Any Questions?
More on Types
Review: Typing Rules

true : Bool
false : Bool

\[
\begin{array}{c}
\frac{t_1 : \text{Bool} \quad t_2 : T \quad t_3 : T}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T}
\end{array}
\]

0 : Nat
\[
\begin{array}{c}
\frac{t_1 : \text{Nat}}{\text{succ } t_1 : \text{Nat}}
\end{array}
\]

\[
\begin{array}{c}
\frac{t_1 : \text{Nat}}{\text{pred } t_1 : \text{Nat}}
\end{array}
\]

\[
\begin{array}{c}
\frac{t_1 : \text{Nat}}{\text{iszero } t_1 : \text{Bool}}
\end{array}
\]
Review: Inversion

Lemma:

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

\[
\begin{align*}
v & ::= \quad \text{values} \\
& \quad \text{true value} \\
& \quad \text{false value} \\
& \quad \text{numeric value} \\
& \quad \text{numeric values} \\
& \quad \text{zero value} \\
& \quad \text{successor value} \\
& \quad \text{true} \\
& \quad \text{false} \\
& \quad \text{nv} \\
& \quad \text{nv} ::= \quad \text{numeric values} \\
& \quad \text{0} \\
& \quad \text{succ nv}
\end{align*}
\]

For part 1,
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:

$v ::= \text{values}$

- $\text{true}$
- $\text{false}$
- $\text{nv}$

$\text{nv} ::= \text{numeric values}$

- $0$
- $\text{succ } \text{nv}$

For part 1, if $v$ is $\text{true}$ or $\text{false}$, the result is immediate.
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:

\[
\begin{align*}
v & ::= \quad \text{values} \\
& \quad \text{true value} \\
& \quad \text{false value} \\
& \quad \text{numeric value} \\
& \\
\text{nv} & ::= \quad \text{numeric values} \\
& \quad \text{zero value} \\
& \quad \text{successor value} \\
\end{align*}
\]

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

\[
v ::= \quad \text{values} \\
\quad \text{true} \quad \quad \quad \quad \quad \quad \text{true value} \\
\quad \text{false} \quad \quad \quad \quad \quad \quad \text{false value} \\
\quad n v \quad \quad \quad \quad \quad \quad \text{numeric value} \\
\]

\[
n v ::= \quad \text{numeric values} \\
\quad 0 \quad \quad \quad \quad \quad \quad \text{zero value} \\
\quad \text{succ} \ n v \quad \quad \quad \quad \quad \text{successor value} \\
\]

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

**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:**
**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 $t : T$. 

---

**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 $t : T$. 

---

---

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

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 true or 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, $t \rightarrow \text{if } t'_1 \text{ then } t_2 \text{ else } t_3$. 
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 \( T\text{-}\text{ZERO}, T\text{-}\text{Succ}, T\text{-}\text{Pred}, \) and \( T\text{-}\text{IsZero} \) are similar.

(Recommended: Try to reconstruct them.)
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.
Preservation

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

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

*Case* \( \mathbf{T-True} \): \( t = \text{true} \quad T = \text{Bool} \)

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

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

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

*Case* \( T\text{-If} \):  
\[
t = \text{if } t_1 \text{ then } t_2 \text{ else } t_3 \quad t_1 : \text{Bool} \quad t_2 : T \quad t_3 : T
\]

There are three evaluation rules by which \( t \rightarrow t' \) can be derived: \( \text{E-IfTrue} \), \( \text{E-IfFalse} \), and \( \text{E-If} \). Consider each case separately.
Preservation

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

**Theorem:** If $t : T$ and $t \rightarrow t'$, 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 $t \rightarrow t'$ can be derived: $\text{E-IfTrue}$, $\text{E-IfFalse}$, and $\text{E-If}$. Consider each case separately.

**Subcase E-If:**

$t_1 \rightarrow t'_1 \quad t' = \text{if } t'_1 \text{ then } t_2 \text{ else } t_3$

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 $\text{T-If}$ to conclude that $\text{if } t'_1 \text{ then } t_2 \text{ else } t_3 : T$, that is, $t' : T$. 
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 $\lambda\to$ 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.
Untyped lambda-calculus with booleans

\[
\begin{align*}
t & ::= \\
& x \\
& \lambda x. t \\
& t \ t \\
& true \\
& false \\
& if \ t \ then \ t \ else \ t
\end{align*}
\]
\[\text{terms}\]
\[\text{variable}\]
\[\text{abstraction}\]
\[\text{application}\]
\[\text{constant true}\]
\[\text{constant false}\]
\[\text{conditional}\]

\[
\begin{align*}
v & ::= \\
& \lambda x. t \\
& true \\
& false
\end{align*}
\]
\[\text{values}\]
\[\text{abstraction value}\]
\[\text{true value}\]
\[\text{false value}\]
“Simple Types”

\[ T ::= \]

\[ \text{Bool} \]

\[ T \rightarrow T \]

