Formal Methods

Lecture 7

(B. Pierce's slides for the book "Types and Programming Languages")

Types

Plan

- For today, we'll go back to the simple language of arithmetic and boolean expressions and show how to equip it with a (very simple) type system
- The key property of this type system will be *soundness*: Well-typed programs do not get stuck
- Next time, we'll develop a simple type system for the lambda-calculus
- We'll spend a good part of the rest of the semester adding features to this type system

Outline

1. begin with a set of terms, a set of values, and an evaluation relation
2. define a set of *types* classifying values according to their “shapes”
3. define a *typing relation* $\mathrm{t} : \mathrm{T}$ that classifies terms according to the shape of the values that result from evaluating them
4. check that the typing relation is *sound* in the sense that,
   4.1 if $\mathrm{t} : \mathrm{T}$ and $\mathrm{t} \longrightarrow^{*} \mathrm{v}$, then $\mathrm{v} : \mathrm{T}$
   4.2 if $\mathrm{t} : \mathrm{T}$, then evaluation of t will not get stuck

Review: Arithmetic Expressions – Syntax

`t ::=`		*terms*
	`true`	*constant true*
	`false`	*constant false*
	`if t then t else t`	*conditional*
	`0`	*constant zero*
	`succ t`	*successor*
	`pred t`	*predecessor*
	`iszero t`	*zero test*
`v ::=`		*values*
	`true`	*true value*
	`false`	*false value*
	`nv`	*numeric value*
`nv ::=`		*numeric values*
	`0`	*zero value*
	`succ nv`	*successor value*

Evaluation Rules

$$\texttt{if true then } t_2 \texttt{ else } t_3 \longrightarrow t_2 \qquad \text{(E-IfTrue)}$$

$$\texttt{if false then } t_2 \texttt{ else } t_3 \longrightarrow t_3 \qquad \text{(E-IfFalse)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 \longrightarrow \texttt{if } t_1' \texttt{ then } t_2 \texttt{ else } t_3} \qquad \text{(E-If)}$$

$$\frac{t_1 \longrightarrow t_1'}{\text{succ } t_1 \longrightarrow \text{succ } t_1'} \quad \text{(E-Succ)}$$

$$\text{pred } 0 \longrightarrow 0 \quad \text{(E-PredZero)}$$

$$\text{pred } (\text{succ } nv_1) \longrightarrow nv_1 \quad \text{(E-PredSucc)}$$

$$\frac{t_1 \longrightarrow t_1'}{\text{pred } t_1 \longrightarrow \text{pred } t_1'} \quad \text{(E-Pred)}$$

$$\text{iszero } 0 \longrightarrow \text{true} \quad \text{(E-IszeroZero)}$$

$$\text{iszero } (\text{succ } nv_1) \longrightarrow \text{false} \quad \text{(E-IszeroSucc)}$$

$$\frac{t_1 \longrightarrow t_1'}{\text{iszero } t_1 \longrightarrow \text{iszero } t_1'} \quad \text{(E-IsZero)}$$

Types

In this language, values have two possible “shapes”: they are either booleans or numbers.

T ::=		*types*
	Bool	*type of booleans*
	Nat	*type of numbers*

Typing Rules

$$\text{true} : \text{Bool} \qquad \text{(T-True)}$$

$$\text{false} : \text{Bool} \qquad \text{(T-False)}$$

$$\frac{t_1 : \text{Bool} \quad t_2 : T \quad t_3 : T}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T} \qquad \text{(T-If)}$$

$$0 : \text{Nat} \qquad \text{(T-Zero)}$$

$$\frac{t_1 : \text{Nat}}{\text{succ } t_1 : \text{Nat}} \qquad \text{(T-Succ)}$$

$$\frac{t_1 : \text{Nat}}{\text{pred } t_1 : \text{Nat}} \qquad \text{(T-Pred)}$$

$$\frac{t_1 : \text{Nat}}{\text{iszero } t_1 : \text{Bool}} \qquad \text{(T-IsZero)}$$

Typing Derivations

Every pair $(t\,,\,T)$ in the typing relation can be justified by a *derivation tree* built from instances of the inference rules.

$$\dfrac{\dfrac{\dfrac{}{0 : \mathtt{Nat}}\;\text{T-Zero}}{\mathtt{iszero}\ 0 : \mathtt{Bool}}\;\text{T-IsZero} \qquad \dfrac{}{0 : \mathtt{Nat}}\;\text{T-Zero} \qquad \dfrac{\dfrac{}{0 : \mathtt{Nat}}\;\text{T-Zero}}{\mathtt{pred}\ 0 : \mathtt{Nat}}\;\text{T-Pred}}{\mathtt{if}\ \mathtt{iszero}\ 0\ \mathtt{then}\ 0\ \mathtt{else}\ \mathtt{pred}\ 0 : \mathtt{Nat}}\;\text{T-If}$$

Proofs of properties about the typing relation often proceed by induction on typing derivations.

