Type Systems Winter Semester 2006

Week 6 November 22

November 22, 2006 - version 1.0

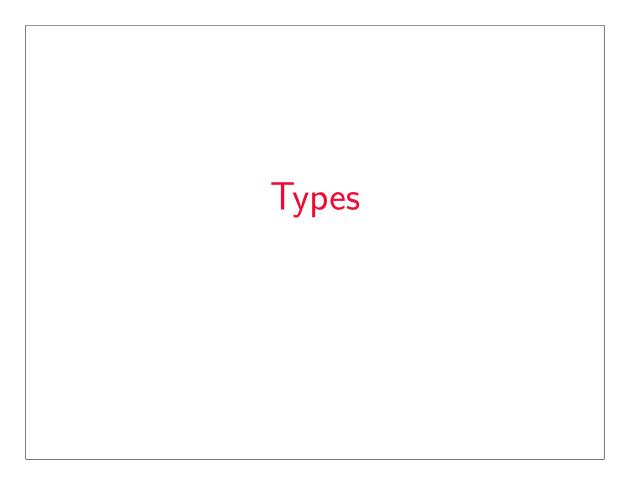
Plan

PREVIOUSLY: untyped lambda calculus

TODAY: types!!

- 1. Two example languages:
 - 1.1 typing arithmetic expressions
 - 1.2 simply typed lambda calculus (STLC)
- 2. For each:
 - 2.1 Define types
 - 2.2 Specify typing rules
 - 2.3 Prove soundness: progress and preservation

NEXT: lambda calculus extensions NEXT: polymorphic typing

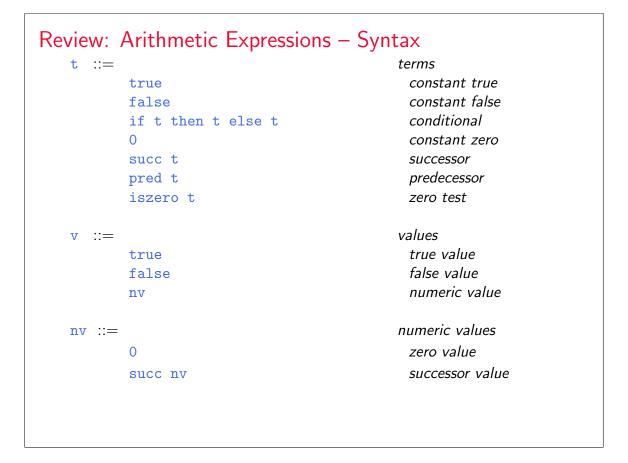


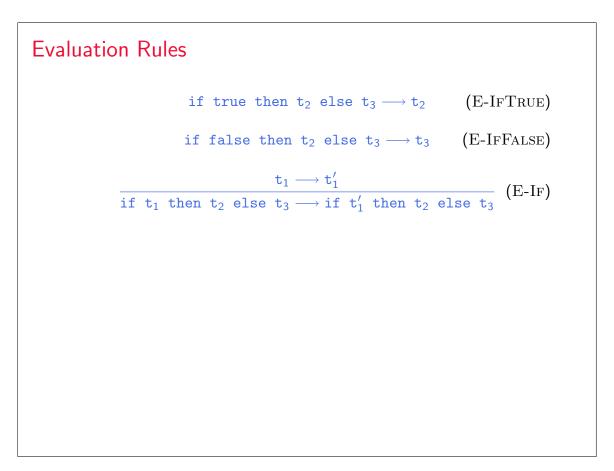
Outline

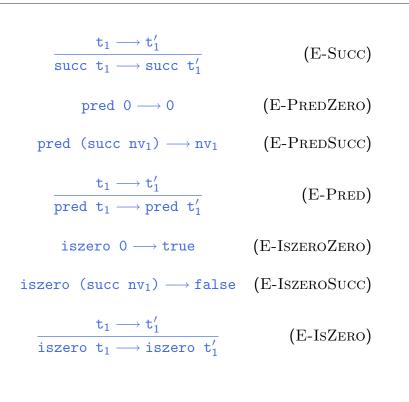
- 1. begin with a set of terms, a set of values, and an evaluation relation
- 2. define a set of *types* classifying values according to their "shapes"
- 3. define a *typing relation* t : T that classifies terms according to the shape of the values that result from evaluating them
- 4. check that the typing relation is *sound* in the sense that,

