Scribe: Daniel Kifer, Yanling Wang | Lecturer: Andrew Myers

1 Review

In the last lecture, we defined the syntax for typed lambda calculus. It is similar to the original lambda calculus - but has an added type annotation (e is an expression, τ is a type and v is a value):

$e ::= x|e_1e_2|\lambda x : \tau.\, e|n|b|u \qquad n \in \mathbb{Z},\ b \in \mathbb{T},\ u \in \mathbb{U}.$

$\tau ::= B|\tau_1 \to \tau_2 \qquad$ where B is a boolean, integer, or unit.

$v ::= \lambda x : \tau.\, e|n|b|u$

We also need special operators for the base types (such as $PLUS : int \to int \to int$ and $NOT : bool \to bool$). As it turns out, we only need to add type annotation to lambda expressions and it will be sufficient to build a syntax directed type-checker.

We define the typing context $\Gamma : Var \to Type$ and define $\Gamma, x : \tau$ to mean "extend Γ with x mapped to τ". A type judgement $\Gamma \vdash e : \tau$ means Γ shows that e has type τ when the free variables in e are looked up in Γ. A program e is considered well-typed (and well-formed) if for some type τ, the empty typing context shows that e has type τ and we express it in this way: $\vdash e : \tau$. The typing rules we need are:

$$\frac{}{\Gamma, x : \tau \vdash x : \tau} \qquad \frac{\Gamma \vdash e_1 : \tau \to \tau' \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1 e_2 : \tau'} \qquad \frac{\Gamma, x : \tau \vdash e : \tau'}{\Gamma \vdash (\lambda x : \tau.\, e) : \tau \to \tau'}$$

2 Denotational Soundness of Typing Rules

Are the typing rules sound from a denotational perspective? Every well-formed program has a meaning and this meaning should have the same type it was assigned by the type-checker. Formally, our typing rules are sound if: $\mathcal{C}[\![\vdash e : \tau]\!]\rho_0 \in \mathcal{T}[\![\tau]\!]$. Here we are taking the meaning of the type derivation of e instead of just the meaning of e and we use ρ_0 to represent the empty environment.

We have the following interpretation of types as domains:

$\mathcal{T}[\![\mathsf{int}]\!] = \mathbb{Z}$

$\mathcal{T}[\![\mathsf{bool}]\!] = \mathbb{T}$

$\mathcal{T}[\![\tau_1 \to \tau_2]\!] = \mathcal{T}[\![\tau_1]\!] \to \mathcal{T}[\![\tau_2]\!]$

and we define the notation $\rho \models \Gamma$ (ρ agrees with Γ) in the following way:

$\rho \models \Gamma \overset{def}{\iff} \forall x \in dom(\Gamma), \rho(x) \in \mathcal{T}[\![\Gamma(x)]\!]$

By writing the denotational semantics of this language, it should be clear that:

$\rho \models \Gamma \Rightarrow \mathcal{C}[\![\Gamma \vdash e : \tau]\!]\rho \in \mathcal{T}[\![\tau]\!]$

Our soundness claim is a specific example of this, with ρ being the empty environment ρ_0 and Γ being the empty context. The claim can be proven using structural induction on the type derivation and a brief sketch of this proof follows:

- $\mathcal{C}[\![\Gamma \vdash n : \mathsf{int}]\!] = n \in \mathbb{Z} = \mathcal{T}[\![\mathsf{int}]\!]$ and similarly for booleans and unit.
- $\mathcal{C}[\![\Gamma, x : \tau \vdash x : \tau]\!]\rho = \rho x$ and clearly $\rho(x) \in \mathcal{T}[\![\tau]\!] = \mathcal{T}[\![\Gamma(x)]\!]$
- $\mathcal{C}[\![\Gamma \vdash e_1\ e_2 : \tau']\!]\rho = \mathcal{C}[\![\Gamma \vdash e_1 : \tau \to \tau']\!]\rho(\mathcal{C}[\![\Gamma \vdash e_2 : \tau]\!]\rho)$
  From the typing derivation, we have $\Gamma \vdash e_1 : \tau \to \tau'$ and $\Gamma \vdash e_2 : \tau$. Since $\mathcal{C}[\![\Gamma \vdash e_1 : \tau \to \tau']\!]\rho$ and $\mathcal{C}[\![\Gamma \vdash e_2 : \tau]\!]\rho$ have shorter typing derivation, we can use induction hypothesis: $\mathcal{C}[\![\Gamma \vdash e_1 : \tau \to \tau']\!]\rho \in \mathcal{T}[\![\tau]\!] \to \mathcal{T}[\![\tau']\!]$ and $\mathcal{C}[\![\Gamma \vdash e_2 : \tau]\!]\rho \in \mathcal{T}[\![\tau]\!]$. Therefore $\mathcal{C}[\![\Gamma \vdash e_1\ e_2 : \tau']\!]\rho \in \mathcal{T}[\![\tau']\!]$.
- $\mathcal{C}[\![\Gamma \vdash (\lambda x : \tau.\, e) : \tau \to \tau']\!]\rho = \lambda v \in \mathcal{T}[\![\tau]\!].\mathcal{C}[\![\Gamma, x : \tau \vdash e : \tau']\!]\rho[x \mapsto v]$
  From the type derivation we know $\Gamma, x : \tau \vdash e$ and $\rho[x \mapsto v] \models \Gamma, x : \tau$, so $\mathcal{C}[\![\Gamma, x : \tau \vdash e : \tau']\!]\rho[x \mapsto v] \in \mathcal{T}[\![\tau']\!]$. Clearly $\mathcal{C}[\![\Gamma \vdash (\lambda x : \tau.\, e) : \tau \to \tau']\!]\rho \in \mathcal{T}[\![\tau \to \tau']\!]$.