Imprecision of Typing

Like other static program analyses, type systems are generally *imprecise*: they do not predict exactly what kind of value will be returned by every program, but just a conservative (safe) approximation.

$$\frac{t_1 : \mathrm{Bool} \qquad t_2 : T \qquad t_3 : T}{\mathrm{if}\ t_1\ \mathrm{then}\ t_2\ \mathrm{else}\ t_3 : T} \qquad \text{(T-If)}$$

Using this rule, we cannot assign a type to

$$\mathrm{if\ \ true\ \ then\ \ 0\ \ else\ \ false}$$

even though this term will certainly evaluate to a number.

Properties of the Typing Relation

Type Safety

The safety (or soundness) of this type system can be expressed by two properties:

1. *Progress*: A well-typed term is not stuck

   If $t : T$*, then either* t *is a value or else* $t \longrightarrow t'$ *for some* t'.

2. *Preservation*: Types are preserved by one-step evaluation

   If $t : T$ *and* $t \longrightarrow t'$*, then* $t' : T$.

Inversion

Lemma:

1. If $\mathtt{true} : R$, then $R = \mathtt{Bool}$.
2. If $\mathtt{false} : R$, then $R = \mathtt{Bool}$.
3. If $\mathtt{if}\ t_1\ \mathtt{then}\ t_2\ \mathtt{else}\ t_3 : R$, then $t_1 : \mathtt{Bool}$, $t_2 : R$, and $t_3 : R$.
4. If $0 : R$, then $R = \mathtt{Nat}$.
5. If $\mathtt{succ}\ t_1 : R$, then $R = \mathtt{Nat}$ and $t_1 : \mathtt{Nat}$.
6. If $\mathtt{pred}\ t_1 : R$, then $R = \mathtt{Nat}$ and $t_1 : \mathtt{Nat}$.
7. If $\mathtt{iszero}\ t_1 : R$, then $R = \mathtt{Bool}$ and $t_1 : \mathtt{Nat}$.

Inversion

Lemma:

1. If $\mathtt{true} : R$, then $R = \mathtt{Bool}$.
2. If $\mathtt{false} : R$, then $R = \mathtt{Bool}$.
3. If $\mathtt{if}\ t_1\ \mathtt{then}\ t_2\ \mathtt{else}\ t_3 : R$, then $t_1 : \mathtt{Bool}$, $t_2 : R$, and $t_3 : R$.
4. If $0 : R$, then $R = \mathtt{Nat}$.
5. If $\mathtt{succ}\ t_1 : R$, then $R = \mathtt{Nat}$ and $t_1 : \mathtt{Nat}$.
6. If $\mathtt{pred}\ t_1 : R$, then $R = \mathtt{Nat}$ and $t_1 : \mathtt{Nat}$.
7. If $\mathtt{iszero}\ t_1 : R$, then $R = \mathtt{Bool}$ and $t_1 : \mathtt{Nat}$.

Proof: ...

This leads directly to a recursive algorithm for calculating the type of a term...

Typechecking Algorithm

```
typeof(t) = if t = true then Bool
              else if t = false then Bool
            else if t = if t1 then t2 else t3 then
              let T1 = typeof(t1) in
              let T2 = typeof(t2) in let
               T3 = typeof(t3) in
              if T1 = Bool and T2=T3 then T2
              else "not typable"
            else if t = 0 then Nat else
            if t = succ t1 then
              let T1 = typeof(t1) in
              if T1 = Nat then Nat else "not typable"
            else if t = pred t1 then
              let T1 = typeof(t1) in
              if T1 = Nat then Nat else "not typable"
            else if t = iszero t1 then
              let T1 = typeof(t1) in
              if T1 = Nat then Bool else "not typable"
```

Canonical Forms

Lemma:

1. If v is a value of type Bool, then v is either $\texttt{true}$ or $\texttt{false}$.
2. If v is a value of type Nat, then v is a numeric value.

Proof:

Canonical Forms

Lemma:

1. If `v` is a value of type `Bool`, then `v` is either `true` or `false`.
2. If `v` is a value of type `Nat`, then `v` is a numeric value.

