Type Soundness

Intuitively, the type system should rule out eventually stuck expressions.
Recall that \( e \) is eventually stuck if \( e \rightarrow^* e' \) and \( e' \) is stuck.

Theorem (Type Soundness) If \( \Gamma \vdash e : T \) then either \( e \)'s evaluation does not terminate or there exists \( v \) such that \( e \rightarrow^* v \) and \( \Gamma \vdash v : T \).
- \( \Gamma \vdash e : T \) is shorthand for \( \{\} \vdash e : T \)

The simply-typed lambda calculus actually satisfies a stronger version of the theorem. Every expression in the simply-typed lambda calculus terminates!

Soundness = Progress + Preservation

Theorem (Type Soundness) If \( \Gamma \vdash e : T \) then either \( e \)'s evaluation does not terminate or there exists \( v \) such that \( e \rightarrow^* v \) and \( \Gamma \vdash v : T \).

A well-typed term \( e \) is not stuck:

Theorem (Progress): If \( \Gamma \vdash e : T \), then either \( e \) is a value or there exists \( e' \) such that \( e \rightarrow e' \).

The \( \rightarrow \) relation preserves typing:

Theorem (Type Preservation): If \( \Gamma \vdash e : T \) and \( e \rightarrow e' \), then \( \Gamma \vdash e' : T \).

Preliminaries

The typing rules make implicit assumptions about the values of various types. For example, T-If assumes that true and false are the only values of type Bool:

\[
\Gamma \vdash e_1 : \text{Bool} \quad \Gamma \vdash e_2 : T \quad \Gamma \vdash e_3 : T \quad (\text{T-If})
\]

The Canonical Forms Lemma validates these assumptions, which are needed by the Progress and Type Preservation proofs.

Lemma (Canonical Forms):
1. If \( \Gamma \vdash v : \text{Bool} \), then \( v \) is either true or false.
2. If \( \Gamma \vdash v : T_1 \rightarrow T_2 \), then \( v \) has the form \( \lambda x : T_1. e \).

Proof “by inspection” of the typing rules.

Preliminaries (cont.)

To prove Type Preservation, we need to reason about the type correctness of the substitution function.

Well-typed substitution produces well-typed terms:

Lemma (Substitution): If \( \Gamma \cup \{x : T\} \vdash e' : T' \) and \( \Gamma \vdash v : T \), then \( \Gamma \vdash [x \mapsto v]e' : T' \).

Proof by strong induction on the depth of the derivation tree for \( \Gamma \cup \{x : T\} \vdash e' : T' \).

Straightforward but somewhat tedious. See the handout for the details.
Theorem (Progress): If ⊢ e : T, then either e is a value or there exists e′ such that e → e′.
- Proof by strong induction on the depth of the derivation tree for ⊢ e : T.
- Case analysis of the last rule used in the derivation.
- Inductive hypothesis allows us to assume the theorem for sub-expressions of e.

Theorem (Type Preservation): If Γ ⊢ e : T and e → e′, then Γ ⊢ e′ : T.
- Proof by strong induction on the depth of the derivation tree for Γ ⊢ e : T.
- Case analysis of the last rule used in the derivation.
- Sub-case analysis of the last rule used in the derivation of e → e′.
- Inductive hypothesis allows us to assume the theorem for sub-expressions of e.

Syntax
\[ e ::= x \mid \lambda x : T. e \mid e_1 e_2 \]
\[ T ::= \text{Bool} \mid T_1 \rightarrow T_2 \]
\[ v ::= \lambda x : T. e \mid \text{true} \mid \text{false} \]

Operational Semantics
\[ (\lambda x : T.e) v \rightarrow [x \mapsto v]e \] (E-AppRed)
\[ e_1 \rightarrow e'_1 \] (E-App1)
\[ e_1 e_2 \rightarrow e'_1 e'_2 \] (E-App2)
\[ \text{if true then } e_2 \text{ else } e_3 \rightarrow e_2 \] (E-IfTrue)
\[ \text{if false then } e_2 \text{ else } e_3 \rightarrow e_3 \] (E-IfFalse)
\[ e_1 \rightarrow e'_1 \] (E-If)
\[ e_1 e_2 \rightarrow e'_1 e'_2 \] (E-If)

Typing Rules
\[ \Gamma, x : T \vdash x : T \] (T-Var)
\[ \Gamma \vdash \lambda x : T_1.e : T_1 \rightarrow T_2 \] (T-Abs)
\[ \Gamma \vdash e_1 : T_1 \rightarrow T \quad \Gamma \vdash e_2 : T_2 \quad (\text{T-App}) \]
\[ \Gamma \vdash \text{true} : \text{Bool} \] (T-True)
\[ \Gamma \vdash \text{false} : \text{Bool} \] (T-False)
\[ \Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 : T \] (T-If)