigma' \rangle$ and $\langle e', \sigma' \rangle \not\rightarrow$, then $e'$ is a value.

**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!

\[
\text{let } r = \text{ref } \lambda x : \text{int}. \ 0 \ \text{in}
\]
Landin’s Knot

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

```ml
let r = 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
```
Using references, we (re)gain the ability define recursive functions!

```ml
let r = ref \( \lambda x : \text{int} \cdot 0 \) in
let f = (\( \lambda x : \text{int} \cdot \text{if } x = 0 \text{ then } 1 \text{ else } x \times (!r) \cdot (x - 1) \)) in
let a = (r := f) in
```
Landin’s Knot

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

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

Syntax

\[ e ::= \cdots | \text{fix } e \]
Fixed Points

Syntax

\[ e ::= \cdots \mid \text{fix } e \]

Semantics

\[ E ::= \cdots \mid \text{fix } E \]

\[
\text{fix } \lambda x: \tau. \ e \rightarrow e\{(\text{fix } \lambda x: \tau. \ e)/x\}
\]
Fixed Points

Syntax

\[ e ::= \cdots | \text{fix} \ e \]

Semantics

\[ E ::= \cdots | \text{fix} \ E \]

\[
\text{fix} \ \lambda x : \tau. \ e \rightarrow e\{\left(\text{fix} \ \lambda x : \tau. \ e\right)/x\}
\]

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