Proof: Recall the syntax of values:

`v ::=`		*values*
	`true`	*true value*
	`false`	*false value*
	`nv`	*numeric value*
`nv ::=`		*numeric values*
	`0`	*zero value*
	`succ nv`	*successor value*

For part 1, if `v` is `true` or `false`, the result is immediate. But `v` cannot be `0` or `succ nv`, since the inversion lemma tells us that `v` would then have type `Nat`, not `Bool`. Part 2 is similar.

Progress

Theorem: Suppose t is a well-typed term (that is, $t : T$ for some T). Then either t is a value or else there is some t' with $t \longrightarrow t'$.

Proof: By induction on a derivation of $t : T$.

Progress

Theorem: Suppose t is a well-typed term (that is, $t : T$ for some T). Then either t is a value or else there is some t' with $t \longrightarrow t'$.

Proof: By induction on a derivation of $t : T$.

The T-True, T-False, and T-Zero cases are immediate, since t in these cases is a value.

Progress

Theorem: Suppose $\mathtt{t}$ is a well-typed term (that is, $\mathtt{t} : \mathtt{T}$ for some $\mathtt{T}$). Then either $\mathtt{t}$ is a value or else there is some $\mathtt{t}'$ with $\mathtt{t} \longrightarrow \mathtt{t}'$.

Proof: By induction on a derivation of $\mathtt{t} : \mathtt{T}$.

The T-True, T-False, and T-Zero cases are immediate, since $\mathtt{t}$ in these cases is a value.

Case T-If: $\mathtt{t} = \mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3$

$\mathtt{t}_1 : \mathtt{Bool} \qquad \mathtt{t}_2 : \mathtt{T} \qquad \mathtt{t}_3 : \mathtt{T}$

By the induction hypothesis, either $\mathtt{t}_1$ is a value or else there is some $\mathtt{t}_1'$ such that $\mathtt{t}_1 \longrightarrow \mathtt{t}_1'$. If $\mathtt{t}_1$ is a value, then the canonical forms lemma tells us that it must be either $\mathtt{true}$ or $\mathtt{false}$, in which case either E-IfTrue or E-IfFalse applies to $\mathtt{t}$. On the other hand, if $\mathtt{t}_1 \longrightarrow \mathtt{t}_1'$, then, by E-If,

$\mathtt{t} \longrightarrow \mathtt{if}\ \mathtt{t}_1'\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3$.

Progress

Theorem: Suppose $\mathtt{t}$ is a well-typed term (that is, $\mathtt{t} : \mathtt{T}$ for some type $\mathtt{T}$). Then either $\mathtt{t}$ is a value or else there is some $\mathtt{t}'$ with $\mathtt{t} \longrightarrow \mathtt{t}'$.

Proof: By induction on a derivation of $\mathtt{t} : \mathtt{T}$.

The cases for rules T-Zero, T-Succ, T-Pred, and T-IsZero are similar.

(Recommended: Try to reconstruct them.)

Preservation

Theorem: If $t : T$ and $t \longrightarrow t'$, then $t' : T$.

Proof: By induction on the given typing derivation.

Preservation

Theorem: If $t : T$ and $t \longrightarrow t'$, then $t' : T$.

Proof: By induction on the given typing derivation.

Case T-True: $\quad t = \text{true} \qquad T = \text{Bool}$

Then t is a value, so it cannot be that $t \longrightarrow t'$ for any t', and the theorem is vacuously true.

Preservation

Theorem: If $t : T$ and $t \longrightarrow t'$, then $t' : T$.

Proof: By induction on the given typing derivation.

Case T-If:

$t = \text{if } t_1 \text{ then } t_2 \text{ else } t_3 \quad t_1 : \text{Bool} \quad t_2 : T \quad t_3 : T$

There are three evaluation rules by which $t \longrightarrow t'$ can be derived:
E-IfTrue, E-IfFalse, and E-If. Consider each case separately.

Preservation

Theorem: If $\mathtt{t} : \mathtt{T}$ and $\mathtt{t} \longrightarrow \mathtt{t}'$, then $\mathtt{t}' : \mathtt{T}$.

Proof: By induction on the given typing derivation.

Case T-If:

$\mathtt{t} = \mathtt{if}\ \mathtt{t_1}\ \mathtt{then}\ \mathtt{t_2}\ \mathtt{else}\ \mathtt{t_3} \quad \mathtt{t_1} : \mathtt{Bool} \quad \mathtt{t_2} : \mathtt{T} \quad \mathtt{t_3} : \mathtt{T}$

There are three evaluation rules by which $\mathtt{t} \longrightarrow \mathtt{t}'$ can be derived: E-IfTrue, E-IfFalse, and E-If. Consider each case separately.

