1 Announcements

Homework 2 is due today. Please turn them in. Homework 3 should be available on the web by the end of the class and is due this Thursday. This is a written homework. Homework 4 will be out this Thursday and will be due next Thursday.

2 Introduction

We gave the following call-by-value operational semantics for simply lambda calculus.
\[
t_1 \rightarrow t'_1 \\
\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \rightarrow \text{if } t'_1 \text{ then } t'_2 \text{ else } t_3 \\
\text{if true then } t_2 \text{ else } t_3 \rightarrow t_2 \\
\text{if false then } t_2 \text{ else } t_3 \rightarrow t_3 \\
t_1 \rightarrow t'_1 \\
t_1 t_2 \rightarrow t'_1 t_2 \\
t_2 \rightarrow t'_2 \\
t_1 t_2 \rightarrow t_1 t'_2 \\
(\lambda x : \tau.t) v \rightarrow [v/x] t
\]

We decided on the following type system. Intuitively we think that if a program (a term) is well typed with respect to the type system, then the operational will be able to evaluate the term to a value (assuming that the term does not cause infinite evaluation).

We now have a set of typing rules for simply typed lambda calculus. But how do we know that these typing rules make sense? Recall that the motivation here

### 3 Type Safety

We now have a set of typing rules for simply typed lambda calculus. But how do we know that these typing rules make sense? Recall that the motivation here
is to design a language in which well-typed terms do not get stuck. To prove this claim we prove type safety. How should we characterize type safety? What conditions do we need?

First remember that if we have a well typed term, then we want to be able to evaluate this term. Let’s write this down more precisely: If \( t : \tau \), then either \( t \) is a value, or \( t \rightarrow t' \).

We refer to this property as progress.

Note that progress property ties together types (the static semantics) and the operational semantics (dynamic semantics).

Does this suffice? May be, but by itself, this is not quite satisfactory, for example, you can have a semantics that evaluates a function to the constant true. Such a semantics “obviously” does not make sense. We will simply reject such semantics. Instead, we will insist that evaluation preserves typing. More precisely,

If \( \Gamma \vdash t : \tau \) and \( t \rightarrow t' \), then \( \Gamma \vdash t' : \tau \).

We refer to this property as preservation.

Let’s think about these properties together. They tell us that if we start with a program (a term) in our language, then either that term is a value, or the term can be reduced to another term without changing its type. The progress property is what makes types a crucial notion in programming. They establish types as an invariant that does not change through evaluation.

4 Proving Type Safety

We are now ready to prove type safety.

**Theorem 1 (Progress)**

Suppose \( \vdash t : \tau \). That is \( t \) is a well typed, closed term (of type \( \tau \)). Then either \( t \) is a value or \( t \rightarrow t' \) for some \( t' \).

**Proof:** The proof is by induction on the typing derivation of \( t : \tau \). As induction hypothesis, we will assume that the theorem holds for all subderivations of a typing derivation. Remember that induction on a derivation means that we can apply the induction hypothesis to subderivations of a derivation.

Let’s do a few of the cases.

**variable:** A variable cannot be well typed with respect to the empty context. The theorem hold vacuously.

**true/false:** We have \( \vdash \text{true} : \text{bool} \). Since true is a value the theorem holds. The case for false is similar.

\( \lambda x : \tau_1. t \): We have \( \vdash \lambda x : \tau_1. t : \tau_1 \rightarrow \tau_2 \). Since \( \lambda x : \tau_1. t \) is a value, the theorem holds.

(In general, the progress theorem holds for all values trivially.)
if \( t_1 \) then \( t_2 \) else \( t_3 \): Suppose we have if \( t_1 \) then \( t_2 \) else \( t_3 \) : \( \tau \), then we know by inversion that \( \vdash t_1 : \text{bool} \), \( \vdash t_2 : \tau \) and \( \vdash t_3 : \tau \). We can now apply the induction hypothesis with \( \vdash t_1 : \text{bool} \). By induction we know that \( t_1 \) is either a value, or \( t_1 \rightarrow t'_1 \). If \( t_1 \) is a value then by canonical forms, we know that \( t_1 = \text{true} \) or \( t_1 = \text{false} \). In this case, we know that

\[
\text{if true then } t_2 \text{ else } t_3 \rightarrow t_2
\]

or

\[
\text{if false then } t_2 \text{ else } t_3 \rightarrow t_3
\]

and thus the theorem holds. If \( t_1 \rightarrow t'_1 \), then we know that

\[
\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \rightarrow \text{if } t'_1 \text{ then } t_2 \text{ else } t_3
\]

and the theorem holds.

\( t_1 \) \( t_2 \): Suppose we have \( \vdash t_1 \ t_2 : \tau_2 \).

