Review: Typing Rules

true : Bool  (T-True)
false : Bool (T-False)
\[ \frac{t_1 : \text{Bool}}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T} \]  (T-If)
\[ \frac{0 : \text{Nat}}{t_1 : \text{Nat}} \]  (T-Zero)
\[ \frac{\text{succ } t_1 : \text{Nat}}{t_1 : \text{Nat}} \]  (T-Succ)
\[ \frac{\text{pred } t_1 : \text{Nat}}{t_1 : \text{Nat}} \]  (T-Pred)
\[ \frac{\text{iszero } t_1 : \text{Bool}}{t_1 : \text{Nat}} \]  (T-IsZero)

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.

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:
**Canonical Forms**

**Lemma:**
1. If \( v \) is a value of type \( \text{Bool} \), then \( v \) is either \( \text{true} \) or \( \text{false} \).
2. If \( v \) is a value of type \( \text{Nat} \), then \( v \) is a numeric value.

**Proof:** Recall the syntax of values:

\[
\begin{align*}
\text{values} & ::= \text{true} \\ 
& \quad \text{false} \\ 
& \quad \text{nv} \\ 
\text{nv} & ::= 0 \\ 
& \quad \text{succ \ nd} \\
\end{align*}
\]

For part 1, if \( v \) is \( \text{true} \) or \( \text{false} \), the result is immediate. But \( v \) cannot be \( 0 \) or \( \text{succ \ nd} \), since the inversion lemma tells us that \( v \) would then have type \( \text{Nat} \), not \( \text{Bool} \).

**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 \( t \rightarrow t' \).

**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 \( t \rightarrow t' \).

*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.)

**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 \( t \rightarrow t' \).

*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 \( t \rightarrow t' \).

*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: \[
\begin{align*}
t &= \text{if } t_1 \text{ then } t_2 \text{ else } t_3 \\
t_1 &: \text{Bool} \quad t_2 &: T \quad t_3 &: T
\end{align*}
\]

By the induction hypothesis, either \( t_1 \) is a value or else there is some \( t'_1 \) such that \( t_1 \rightarrow t'_1 \). If \( t_1 \) is a value, then the canonical forms lemma tells us that it must be either \textit{true} or \textit{false}, in which case either E-IfTrue or E-IfFalse applies to \( t \). On the other hand, if \( t_1 \rightarrow t'_1 \), then, by E-If, \( t \rightarrow \text{if } t'_1 \text{ then } t_2 \text{ else } t_3 \).

**Preservation**

*Theorem:* If \( t : T \) and \( t \rightarrow t' \), then \( t' : T \).
Preservation

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

**Proof:** By induction on the given typing derivation.

Case $T$-True:

- $t = \text{true}$
- $T = \text{Bool}$

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

Case $T$-If:

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

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

Subcase $E$-IfTrue:

- $t_1 = \text{true}$
- $t' = t_2$

Immediate, by the assumption $t_2 : T$.

(E-IfFalse subcase: Similar.)

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 $\lambda \to$, for short.

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

- So, strictly speaking, there are many variants of $\lambda \to$, depending on the choice of base types.
- For now, we’ll work with a variant constructed over the booleans.

“Simple Types”

$T ::= $

- $\text{Bool}$ type of booleans
- $T \to T$ 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

$$
\begin{array}{l}
\text{true : Bool} & \quad \text{(T-True)} \\
\text{false : Bool} & \quad \text{(T-False)} \\
\hline
\frac{t_1 : \text{Bool} \quad t_2 : T \quad t_3 : T}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T} & \quad \text{(T-If)}
\end{array}
$$

$$
\begin{array}{l}
\text{true : Bool} & \quad \text{(T-True)} \\
\text{false : Bool} & \quad \text{(T-False)} \\
\hline
\frac{t_1 : \text{Bool} \quad t_2 : T \quad t_3 : T}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T} & \quad \text{(T-If)}
\end{array}
$$

$$
\begin{array}{l}
\text{true : Bool} & \quad \text{(T-True)} \\
\text{false : Bool} & \quad \text{(T-False)} \\
\hline
\frac{t_1 : \text{Bool} \quad t_2 : T \quad t_3 : T}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T} & \quad \text{(T-If)}
\end{array}
$$