\text{types}
\text{type of booleans}
\text{types of functions}
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

true : Bool \quad \text{(T-TRUE)}

false : Bool \quad \text{(T-FALSE)}

\[
\frac{t_1 : \text{Bool} \quad t_2 : T \quad t_3 : T}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T} \quad \text{(T-IF)}
\]
Typing rules

true : Bool \hspace{5cm} (T-TRUE)
false : Bool \hspace{5cm} (T-FALSE)

\[ \begin{align*}
t_1 : \text{Bool} \quad t_2 : T \quad t_3 : T \\
\text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T
\end{align*} \] \hspace{5cm} (T-IF)

\[ \begin{align*}
??? \\
\lambda x : T_1 . t_2 : T_1 \rightarrow T_2
\end{align*} \] \hspace{5cm} (T-ABS)
Typing rules

true : Bool  \quad (T-\text{TRUE})

false : Bool  \quad (T-\text{FALSE})

\[
\frac{\begin{array}{c}
t_1 : \text{Bool} \\
t_2 : T \\
t_3 : T
\end{array}}{	ext{if } t_1 \text{ then } t_2 \text{ else } t_3 : T} \quad (T-\text{IF})
\]

\[
\frac{\Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x : T_1. t_2 : T_1 \rightarrow T_2} \quad (T-\text{ABS})
\]

\[
\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \quad (T-\text{VAR})
\]
Typing rules

\[ \Gamma \vdash \text{true} : \text{Bool} \]  \hspace{1cm} (T-TRUE)

\[ \Gamma \vdash \text{false} : \text{Bool} \]  \hspace{1cm} (T-FALSE)

\[ \begin{align*}
\Gamma \vdash t_1 : \text{Bool} & \quad \Gamma \vdash t_2 : T & \quad \Gamma \vdash t_3 : T \\
\hline
\Gamma \vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T
\end{align*} \]  \hspace{1cm} (T-IF)

\[ \begin{align*}
\Gamma, x : T_1 & \vdash t_2 : T_2 \\
\hline
\Gamma \vdash \lambda x : T_1 . t_2 : T_1 \rightarrow T_2
\end{align*} \]  \hspace{1cm} (T-ABS)

\[ \begin{align*}
x : T & \in \Gamma \\
\hline
\Gamma \vdash x : T
\end{align*} \]  \hspace{1cm} (T-VAR)

\[ \begin{align*}
\Gamma \vdash t_1 : T_{11} \rightarrow T_{12} & \quad \Gamma \vdash t_2 : T_{11} \\
\hline
\Gamma \vdash t_1 \ t_2 : T_{12}
\end{align*} \]  \hspace{1cm} (T-APP)
Typing Derivations

What derivations justify the following typing statements?

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

2. **Preservation:** Types are preserved by one-step evaluation
   
   If $\Gamma \vdash t : T$ and $t \rightarrow t'$, 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 $\Gamma \vdash \text{true} : R$, 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 $\Gamma \vdash t_2, t_3 : R$. 
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
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 $x : R \in \Gamma$. 
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
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 $\Gamma, x : T_1 \vdash t_2 : R_2$. 
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
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:
Lemma:

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

1. If $v$ is a value of type $\text{Bool}$, then $v$ is either $\text{true}$ or $\text{false}$.
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
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 \( \lambda x : T_1 . t_2 \).
**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 $t \rightarrow t'$.

*Proof:* By induction
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 $t \rightarrow t'$.

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 $t \rightarrow t'$.

*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 $t \rightarrow t'$.

**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 $\vdash t_2 : T_{11}$. 
**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 $t \rightarrow t'$.

*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 $t_2$. 
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 $t \rightarrow t'$.

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 $E\text{-App1}$ applies to $t$. If $t_1$ is a value and $t_2$ can take a step, then rule $E\text{-App2}$ 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 $E\text{-AppAbs}$ applies to $t$. 


Preservation

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

**Proof:** By induction
Preservation

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

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

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

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

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

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

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

Proof: By induction on typing derivations.

Case $\text{T-App}$: 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 $t \rightarrow t'$, then $\Gamma \vdash t' : T$.

*Proof:* By induction on typing derivations.

**Case $T\text{-}\text{App}$:** 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 $t \rightarrow t'$, then $\Gamma \vdash t' : T$.

Proof: By induction on typing derivations.

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

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

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

$T = T_{12}$

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 substitution.
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 substitution.
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