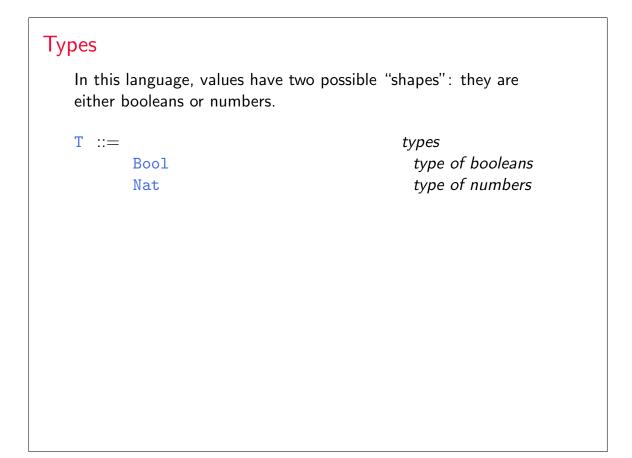
4.1 if t : T and t \longrightarrow^* v, then v : T

4.2 if t : T, then evaluation of t will not get stuck



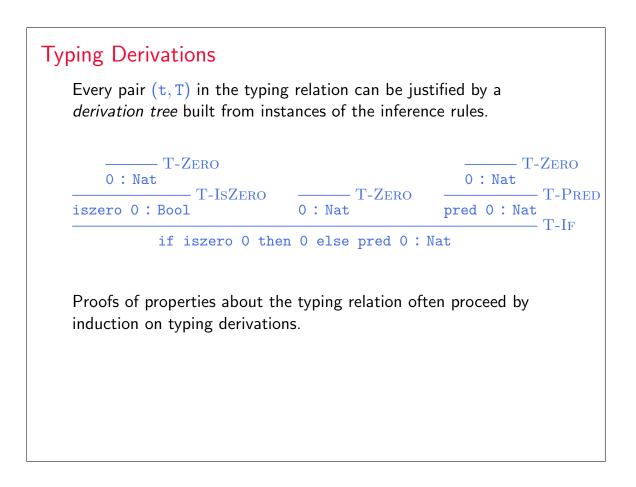






Typing Rules

true : Bool false : Bool	(T-True) (T-False)
$\frac{t_1:Bool}{if t_1 then t_2 else t_3:T}$	(T-IF)
0 : Nat	(T-Zero)
$\frac{\texttt{t}_1:\texttt{Nat}}{\texttt{succ }\texttt{t}_1:\texttt{Nat}}$	(T-Succ)
$\frac{\texttt{t}_1:\texttt{Nat}}{\texttt{pred }\texttt{t}_1:\texttt{Nat}}$	(T-Pred)
$\frac{\texttt{t}_1:\texttt{Nat}}{\texttt{iszero }\texttt{t}_1:\texttt{Bool}}$	(T-IsZero)



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.

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

Using this rule, we cannot assign a type to

if true then 0 else false

even though this term will certainly evaluate to a number.

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.

Lemma:

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

Inversion

Lemma:

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

Proof: ...