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

Untyped lambda-calculus with booleans

$t ::= $

- terms
  - $x$ variable
  - $\lambda x. t$ abstraction
  - $t \ t$ application
  - $\text{true}$ constant true
  - $\text{false}$ constant false
  - $\text{if } t \text{ then } t \text{ else } t$ conditional

$v ::= $

- values
  - $\lambda x. t$ abstraction value
  - $\text{true}$ true value
  - $\text{false}$ false value
Typing rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true : Bool</td>
<td>(T-TRUE)</td>
</tr>
<tr>
<td>false : Bool</td>
<td>(T-FALSE)</td>
</tr>
<tr>
<td>if ( t_1 ) then ( t_2 ) else ( t_3 ) : T</td>
<td>(T-IF)</td>
</tr>
<tr>
<td>( \Gamma, x : T_1 \vdash t_2 : T_2 )</td>
<td>(T-ABS)</td>
</tr>
<tr>
<td>( x : T \in \Gamma )</td>
<td>(T-VAR)</td>
</tr>
</tbody>
</table>

Typing Derivations

What derivations justify the following typing statements?

\( \vdash (\lambda x : \text{Bool}. x) \) true : Bool
\( f : \text{Bool} \rightarrow \text{Bool} \vdash f \) (if false then true else false) : Bool
\( f : \text{Bool} \rightarrow \text{Bool} \vdash \lambda x : \text{Bool}. f \) (if x then false else x) : Bool \rightarrow \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
   \( \text{If } \Gamma \vdash t : T, \text{ then either } t \text{ is a value or else } t \rightarrow t' \) for some \( t' \).

2. Preservation: Types are preserved by one-step evaluation
   \( \text{If } \Gamma \vdash t : T \text{ and } t \rightarrow t', \text{ then } \Gamma \vdash t' : T. \)

Proving progress

Same steps as before...

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

Lemma:
1. If $\Gamma \vdash \text{true} : R$, then $R = \text{Bool}$.
2. If $\Gamma \vdash \text{false} : R$, then $R = \text{Bool}$.
3. If $\Gamma \vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : R$, then $\Gamma \vdash t_1 : \text{Bool}$ 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 $\Gamma \vdash t_2 : T_{11}$.

Inversion

Lemma:
1. If $\Gamma \vdash \text{true} : R$, then $R = \text{Bool}$.
2. If $\Gamma \vdash \text{false} : R$, then $R = \text{Bool}$.
3. If $\Gamma \vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : R$, then $\Gamma \vdash t_1 : \text{Bool}$ 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 $\Gamma \vdash t_2 : T_{11}$. 
Inversion

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

**Lemma:**
1. If $v$ is a value of type $\text{Bool}$, then $v$ is either $\text{true}$ or $\text{false}$.
2. If $v$ is a value of type $T_1 \rightarrow T_2$, then $v$ has the form $\lambda x : T_1.t_2$. 
The cases for boolean is closed. The abstraction case is immediate, on typing derivations.

\[ \text{Proof: By induction on typing derivations.} \]

The cases for boolean is closed. The abstraction case is immediate, on typing derivations.

\[ \text{Proof: By induction on typing derivations.} \]

The cases for boolean is closed. The abstraction case is immediate, on typing derivations.

\[ \text{Proof: By induction on typing derivations.} \]

The cases for boolean is closed. The abstraction case is immediate, on typing derivations.

\[ \text{Proof: By induction on typing derivations.} \]

The cases for boolean is closed. The abstraction case is immediate, on typing derivations.

\[ \text{Proof: By induction on typing derivations.} \]

The cases for boolean is closed. The abstraction case is immediate, on typing derivations.

\[ \text{Proof: By induction on typing derivations.} \]

The cases for boolean is closed. The abstraction case is immediate, on typing derivations.

\[ \text{Proof: By induction on typing derivations.} \]
**Preservation**

**Theorem:** If $\Gamma \vdash t : T$ and $t \rightarrow^* t'$, then $\Gamma \vdash t' : T$.

**Proof:** By induction on typing derivations.

Case T-App: Given $t = t_1 \; t_2$
- $\Gamma \vdash t_1 : T_1 \rightarrow T_{12}$
- $\Gamma \vdash t_2 : T_{11}$
- $T = T_{12}$

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 substitution.
That is, if □, x:S ⊢ t : T and □ ⊢ s : S, then □ ⊢ [x ↦→ s]t : T.

Proof: ...

Preservation

Recommended: Complete the proof of preservation