Subcase E-IfTrue: $\mathtt{t_1} = \mathtt{true} \quad \mathtt{t}' = \mathtt{t_2}$

Immediate, by the assumption $\mathtt{t_2} : \mathtt{T}$.

(E-IfFalse subcase: Similar.)

Preservation

Theorem: If $t : T$ and $t \longrightarrow t'$, then $t' : T$.

Proof: By induction on the given typing derivation.

Case T-If:

$t = \text{if } t_1 \text{ then } t_2 \text{ else } t_3 \quad t_1 : \text{Bool} \quad t_2 : T \quad t_3 : T$

There are three evaluation rules by which $t \longrightarrow t'$ can be derived:
E-IfTrue, E-IfFalse, and E-If. Consider each case separately.

Subcase E-If: $\quad t_1 \longrightarrow t_1' \quad t' = \text{if } t_1' \text{ then } t_2 \text{ else } t_3$

Applying the IH to the subderivation of $t_1 : \text{Bool}$ yields $t_1' : \text{Bool}$. Combining this with the assumptions that $t_2 : T$ and $t_3 : T$, we can apply rule T-If to conclude that $\text{if } t_1' \text{ then } t_2 \text{ else } t_3 : T$, that is, $t' : T$.

Recap: Type Systems

- Very successful example of a *lightweight formal method*
- big topic in PL research
- enabling technology for all sorts of other things, e.g. language-based security
- the skeleton around which modern programming languages are designed

The Simply Typed Lambda-Calculus

The simply typed lambda-calculus

The system we are about to define is commonly called the *simply typed lambda-calculus*, or $\lambda_{\rightarrow}$ for short.

Unlike the untyped lambda-calculus, the "pure" form of $\lambda_{\rightarrow}$ (with no primitive values or operations) is not very interesting; to talk about $\lambda_{\rightarrow}$, we always begin with some set of "base types."

- So, strictly speaking, there are *many* variants of $\lambda_{\rightarrow}$, depending on the choice of base types.
- For now, we'll work with a variant constructed over the booleans.

Untyped lambda-calculus with booleans

t ::=		*terms*
	x	*variable*
	λx.t	*abstraction*
	t t	*application*
	true	*constant true*
	false	*constant false*
	if t then t else t	*conditional*
v ::=		*values*
	λx.t	*abstraction value*
	true	*true value*
	false	*false value*

"Simple Types"

T ::=		*types*
	Bool	*type of booleans*
	$T \rightarrow T$	*types of functions*

Type Annotations

We now have a choice to make. Do we...

- annotate lambda-abstractions with the expected type of the argument

  $$\lambda x{:}T_1.\ t_2$$

  (as in most mainstream programming languages), or
- continue to write lambda-abstractions as before

  $$\lambda x.\ t_2$$

  and ask the typing rules to “guess” an appropriate annotation (as in OCaml)?

Both are reasonable choices, but the first makes the job of defining the typing rules simpler. Let’s take this choice for now.

Typing rules

$$\text{true} : \text{Bool} \qquad \text{(T-True)}$$

$$\text{false} : \text{Bool} \qquad \text{(T-False)}$$

$$\frac{t_1 : \text{Bool} \qquad t_2 : T \qquad t_3 : T}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T} \qquad \text{(T-If)}$$

Typing rules

$$\text{true} : \text{Bool} \qquad \text{(T-True)}$$

$$\text{false} : \text{Bool} \qquad \text{(T-False)}$$

$$\frac{t_1 : \text{Bool} \qquad t_2 : T \qquad t_3 : T}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T} \qquad \text{(T-If)}$$

$$\frac{???}{\lambda x{:}T_1.t_2 : T_1 \to T_2} \qquad \text{(T-Abs)}$$

Typing rules

$$\text{true} : \text{Bool} \qquad \text{(T-True)}$$

$$\text{false} : \text{Bool} \qquad \text{(T-False)}$$

$$\frac{t_1 : \text{Bool} \quad t_2 : T \quad t_3 : T}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T} \qquad \text{(T-If)}$$

$$\frac{\Gamma, x{:}T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x{:}T_1.\, t_2 : T_1 \to T_2} \qquad \text{(T-Abs)}$$