Then by inversion we know that \( \vdash t_1 : \tau_1 \rightarrow \tau_2 \) and \( \vdash t_2 : \tau_1 \). Now by induction \( t_1 \) is either a value or \( t_1 \rightarrow t'_1 \). If \( t_1 \rightarrow t'_1 \) then we know that \( t_1 \ t_2 \rightarrow t'_1 \ t_2 \) and the theorem holds.

If \( t_1 \) is a value, then by canonical forms lemma, we know that \( t_1 \) is a lambda term, i.e. \( t_1 = \lambda x : \tau_1 \ t \). Now consider \( t_2 \). By inversion we know that \( \vdash t_2 : \tau_1 \). Now by induction \( t_2 \) is either a value of \( t_2 \rightarrow t'_2 \). If \( t_2 \) is a value, then we have \( (\lambda x : \tau_1 \ t) t_2 \rightarrow [t_2/x] \ t \) and the theorem holds. If \( t_1 \rightarrow t'_2 \) then, we know that \( t_1 \ t_2 \rightarrow t_1 \ t'_2 \) and the theorem holds.

\[\square\]

**Theorem 2 (Preservation)**

If \( \Gamma \vdash t : \tau \) and \( t \rightarrow t' \) then \( t' : \tau \).

**Proof:** The proof is by induction on the typing derivation \( \Gamma \vdash t : \tau \). As induction hypothesis, we will assume that the theorem holds for the subderivations of the typing derivation.

The theorem holds for all values and variables vacuously because such terms does not take a step.

For the rest of the terms, we will consider each possible evaluation rule for \( t \rightarrow t' \).

- **if true then \( t_2 \) else \( t_3 \) \( \rightarrow \) \( t_2 \).** We have to show that \( \Gamma \vdash t_2 : \tau \). This follows directly by inversion.

- **if false then \( t_2 \) else \( t_3 \) \( \rightarrow \) \( t_3 \).** We have to show that \( \Gamma \vdash t_3 : \tau \). This follows directly by inversion.
\[ t_1 \rightarrow t'_1 \]

\[
\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \rightarrow \text{ if } t'_1 \text{ then } t_2 \text{ else } t_3
\]

We know that
\[
\Gamma \vdash t_1 : \text{bool} \quad \Gamma \vdash t_2 : \tau \quad \Gamma \vdash t_3 : \tau
\]

Since we know that \( t_1 \rightarrow t'_1 \) and we have the derivation for \( \Gamma \vdash t_1 : \text{bool} \), we can apply induction to obtain \( \Gamma \vdash t'_1 : \text{bool} \). We then know that
\[
\Gamma \vdash t'_1 : \text{bool} \quad \Gamma \vdash t_2 : \tau \quad \Gamma \vdash t_3 : \tau
\]

Let’s apply induction to \( \Gamma \vdash t_2 : \tau \) and \( t_1 \rightarrow t'_1 \). We know that \( \Gamma \vdash t'_1 : \tau_1 \rightarrow \tau_2 \) and \( t_1 \rightarrow t'_1 \). But then we have
\[
\Gamma \vdash t'_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash t_2 : \tau_1
\]

This shows that theorem holds.

Thus the theorem holds.

\[ t_1 \rightarrow t'_1 \]

\[
\frac{t_1 \rightarrow t'_1}{t_1 t_2 \rightarrow t'_1 t_2}
\]

\[
\Gamma \vdash t_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash t_2 : \tau_1
\]

\[
\Gamma \vdash t_1 t_2 : \tau_2
\]

Let’s apply induction to \( \Gamma \vdash t_2 : \tau_1 \) and \( t_2 \rightarrow t'_2 \). We know that \( \Gamma \vdash t'_2 : \tau_1 \). But then we have
\[
\Gamma \vdash t_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash t'_2 : \tau_1
\]

Thus the theorem holds.
\[(\lambda x : \tau_1 . t) \, v \rightarrow [v/x] \, t\]

In this case, we have to show that \(\Gamma \vdash [v/x] \, t : \tau\). For this, we need to prove a “substitution” lemma. This case will follow directly from substitution lemma.

\[\square\]

**Lemma 3 (Substitution)**

*If* \(\Gamma, x : \tau_1 \vdash t : \tau_2\) *and* \(\Gamma \vdash t_1 : \tau_1\) *then* \(\Gamma \vdash [t_1/x] \, t : \tau_2\).

**Proof:**

The proof on the typing derivation \(\Gamma, x : \tau_1 \vdash t : \tau_2\). We will consider each possible typing rule for the final typing judgement.

