CIS 500
Software Foundations
Fall 2006

October 4

Any Questions?

More on Types

Review: Typing Rules

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

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

$$\frac{\texttt{t}_1 : \texttt{Bool} \qquad \texttt{t}_2 : \texttt{T} \qquad \texttt{t}_3 : \texttt{T}}{\texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 : \texttt{T}} \quad \text{(T-If)}$$

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

$$\frac{\texttt{t}_1 : \texttt{Nat}}{\texttt{succ } \texttt{t}_1 : \texttt{Nat}} \quad \text{(T-Succ)}$$

$$\frac{\texttt{t}_1 : \texttt{Nat}}{\texttt{pred } \texttt{t}_1 : \texttt{Nat}} \quad \text{(T-Pred)}$$

$$\frac{\texttt{t}_1 : \texttt{Nat}}{\texttt{iszero } \texttt{t}_1 : \texttt{Bool}} \quad \text{(T-IsZero)}$$

Review: Inversion

Lemma:

1. If $\texttt{true} : \texttt{R}$, then $\texttt{R} = \texttt{Bool}$.
2. If $\texttt{false} : \texttt{R}$, then $\texttt{R} = \texttt{Bool}$.
3. If $\texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 : \texttt{R}$, then $\texttt{t}_1 : \texttt{Bool}$, $\texttt{t}_2 : \texttt{R}$, and $\texttt{t}_3 : \texttt{R}$.
4. If $\texttt{0} : \texttt{R}$, then $\texttt{R} = \texttt{Nat}$.
5. If $\texttt{succ } \texttt{t}_1 : \texttt{R}$, then $\texttt{R} = \texttt{Nat}$ and $\texttt{t}_1 : \texttt{Nat}$.
6. If $\texttt{pred } \texttt{t}_1 : \texttt{R}$, then $\texttt{R} = \texttt{Nat}$ and $\texttt{t}_1 : \texttt{Nat}$.
7. If $\texttt{iszero } \texttt{t}_1 : \texttt{R}$, then $\texttt{R} = \texttt{Bool}$ and $\texttt{t}_1 : \texttt{Nat}$.

Canonical Forms

Lemma:

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

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.

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.

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 type T). Then either t is a value or else there is some t' with $t \longrightarrow t'$.

Progress

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

Proof:

Progress

Theorem: Suppose t is a well-typed term (that is, $t : T$ for some type 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 type 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 t is a well-typed term (that is, $t : T$ for some type 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.

Case T-If: $t = \texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3$
$t_1 : \texttt{Bool} \quad t_2 : T \quad t_3 : T$

Progress

Theorem: Suppose t is a well-typed term (that is, $t : T$ for some type 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.

Case T-If: $t = \texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3$
$t_1 : \texttt{Bool} \quad t_2 : T \quad t_3 : T$

By the induction hypothesis, either t_1 is a value or else there is some t_1' such that $t_1 \longrightarrow t_1'$. If t_1 is a value, then the canonical forms lemma tells us that it must be either $\texttt{true}$ or $\texttt{false}$, in which case either E-IfTrue or E-IfFalse applies to t. On the other hand, if $t_1 \longrightarrow t_1'$, then, by E-If, $t \longrightarrow \texttt{if } t_1' \texttt{ then } t_2 \texttt{ else } t_3$.

Progress

Theorem: Suppose t is a well-typed term (that is, $t : T$ for some type 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 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 $t' : T$.

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.

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

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

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.

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: $\quad \mathtt{t}_1 = \mathtt{true} \qquad \mathtt{t}' = \mathtt{t}_2$

Immediate, by the assumption $\mathtt{t}_2 : \mathtt{T}$.

(E-IfFalse subcase: Similar.)

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-If: $\quad \mathtt{t}_1 \longrightarrow \mathtt{t}_1' \qquad \mathtt{t}' = \mathtt{if}\ \mathtt{t}_1'\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3$

Applying the IH to the subderivation of $\mathtt{t}_1 : \mathtt{Bool}$ yields $\mathtt{t}_1' : \mathtt{Bool}$. Combining this with the assumptions that $\mathtt{t}_2 : \mathtt{T}$ and $\mathtt{t}_3 : \mathtt{T}$, we can apply rule T-If to conclude that $\mathtt{if}\ \mathtt{t}_1'\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 : \mathtt{T}$, that is, $\mathtt{t}' : \mathtt{T}$.

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→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 typin rules simpler. Let's take this choice for now.

Typing rules

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

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

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

Typing rules

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

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

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

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

Typing rules

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

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

$$\frac{\texttt{t}_1 : \texttt{Bool} \quad \texttt{t}_2 : \texttt{T} \quad \texttt{t}_3 : \texttt{T}}{\texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 : \texttt{T}} \qquad \text{(T-If)}$$

$$\frac{\Gamma, \texttt{x}{:}\texttt{T}_1 \vdash \texttt{t}_2 : \texttt{T}_2}{\Gamma \vdash \lambda\texttt{x}{:}\texttt{T}_1\texttt{.t}_2 : \texttt{T}_1 \to \texttt{T}_2} \qquad \text{(T-Abs)}$$