$$\frac{x{:}T \in \Gamma}{\Gamma \vdash x : T} \qquad \text{(T-Var)}$$

Typing rules

$$\Gamma \vdash \mathrm{true} : \mathrm{Bool} \qquad \text{(T-True)}$$

$$\Gamma \vdash \mathrm{false} : \mathrm{Bool} \qquad \text{(T-False)}$$

$$\frac{\Gamma \vdash t_1 : \mathrm{Bool} \quad \Gamma \vdash t_2 : T \quad \Gamma \vdash t_3 : T}{\Gamma \vdash \mathrm{if}\ t_1\ \mathrm{then}\ t_2\ \mathrm{else}\ t_3 : T} \qquad \text{(T-If)}$$

$$\frac{\Gamma, x{:}T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x{:}T_1.t_2 : T_1 \to T_2} \qquad \text{(T-Abs)}$$

$$\frac{x{:}T \in \Gamma}{\Gamma \vdash x : T} \qquad \text{(T-Var)}$$

$$\frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \quad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \qquad \text{(T-App)}$$

Typing Derivations

What derivations justify the following typing statements?

- $\vdash (\lambda x{:}\text{Bool}.x)\ \text{true} : \text{Bool}$
- $f{:}\text{Bool}\to\text{Bool} \vdash f\ (\text{if false then true else false}) : \text{Bool}$
- $f{:}\text{Bool}\to\text{Bool} \vdash \lambda x{:}\text{Bool}.\ f\ (\text{if } x \text{ then false else } x) : \text{Bool}\to\text{Bool}$

Properties of $\lambda_{\rightarrow}$

The fundamental property of the type system we have just defined is *soundness* with respect to the operational semantics.

1. *Progress*: A closed, well-typed term is not stuck

   If $\vdash t : T$, *then either* t *is a value or else* $t \longrightarrow t'$ *for some* t'.

2. *Preservation*: Types are preserved by one-step evaluation

   If $\Gamma \vdash t : T$ *and* $t \longrightarrow t'$, *then* $\Gamma \vdash t' : T$.

Proving progress

Same steps as before...

- inversion lemma for typing relation
- canonical forms lemma
- progress theorem

Inversion

Lemma:

1. If $\Gamma \vdash \texttt{true} : \texttt{R}$, then $\texttt{R} = \texttt{Bool}$.
2. If $\Gamma \vdash \texttt{false} : \texttt{R}$, then $\texttt{R} = \texttt{Bool}$.
3. If $\Gamma \vdash \texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 : \texttt{R}$, then $\Gamma \vdash \texttt{t}_1 : \texttt{Bool}$ and $\Gamma \vdash \texttt{t}_2, \texttt{t}_3 : \texttt{R}$.

Inversion

Lemma:

1. If $\Gamma \vdash \mathtt{true} : R$, then $R = \mathtt{Bool}$.
2. If $\Gamma \vdash \mathtt{false} : R$, then $R = \mathtt{Bool}$.
3. If $\Gamma \vdash \mathtt{if}\ t_1\ \mathtt{then}\ t_2\ \mathtt{else}\ t_3 : R$, then $\Gamma \vdash t_1 : \mathtt{Bool}$ and $\Gamma \vdash t_2, t_3 : R$.
4. If $\Gamma \vdash x : R$, then

Inversion

Lemma:

1. If $\Gamma \vdash \texttt{true} : R$, then $R = \texttt{Bool}$.
2. If $\Gamma \vdash \texttt{false} : R$, then $R = \texttt{Bool}$.
3. If $\Gamma \vdash \texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 : R$, then $\Gamma \vdash t_1 : \texttt{Bool}$ and $\Gamma \vdash t_2, t_3 : R$.
4. If $\Gamma \vdash x : R$, then $x{:}R \in \Gamma$.

Inversion

Lemma:

1. If $\Gamma \vdash \texttt{true} : \texttt{R}$, then $\texttt{R} = \texttt{Bool}$.
2. If $\Gamma \vdash \texttt{false} : \texttt{R}$, then $\texttt{R} = \texttt{Bool}$.
3. If $\Gamma \vdash \texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 : \texttt{R}$, then $\Gamma \vdash t_1 : \texttt{Bool}$ and $\Gamma \vdash t_2, t_3 : \texttt{R}$.
4. If $\Gamma \vdash \texttt{x} : \texttt{R}$, then $\texttt{x:R} \in \Gamma$.
5. If $\Gamma \vdash \lambda \texttt{x:T}_1\texttt{.}t_2 : \texttt{R}$, then

Inversion

Lemma:

1. If $\Gamma \vdash \texttt{true} : R$, then $R = \texttt{Bool}$.
2. If $\Gamma \vdash \texttt{false} : R$, then $R = \texttt{Bool}$.
3. If $\Gamma \vdash \texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 : R$, then $\Gamma \vdash t_1 : \texttt{Bool}$ and $\Gamma \vdash t_2, t_3 : R$.
4. If $\Gamma \vdash x : R$, then $x{:}R \in \Gamma$.
5. If $\Gamma \vdash \lambda x{:}T_1.t_2 : R$, then $R = T_1 \rightarrow R_2$ for some R_2 with $\Gamma, x{:}T_1 \vdash t_2 : R_2$.