- The term \(t\) is \textit{true} or \textit{false}. These are very similar consider \(t = \text{true}\).

  \[\Gamma, x : \tau_1 \vdash \text{true} : \text{bool}\]

  Since \([t_1/x] \, \text{true} = \text{true}\) by the definition of substitution, we know that \(\Gamma, x : \tau_1 \vdash \text{true} : \text{bool}\). Since \textit{true} has no free variables, it follows that \(\Gamma \vdash \text{true} : \text{bool}\).

- The term \(t\) is the variable \(y\).

  \[\frac{(y, \tau_2) \in (\Gamma, x : \tau_1)}{\Gamma, x : \tau_1 \vdash y : \tau_2}\]

  Consider now the substituted term \([t_1/x] \, y\). We have two cases to consider.

  In the first case, we have \(x = y\) and \([t_1/x] \, y = t_1\). Note that since \(x = y\), we have \(\tau_1 = \tau_2\). To prove the lemma we have to show that \(\Gamma \vdash t_1 : \tau_1\). This follows immediately by the assumption.

  In this second case, we have \(x \neq y\) and \([t_1/x] \, y = y\). In this case, we have to show that \(\Gamma \vdash y : \tau_2\). Since we know that \((y, \tau_2) \in (\Gamma, x : \tau_1)\) and \(x \neq y\), we have \((y, \tau_2) \in \Gamma\), i.e., \(\Gamma \vdash y : \tau_2\).

- The term \(t\) is a conditional.

  \[\frac{\Gamma, x : \tau_1 \vdash t_2 : \text{bool} \quad \Gamma, x : \tau_1 \vdash t_3 : \tau \quad \Gamma, x : \tau_1 \vdash t_4 : \tau}{\Gamma, x : \tau_1 \vdash \text{if} \, t_2 \, \text{then} \, t_3 \, \text{else} \, t_4 : \tau}\]
We can now apply induction to each of the subderivations to obtain $\Gamma \vdash [t_1/x] t_2 : \text{bool}$, $\Gamma \vdash [t_1/x] t_3 : \tau$, and $\Gamma \vdash [t_1/x] t_4 : \tau$. We then know that

$$\Gamma \vdash [t_1/x] t_2 : \text{bool} \quad \Gamma \vdash [t_1/x] t_3 : \tau \quad \Gamma \vdash [t_1/x] t_4 : \tau$$

Since if $[t_1/x] t_2$ then $[t_1/x] t_3$ else $[t_1/x] t_4 = [t_1/x] \text{if} t_2 \text{ then } t_3 \text{ else } t_4$, the lemma follows.

- The term $t$ is an application.

$$\Gamma, x : \tau_1 \vdash t_2 : \tau_{21} \Rightarrow \tau_{22} \quad \Gamma, x : \tau_2 \vdash t_1 : \tau_{21}$$

We can now apply induction to each of the subderivations to obtain $\Gamma \vdash [t_1/x] t_2 : \tau_{21} \Rightarrow \tau_{22}$ and $\Gamma \vdash [t_1/x] t_2 : \tau_{21}$. But then we have

$$\Gamma \vdash ([t_1/x] t_2) : \tau_{21} \Rightarrow \tau_{22} \quad \Gamma \vdash ([t_1/x] t_21) : \tau_{21}$$

Since $([t_1/x] t_2) ([t_1/x] t_21) = [t_1/x] (t_2 t_21)$ the lemma holds.

- The term $t$ is a lambda abstraction.

$$\Gamma, x : \tau_1, y : \tau_{21} \vdash t_2 : \tau_{22} \quad \Gamma, x : \tau_1 \vdash \lambda y : \tau_{21}. t_2 : \tau_{21} \Rightarrow \tau_{22}$$

By lambda conversion and capture avoiding property of substitutions, we know that $y \notin \text{dom}(\Gamma) \cup \{x\}$ and $y \notin \text{FV}(t_1)$. We have to show that $\Gamma \vdash [t_1/x](\lambda y : \tau_{21}. t_2) : \tau_{21} \Rightarrow \tau_{22}$ assuming that $\Gamma \vdash t_1 : \tau_1$.

By weakening, we know that $\Gamma, y : \tau_{21} \vdash t_1 : \tau_1$. Knowing this, we apply induction to the derivation $\Gamma, y : \tau_{21}, x : \tau_1 \vdash t_2 : \tau_{22}$ which we obtain by permutation of the context. By induction, we know that $\Gamma, y : \tau_{21} \vdash [t_1/x] t_2 : \tau_{22}$.

But then we know that

$$\Gamma, y : \tau_{21} \vdash [t_1/x] t_2 : \tau_{22} \quad \Gamma \vdash \lambda y : \tau_{21}. t_2 : \tau_{21} \Rightarrow \tau_{22}$$

Since we know that $[t_1/x](\lambda y : \tau_{21}. t_2) = \lambda y : \tau_{21}. [t_1/x] t_2$ by the definition of substitution. The lemma holds.

\[ \square \]