Lemma:

- 1. If true : R, then R = Bool.
- 2. If false : R, then R = Bool.
- 3. If if t_1 then t_2 else t_3 : R, then t_1 : Bool, t_2 : R, and t_3 : R.
- 4. If 0 : R, then R = Nat.
- 5. If succ t_1 : R, then R = Nat and t_1 : Nat.
- 6. If pred t_1 : R, then R = Nat and t_1 : Nat.
- 7. If iszero t_1 : R, then R = Bool and t_1 : 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 = 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"
```

Properties of the Typing Relation

Review: Typing Rules

(T-TRUE)	true : Bool
(T-FALSE)	false : Bool
(T-IF)	$\frac{\mathtt{t}_1:\texttt{Bool}}{\texttt{if }\mathtt{t}_1\texttt{ then }\mathtt{t}_2\texttt{ else }\mathtt{t}_3\texttt{:}\mathtt{T}}$
(T-Zero)	0: Nat
(T-Succ)	$\frac{\texttt{t}_1:\texttt{Nat}}{\texttt{succ }\texttt{t}_1:\texttt{Nat}}$
(T-Pred)	$\frac{\mathtt{t}_1:\mathtt{Nat}}{\mathtt{pred}\ \mathtt{t}_1:\mathtt{Nat}}$
(T-IsZero)	t_1 : Nat iszero t_1 : Bool

Review: Inversion

Lemma:

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

Canonical Forms

Lemma:

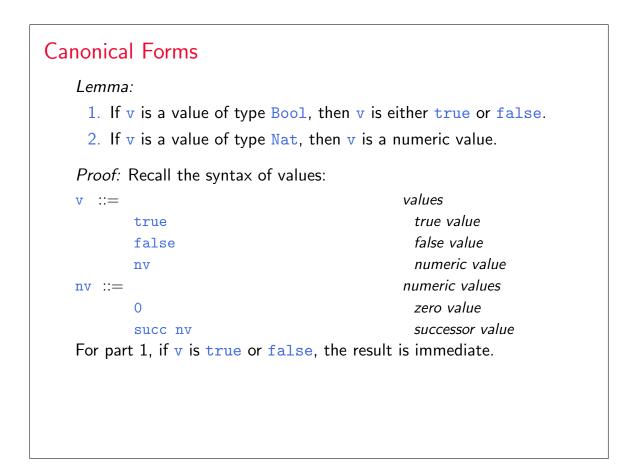
- 1. If v is a value of type Bool, then v is either true or false.
- 2. If v is a value of type Nat, then v is a numeric value.

Proof:

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 par	rt 1,	



Canonical Forms

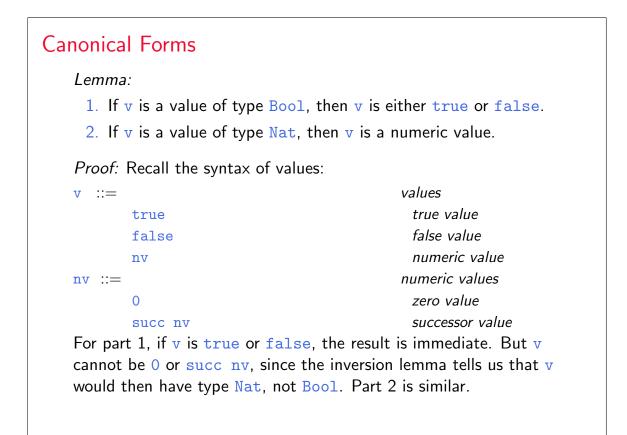
Lemma:

- 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. But v cannot be 0 or succ nv, since the inversion lemma tells us that v would then have type Nat, not Bool.



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:

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 $T\text{-}T\text{-}T\text{-}\text{RUE},\ T\text{-}\text{FALSE},$ 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 $T\text{-}T\text{-}T\text{-}\text{RUE},\ T\text{-}\text{FALSE},$ 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.

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 true or 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, $t \longrightarrow \text{if } t'_1$ then t_2 else t_3 .

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 $T\mathchar`-ZERO,$ $T\mathchar`-Succ,$ $T\mathchar`-PRED,$ and $T\mathchar`-IsZERO$ are similar.

(Recommended: Try to reconstruct them.)

Preservation

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

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: t = true T = BoolThen t is a value.

Preservation

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

Proof: By induction on the given typing derivation.

Case T-IF:

```
t = if t_1 then t_2 else t_3 t_1 : Bool t_2 : T 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 t : T and t \longrightarrow t', then t' : T.