Inversion

Lemma:

1. If $\Gamma \vdash \mathtt{true} : R$, then $R = \mathtt{Bool}$.
2. If $\Gamma \vdash \mathtt{false} : R$, then $R = \mathtt{Bool}$.
3. If $\Gamma \vdash \mathtt{if}\ t_1\ \mathtt{then}\ t_2\ \mathtt{else}\ t_3 : R$, then $\Gamma \vdash t_1 : \mathtt{Bool}$ and $\Gamma \vdash t_2, t_3 : R$.
4. If $\Gamma \vdash x : R$, then $x{:}R \in \Gamma$.
5. If $\Gamma \vdash \lambda x{:}T_1.t_2 : R$, then $R = T_1 \to R_2$ for some R_2 with $\Gamma, x{:}T_1 \vdash t_2 : R_2$.
6. If $\Gamma \vdash t_1\ t_2 : R$, then

Inversion

Lemma:

1. If $\Gamma \vdash \texttt{true} : \texttt{R}$, then $\texttt{R} = \texttt{Bool}$.
2. If $\Gamma \vdash \texttt{false} : \texttt{R}$, then $\texttt{R} = \texttt{Bool}$.
3. If $\Gamma \vdash \texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 : \texttt{R}$, then $\Gamma \vdash \texttt{t}_1 : \texttt{Bool}$ and $\Gamma \vdash \texttt{t}_2, \texttt{t}_3 : \texttt{R}$.
4. If $\Gamma \vdash \texttt{x} : \texttt{R}$, then $\texttt{x:R} \in \Gamma$.
5. If $\Gamma \vdash \lambda\texttt{x:T}_1.\texttt{t}_2 : \texttt{R}$, then $\texttt{R} = \texttt{T}_1 \to \texttt{R}_2$ for some $\texttt{R}_2$ with $\Gamma, \texttt{x:T}_1 \vdash \texttt{t}_2 : \texttt{R}_2$.
6. If $\Gamma \vdash \texttt{t}_1\ \texttt{t}_2 : \texttt{R}$, then there is some type $\texttt{T}_{11}$ such that $\Gamma \vdash \texttt{t}_1 : \texttt{T}_{11} \to \texttt{R}$ and $\Gamma \vdash \texttt{t}_2 : \texttt{T}_{11}$.

Canonical Forms

Lemma:

1. If `v` is a value of type `Bool`, then `v` is either `true` or `false`.

Canonical Forms

Lemma:

1. If v is a value of type Bool, then v is either true or false.
2. If v is a value of type $T_1 \to T_2$, then v has the form $\lambda x{:}T_1.\,t_2$.

Progress

Theorem: Suppose t is a closed, well-typed term (that is, $\vdash t : T$ for some T). Then either t is a value or else there is some t' with $t \longrightarrow t'$.

Proof: By induction on typing derivations.

Progress

Theorem: Suppose t is a closed, well-typed term (that is, $\vdash t : T$ for some T). Then either t is a value or else there is some t' with $t \longrightarrow t'$.

Proof: By induction on typing derivations. The cases for boolean constants and conditions are the same as before. The variable case is trivial (because t is closed). The abstraction case is immediate, since abstractions are values.

Progress

Theorem: Suppose t is a closed, well-typed term (that is, $\vdash t : T$ for some T). Then either t is a value or else there is some t' with $t \longrightarrow t'$.

Proof: By induction on typing derivations. The cases for boolean constants and conditions are the same as before. The variable case is trivial (because t is closed). The abstraction case is immediate, since abstractions are values.
Consider the case for application, where $t = t_1\ t_2$ with $\vdash t_1 : T_{11} \to T_{12}$ and $\vdash t_2 : T_{11}$.

Progress

Theorem: Suppose t is a closed, well-typed term (that is, $\vdash t : T$ for some T). Then either t is a value or else there is some t' with $t \longrightarrow t'$.