3 Operational Soundness of Typing Rules

With these denotational semantics, the meaning of a program is its type. ⟩From our semantics it is clear that all well-formed programs terminate and have a meaning. There is no possibility of error or nontermination. To show that the type system is sound with respect to the operational semantics,however, we need to show that programs don't get "stuck." Formally, we need to prove:

$\vdash e : \tau \ \wedge \ e \rightarrow^* e' \Rightarrow (e' \in \mathbf{Value} \ \vee \ \exists \mathbf{e''}.\ \mathbf{e'} \rightarrow \mathbf{e''})$

To do this, we need to prove the following:

- Preservation/Subject Reduction:
  $\vdash e : \tau \ \wedge \ e \rightarrow e' \ \Rightarrow \vdash e' : \tau$ – The type will be preserved after each step.
- Progress:
  $\vdash e : \tau \ \Rightarrow \ (e \in \mathbf{Value} \ \vee \ \exists \mathbf{e''} : \mathbf{e} \rightarrow \mathbf{e''})$ – If e is well-formed and is not a value then we can always take more steps.

These two properties can be used to show soundness by induction on the number of steps in the type derivation tree. Note that we can no longer write $(\lambda\ x\ (x\ x))(\lambda\ x\ (x\ x))$ in our language, so we have lost some expressive power through the requirement of well-formedness.

4 Preservation

We know that $\vdash e : \tau' \ \wedge \ e \rightarrow e'$ and we wish to show $\vdash e' : \tau'$. To do this we will need a stronger inductive hypothesis: $\Gamma \vdash e : \tau \ \wedge \ e \rightarrow e' \ \Rightarrow \ \Gamma \vdash e' : \tau$. Now since e can step to e', e must be an application. There are only three inference rules for application.

i. $$\frac{e_1 \rightarrow e_1'}{e_1\ e_2 \rightarrow e_1'\ e_2}$$

ii. $$\frac{e_2 \rightarrow e_2'}{e_1\ e_2 \rightarrow e_1\ e_2'}$$

iii. $(\lambda x : \tau.\, e)e' \rightarrow e\{e'/x\}$

Since $e = e_1\ e_2$ and $\vdash e_1\ e_2 : \tau'$, we need to prove $\vdash e_1'\ e_2 : \tau'$
The proof for the first inference rule is easy.
From the typing derivation we know that:

$$\frac{\Gamma \vdash e_1 : \tau \rightarrow \tau' \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1\ e_2 : \tau'}$$

and from our inference rules we know that:

$$\frac{e_1 \rightarrow e_1'}{e_1\ e_2 \rightarrow e_1'\ e_2}$$

Our induction hypothesis calmly assures us that e_1' will have the same type as e_1:
$\Gamma \vdash e_1 : \tau \rightarrow \tau' \ \Rightarrow \ \Gamma \vdash e_1' : \tau \rightarrow \tau'$
Therefore it is decreed that:

$$\frac{\Gamma \vdash e_1' : \tau \rightarrow \tau' \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1'\ e_2 : \tau'}$$

The proof for the second inference rule eerily resembles that of the first and is therefore rendered trivial.

The proof for the third rule is much more interesting:
$(\lambda x : \tau.\, e)e' \rightarrow e\{e'/x\}$
We know $\Gamma \vdash (\lambda x : \tau.\, e)e' : \tau' \ \wedge \ (\lambda x : \tau.\, e)e' \rightarrow e\{e'/x\}$ and hope to deduce $\Gamma \vdash e\{e'/x\} : \tau'$.