Proof: By induction on the given typing derivation.

Case T-IF:

```
t = if t_1 then t_2 else t_3 t_1 : Bool t_2 : T 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-IFTRUE: $t_1 = true$ $t' = t_2$ Immediate, by the assumption t_2 : 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:

 $t = if t_1 then t_2 else t_3 t_1 : Bool t_2 : T t_3 : T$

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: $t_1 \longrightarrow t'_1$ $t' = \text{if } t'_1$ then t_2 else t_3 Applying the IH to the subderivation of t_1 : Bool yields t'_1 : Bool. Combining this with the assumptions that t_2 : T and t_3 : T, we can apply rule T-IF to conclude that if t'_1 then t_2 else t_3 : T, that is, t': T.

Messing With It

Messing with it: Remove a rule

What if you remove E-PREDZERO ?

Messing with it: Remove a rule

What if you remove $\operatorname{E-PREDZERO}$?

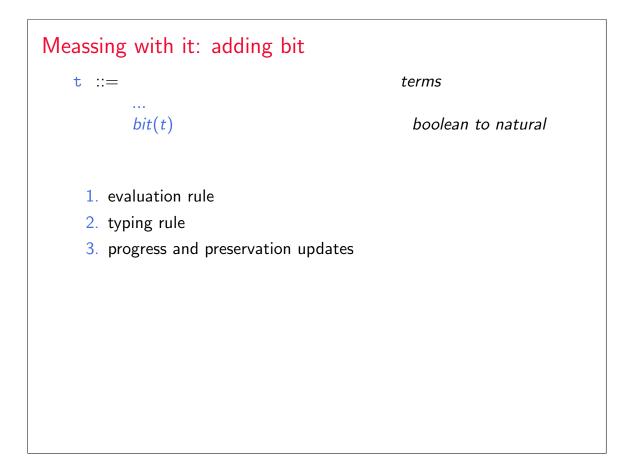
Then $pred \ 0$ type checks is stuck, and it is not $pred \ 0$ a value. Thus the progress theorem fails.

Messing with it: If

What if you changed the rule for typing if's to the following:

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

Messing with it: If What if you changed the rule for typing if's to the following: $\frac{t_1 : Bool \quad t_2 : Nat \quad t_3 : Nat}{if \ t_1 \ then \ t_2 \ else \ t_3 : Nat} \qquad (T-IF)$ The system is still sound. Some if's do not type, but those that do are fine.



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 λ_{\rightarrow} for short.

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

- ► So, strictly speaking, there are many variants of λ→, 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
$\lambda \texttt{x.t}$	abstraction
t t	application
true	constant true
false	constant false
if t then t else t	conditional
v ::=	values
$\lambda x.t$	abstraction value
true	true value
false	false value

"Simple Types"

 $\begin{array}{c} T & ::= \\ & \text{Bool} \\ & T {\rightarrow} T \end{array}$

types type of booleans types of functions

What are some examples?

Type Annotations

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

annotate lambda-abstractions with the expected type of the argument

$\lambda \mathtt{x}:\mathtt{T}_1.$ t₂

(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

(T-TRUE)	true : Bool
(T-False)	false : Bool
(T-IF)	$t_1: Bool t_2: T t_3: T$
()	if t_1 then t_2 else t_3 : T

Typing rulestrue : Bool(T-TRUE)false : Bool(T-FALSE) $\frac{t_1 : Bool}{t_2 : T}$ $t_3 : T$ (T-IF)if t_1 then t_2 else $t_3 : T$ (T-IF) $\frac{???}{\lambda x : T_1 \cdot t_2 : T_1 \rightarrow T_2}$ (T-ABS)

Typing rules

true : Bool	(T-TRUE)
false : Bool	(T-FALSE)
$\frac{\texttt{t}_1:\texttt{Bool}}{\texttt{if t}_1\texttt{ then }\texttt{t}_2\texttt{ : }\texttt{T}\texttt{ t}_3\texttt{ : }\texttt{T}}$	(T-IF)
$\frac{\Gamma, \mathbf{x}: \mathtt{T}_1 \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma \vdash \lambda \mathtt{x}: \mathtt{T}_1 \cdot \mathtt{t}_2 : \mathtt{T}_1 \rightarrow \mathtt{T}_2}$	(T-Abs)