Proof: By induction on typing derivations. The cases for boolean constants and conditions are the same as before. The variable case is trivial (because t is closed). The abstraction case is immediate, since abstractions are values.
Consider the case for application, where $t = t_1 \; t_2$ with $\vdash t_1 : T_{11} \rightarrow T_{12}$ and $\vdash t_2 : T_{11}$. By the induction hypothesis, either t_1 is a value or else it can make a step of evaluation, and likewise t_2.

Progress

Theorem: Suppose t is a closed, well-typed term (that is, $\vdash t : T$ for some T). Then either t is a value or else there is some t' with $t \longrightarrow t'$.

Proof: By induction on typing derivations. The cases for boolean constants and conditions are the same as before. The variable case is trivial (because t is closed). The abstraction case is immediate, since abstractions are values.
Consider the case for application, where $t = t_1\ t_2$ with $\vdash t_1 : T_{11} \to T_{12}$ and $\vdash t_2 : T_{11}$. By the induction hypothesis, either t_1 is a value or else it can make a step of evaluation, and likewise t_2. If t_1 can take a step, then rule E-App1 applies to t. If t_1 is a value and t_2 can take a step, then rule E-App2 applies. Finally, if both t_1 and t_2 are values, then the canonical forms lemma tells us that t_1 has the form $\lambda x{:}T_{11}.\,t_{12}$, and so rule E-AppAbs applies to t.

Preservation

Theorem: If $\Gamma \vdash t : T$ and $t \longrightarrow t'$, then $\Gamma \vdash t' : T$.

Steps of proof:

- Weakening
- Permutation
- Substitution preserves types
- Reduction preserves types (i.e., preservation)

Weakening and Permutation

Weakening tells us that we can *add assumptions* to the context without losing any true typing statements.

Lemma: If $\Gamma \vdash t : T$ and $x \notin dom(\Gamma)$, then $\Gamma, x{:}S \vdash t : T$.

Weakening and Permutation

Weakening tells us that we can *add assumptions* to the context without losing any true typing statements.

Lemma: If $\Gamma \vdash t : T$ and $x \notin dom(\Gamma)$, then $\Gamma, x{:}S \vdash t : T$.

Permutation tells us that the order of assumptions in (the list) Γ does not matter.

Lemma: If $\Gamma \vdash t : T$ and Δ is a permutation of Γ, then $\Delta \vdash t : T$.

Weakening and Permutation

Weakening tells us that we can *add assumptions* to the context without losing any true typing statements.

Lemma: If $\Gamma \vdash t : T$ and $x \notin dom(\Gamma)$, then $\Gamma, x{:}S \vdash t : T$.

Moreover, the latter derivation has the same depth as the former.

Permutation tells us that the order of assumptions in (the list) Γ does not matter.

Lemma: If $\Gamma \vdash t : T$ and Δ is a permutation of Γ, then $\Delta \vdash t : T$.

Moreover, the latter derivation has the same depth as the former.

Preservation

Theorem: If $\Gamma \vdash t : T$ and $t \longrightarrow t'$, then $\Gamma \vdash t' : T$.

Proof: By induction on typing derivations.

Which case is the hard one??

Preservation

Theorem: If $\Gamma \vdash t : T$ and $t \longrightarrow t'$, then $\Gamma \vdash t' : T$.

Proof: By induction on typing derivations.

Case T-App: Given

$$t = t_1\ t_2$$
$$\Gamma \vdash t_1 : T_{11} \rightarrow T_{12}$$
$$\Gamma \vdash t_2 : T_{11}$$
$$T = T_{12}$$

Show $\Gamma \vdash t' : T_{12}$

Preservation

Theorem: If $\Gamma \vdash t : T$ and $t \longrightarrow t'$, then $\Gamma \vdash t' : T$.

Proof: By induction on typing derivations.

Case T-App: Given $t = t_1\ t_2$

$\Gamma \vdash t_1 : T_{11} \to T_{12}$

$\Gamma \vdash t_2 : T_{11}$

$T = T_{12}$

Show $\Gamma \vdash t' : T_{12}$

By the inversion lemma for evaluation, there are three subcases...

Preservation

Theorem: If $\Gamma \vdash t : T$ and $t \longrightarrow t'$, then $\Gamma \vdash t' : T$.

Proof: By induction on typing derivations.

Case T-App: Given $t = t_1 \ t_2$

$\Gamma \vdash t_1 : T_{11} \to T_{12}$

$\Gamma \vdash t_2 : T_{11}$

$T = T_{12}$

Show $\Gamma \vdash t' : T_{12}$

By the inversion lemma for evaluation, there are three subcases...