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

Typing rules

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

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

$$\frac{\Gamma \vdash \texttt{t}_1 : \texttt{Bool} \quad \Gamma \vdash \texttt{t}_2 : \texttt{T} \quad \Gamma \vdash \texttt{t}_3 : \texttt{T}}{\Gamma \vdash \texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 : \texttt{T}} \qquad \text{(T-If)}$$

$$\frac{\Gamma, \texttt{x}{:}\texttt{T}_1 \vdash \texttt{t}_2 : \texttt{T}_2}{\Gamma \vdash \lambda\texttt{x}{:}\texttt{T}_1\texttt{.t}_2 : \texttt{T}_1 \to \texttt{T}_2} \qquad \text{(T-Abs)}$$

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

$$\frac{\Gamma \vdash \texttt{t}_1 : \texttt{T}_{11} \to \texttt{T}_{12} \quad \Gamma \vdash \texttt{t}_2 : \texttt{T}_{11}}{\Gamma \vdash \texttt{t}_1\ \texttt{t}_2 : \texttt{T}_{12}} \qquad \text{(T-App)}$$

Typing Derivations

What derivations justify the following typing statements?

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

Properties of $\lambda_{\to}$

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...

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 $\Gamma \vdash \texttt{true} : \texttt{R}$, 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

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 $\texttt{x}{:}\texttt{R} \in \Gamma$.

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 $\Gamma \vdash \texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 : \texttt{R}$, 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 $\texttt{x}{:}\texttt{R} \in \Gamma$.
5. If $\Gamma \vdash \lambda\texttt{x}{:}\texttt{T}_1.\texttt{t}_2 : \texttt{R}$, 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 $\texttt{x}{:}\texttt{R} \in \Gamma$.
5. If $\Gamma \vdash \lambda\texttt{x}{:}\texttt{T}_1.\texttt{t}_2 : \texttt{R}$, then $\texttt{R} = \texttt{T}_1 \to \texttt{R}_2$ for some $\texttt{R}_2$ with $\Gamma, \texttt{x}{:}\texttt{T}_1 \vdash \texttt{t}_2 : \texttt{R}_2$.

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 $\texttt{x}{:}\texttt{R} \in \Gamma$.
5. If $\Gamma \vdash \lambda\texttt{x}{:}\texttt{T}_1.\texttt{t}_2 : \texttt{R}$, then $\texttt{R} = \texttt{T}_1 \to \texttt{R}_2$ for some $\texttt{R}_2$ with $\Gamma, \texttt{x}{:}\texttt{T}_1 \vdash \texttt{t}_2 : \texttt{R}_2$.
6. If $\Gamma \vdash \texttt{t}_1\ \texttt{t}_2 : \texttt{R}$, 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 $\texttt{x}{:}\texttt{R} \in \Gamma$.
5. If $\Gamma \vdash \lambda\texttt{x}{:}\texttt{T}_1\texttt{.t}_2 : \texttt{R}$, then $\texttt{R} = \texttt{T}_1 \rightarrow \texttt{R}_2$ for some $\texttt{R}_2$ with $\Gamma, \texttt{x}{:}\texttt{T}_1 \vdash \texttt{t}_2 : \texttt{R}_2$.
6. If $\Gamma \vdash \texttt{t}_1\ \texttt{t}_2 : \texttt{R}$, then there is some type $\texttt{T}_{11}$ such that $\Gamma \vdash \texttt{t}_1 : \texttt{T}_{11} \rightarrow \texttt{R}$ and $\Gamma \vdash \texttt{t}_2 : \texttt{T}_{11}$.

Canonical Forms

Lemma:

Canonical Forms

Lemma:

1. If $\texttt{v}$ is a value of type $\texttt{Bool}$, then

Canonical Forms

Lemma:

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

Canonical Forms

Lemma:

1. If $\texttt{v}$ is a value of type $\texttt{Bool}$, then $\texttt{v}$ is either $\texttt{true}$ or $\texttt{false}$.
2. If $\texttt{v}$ is a value of type $\texttt{T}_1 \rightarrow \texttt{T}_2$, then

Canonical Forms

Lemma:

1. If $\texttt{v}$ is a value of type $\texttt{Bool}$, then $\texttt{v}$ is either $\texttt{true}$ or $\texttt{false}$.
2. If $\texttt{v}$ is a value of type $\texttt{T}_1 \rightarrow \texttt{T}_2$, then $\texttt{v}$ has the form $\lambda\texttt{x}{:}\texttt{T}_1\texttt{.t}_2$.

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

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

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 $\Gamma \vdash t' : T$.

Proof: By induction

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 $t = t_1\ t_2$

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

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

$T = T_{12}$

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 $t = t_1\ t_2$

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

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

$T = T_{12}$

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

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

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

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

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

Uh oh.

The "Substitution Lemma"

Lemma: Types are preserved under substitition.

That is, if $\Gamma, x{:}S \vdash t : T$ and $\Gamma \vdash s : S$, then $\Gamma \vdash [x \mapsto s]t : T$.

The "Substitution Lemma"

Lemma: Types are preserved under substitition.

That is, if $\Gamma, x{:}S \vdash t : T$ and $\Gamma \vdash s : S$, then $\Gamma \vdash [x \mapsto s]t : T$.

Proof: ...

Preservation

Recommended: Complete the proof of preservation