$\frac{\mathtt{x}\!:\!\mathtt{T}\inF}{F\vdash\!\mathtt{x}:\mathtt{T}}$	(T-VAR)

Typing rules	
Γ⊢true : Bool	(T-TRUE)
Γ⊢false : Bool	(T-FALSE)
$\frac{\Gamma \vdash t_1 : \texttt{Bool} \Gamma \vdash t_2 : \texttt{T} \Gamma \vdash t_3 : \texttt{T}}{\Gamma \vdash \texttt{if} \ t_1 \ \texttt{then} \ t_2 \ \texttt{else} \ t_3 : \texttt{T}}$	(T-IF)
$\frac{\Gamma, \mathtt{x}: \mathtt{T}_1 \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma \vdash \lambda \mathtt{x}: \mathtt{T}_1 \cdot \mathtt{t}_2 : \mathtt{T}_1 \rightarrow \mathtt{T}_2}$	(T-Abs)
$\frac{\mathbf{x}:T\in\Gamma}{\Gamma\vdashx:T}$	(T-VAR)
$\frac{\Gamma \vdash t_1 : T_{11} \rightarrow T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1 \ t_2 : T_{12}}$	(T-App)

Typing Derivations

What derivations justify the following typing statements?

- ► \vdash (λ x:Bool.x) true : Bool
- ▶ f:Bool→Bool ⊢ f (if false then true else false) :
 Bool

Properties of λ_{\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 $If \Gamma \vdash t : T \text{ and } t \longrightarrow t', \text{ then } \Gamma \vdash t' : T.$

Proving progress

Same steps as before...

Proving progress

Same steps as before...

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

Lemma:

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

Inversion

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

Lemma:

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

Inversion

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

Lemma:

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

Inversion

- 1. If $\Gamma \vdash \text{true} : R$, then R = Bool.
- 2. If $\Gamma \vdash false : R$, then R = Bool.
- 3. If $\Gamma \vdash if t_1$ then t_2 else $t_3 : R$, then $\Gamma \vdash t_1 :$ 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 .
- 6. If $\Gamma \vdash t_1 t_2 : R$, then

Lemma:

- 1. If $\Gamma \vdash \text{true} : R$, then R = Bool.
- 2. If $\Gamma \vdash false : R$, then R = Bool.
- 3. If $\Gamma \vdash if t_1$ then t_2 else $t_3 : R$, then $\Gamma \vdash t_1 :$ 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 .
- 6. If $\Gamma \vdash t_1 \ t_2 : R$, then there is some type T_{11} such that $\Gamma \vdash t_1 : T_{11} \rightarrow R$ and $\Gamma \vdash t_2 : T_{11}$.

Canonical Forms

Canonical Forms

Lemma:

1. If ${\tt v}$ is a value of type <code>Bool</code>, then

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 $T_1 {\rightarrow} T_2$, then

Canonical Forms

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

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.

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 $\vdash t_1 : T_{11} \rightarrow T_{12}$ and $\vdash t_2 : T_{11}$.

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 $\vdash t_1 : T_{11} \rightarrow T_{12}$ 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 . 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 $\lambda x: T_{11} \cdot t_{12}$, and so rule E-APPABS applies to t.