Subcase: $t_1 = \lambda x{:}T_{11}.\ t_{12}$

t_2 a value v_2

$t' = [x \to v_2] t_{12}$

Preservation

Theorem: If $\Gamma \vdash t : T$ and $t \longrightarrow t'$, then $\Gamma \vdash t' : T$.

Proof: By induction on typing derivations.

Case T-App: Given
$t = t_1\ t_2$
$\Gamma \vdash t_1 : T_{11} \to T_{12}$
$\Gamma \vdash t_2 : T_{11}$
$T = T_{12}$
Show $\Gamma \vdash t' : T_{12}$

By the inversion lemma for evaluation, there are three subcases...

Subcase: $t_1 = \lambda x{:}T_{11}.\ t_{12}$
t_2 a value v_2
$t' = [x \mapsto v_2]t_{12}$

What do we need to know to make this case go through??

The “Substitution Lemma”

Lemma: If $\Gamma, x{:}S \vdash t : T$ and $\Gamma \vdash s : S$, then $\Gamma \vdash [x \to s]t : T$.

I.e., “Types are preserved under substitition.”

The "Substitution Lemma"

Lemma: If $\Gamma, x{:}S \vdash t : T$ and $\Gamma \vdash s : S$, then $\Gamma \vdash [x \to s]t : T$.

Proof: By induction on the *depth* of a derivation of $\Gamma, x{:}S \vdash t : T$. Proceed by cases on the final typing rule used in the derivation.

The "Substitution Lemma"

Lemma: If $\Gamma, x{:}S \vdash t : T$ and $\Gamma \vdash s : S$, then $\Gamma \vdash [x \mapsto s]t : T$.

Proof: By induction on the *depth* of a derivation of $\Gamma, x{:}S \vdash t : T$. Proceed by cases on the final typing rule used in the derivation.

Case T-App:

$$t = t_1 \; t_2$$
$$\Gamma, x{:}S \vdash t_1 : T_2 \to T_1$$
$$\Gamma, x{:}S \vdash t_2 : T_2$$
$$T = T_1$$

By the induction hypothesis, $\Gamma \vdash [x \mapsto s]t_1 : T_2 \to T_1$ and $\Gamma \vdash [x \mapsto s]t_2 : T_2$. By T-App, $\Gamma \vdash [x \mapsto s]t_1 \; [x \mapsto s]t_2 : T$, i.e., $\Gamma \vdash [x \mapsto s](t_1 \; t_2) : T$.

The "Substitution Lemma"

Lemma: If $\Gamma, x{:}S \vdash t : T$ and $\Gamma \vdash s : S$, then $\Gamma \vdash [x \mapsto s]t : T$.

Proof: By induction on the *depth* of a derivation of $\Gamma, x{:}S \vdash t : T$. Proceed by cases on the final typing rule used in the derivation.

Case T-Var: $t = z$
with $z{:}T \in (\Gamma, x{:}S)$

There are two sub-cases to consider, depending on whether z is x or another variable. If $z = x$, then $[x \mapsto s]z = s$. The required result is then $\Gamma \vdash s : S$, which is among the assumptions of the lemma. Otherwise, $[x \mapsto s]z = z$, and the desired result is immediate.

The "Substitution Lemma"

Lemma: If $\Gamma, x{:}S \vdash t : T$ and $\Gamma \vdash s : S$, then $\Gamma \vdash [x \to s]t : T$.

Proof: By induction on the *depth* of a derivation of $\Gamma, x{:}S \vdash t : T$. Proceed by cases on the final typing rule used in the derivation.

Case T-Abs: $t = \lambda y{:}T_2.\, t_1 \qquad T = T_2 \to T_1$

$\Gamma, x{:}S, y{:}T_2 \vdash t_1 : T_1$

By our conventions on choice of bound variable names, we may assume $x \neq y$ and $y \notin FV(s)$. Using *permutation* on the given subderivation, we obtain $\Gamma, y{:}T_2, x{:}S \vdash t_1 : T_1$.

Using *weakening* on the other given derivation ($\Gamma \vdash s : S$), we obtain $\Gamma, y{:}T_2 \vdash s : S$. Now, by the induction hypothesis, $\Gamma, y{:}T_2 \vdash [x \to s]t_1 : T_1$.

By T-Abs, $\Gamma \vdash \lambda y{:}T_2.\ [x \to s]t_1 : T_2 \to T_1$, i.e. (by the definition of substitution), $\Gamma \vdash [x \to s]\lambda y{:}T_2.\ t_1 : T_2 \to T_1$.

Preservation

Recommended: Complete the proof of preservation