From the typing derivation, we know that:

$$\frac{\dfrac{\Gamma, x : \tau \vdash e : \tau'}{\Gamma \vdash (\lambda x : \tau.\, e) : \tau \to \tau'} \quad \Gamma \vdash e' : \tau}{(\lambda x : \tau.\, e) e' : \tau'}$$

Now using the magical typed Substitution Lemma:

$(\Gamma, x : \tau \vdash e : \tau') \wedge (\Gamma \vdash e' : \tau) \;\Rightarrow\; \Gamma \vdash e\{e'/x\} : \tau'$ – our induction is complete.

5 Substitution Lemma

We prove the substitution lemma by induction on the height of the typing derivation tree.
$(\Gamma, x : \tau \vdash e : \tau') \wedge (\Gamma \vdash e' : \tau) \;\Rightarrow\; \Gamma \vdash e\{e'/x\} : \tau'$

Case 1: x

$\Gamma, x : \tau \vdash x : \tau \;\wedge\; \Gamma \vdash e' : \tau \;\Rightarrow\; \Gamma \vdash x\{e'/x\} : \tau$ because $x\{e'/x\} = e' : \tau$

Case 2: $e_1\ e_2$

Because of the type derivation, we know that:

$$\frac{\Gamma, x : \tau \vdash e_1 : \tau'' \to \tau' \quad \Gamma, x : \tau \vdash e_2 : \tau''}{\Gamma, x : \tau \vdash e_1\ e_2 : \tau'}$$

According to the inductive hypothesis, e_1 and e_2 both satisfy the preservation property:

$\Gamma \vdash e_1\{e'/x\} : \tau'' \to \tau'$

$\Gamma \vdash e_2\{e'/x\} : \tau''$

Using the typing rule for application, it is plain to see that:

$$\frac{\Gamma \vdash e_1\{e'/x\} : \tau'' \to \tau' \quad \Gamma \vdash e_2\{e'/x\} : \tau''}{\Gamma \vdash e_1\{e'/x\}\ e_2\{e'/x\} : \tau'}$$

So we have $\Gamma \vdash e_1\{e'/x\}\ e_2\{e'/x\} : \tau'$. By definition of substitution, $\Gamma \vdash (e_1\ e_2)\{e'/x\} : \tau' \Leftrightarrow \Gamma \vdash e_1\{e'/x\}\ e_2\{e'/x\} : \tau'$.

Case 3: $(\lambda y : \tau''.\, e)$

We know:

$\Gamma, x : \tau \vdash (\lambda y : \tau''.\, e) : \tau'' \to \tau' \;\wedge\; \Gamma \vdash e' : \tau$

and we will be very happy if this implies:

$\Gamma \vdash (\lambda y : \tau''.\, e)\{e'/x\} : \tau'' \to \tau'$

The typing derivation has informed us that:

$$\frac{\Gamma, x : \tau, y : \tau'' \vdash e : \tau'}{\Gamma, x : \tau \vdash (\lambda y : \tau''.\, e) : \tau'' \to \tau'}$$

If $x = y$ then $(\lambda y : \tau''.\, e)\{e'/x\} = (\lambda y : \tau''.\, e)$. Since nothing changed, preservation is maintained.

If $x \neq y$ then we can commute: $\Gamma, x : \tau, y : \tau'' = \Gamma, y : \tau'', x : \tau$.

Using our inductive hypothesis and favorite neural pathways:

$\Gamma, y : \tau'' \vdash e\{e'/x\} : \tau'$

$\Gamma \vdash (\lambda y : \tau''.\, e\{e'/x\}) : \tau'' \to \tau'$ (applying the typing rule for abstraction terms)

$\Gamma \vdash (\lambda y : \tau''.\, e)\{e'/x\} : \tau'' \to \tau'$

And the lemma is proved!

6 Progress

This is our statement of progress:

$\vdash e : \tau \;\Rightarrow\; e \in Value \;\vee\; \exists e'', e \to e''$

We can prove this using structural induction.

If e is a value then we are done.

If not, then e must be an application: $e = e_1\ e_2$.

Therefore e_1 is either a λ-term or, according to the inductive hypothesis, $\exists e_1'$ such that $e_1 \rightarrow e_1'$ and so if $e' = e_1'\ e_2$ then $e \rightarrow e'$.
There are two choices now, either e_1 is a λ-term and e_2 is a value or e_1 is a λ-term and, by inductive hypothesis, $\exists e_2'$ such that $e_2 \rightarrow e_2'$.
If e_2 is not a value then let $e' = e_1\ e_2'$. Clearly $e \rightarrow e'$.
We know that e_1 is a lambda expression ($e_1 = (\lambda\ x\ e_3)$). If e_2 is a value v then $e = (\lambda\ x\ e_3)v$ and $e \rightarrow e_3\{v/x\} = e'$ and the induction is complete.