Programming Language Theory

Type Safety of the Simply-Typed Lambda Calculus
Outline

- Type-safety proof
  - Detailed proof posted on website
- Discuss the proof
  - Consider lemma dependencies and what they represent
  - Consider elegance of inverting static and dynamic derivations
- Next lecture: Extend STLC
  - Pairs, sums, recursion, ...
  - For each, sketch proof additions
  - Discuss the general approach

- On homework: References
- Further ahead: More flexible typing via polymorphism
Lambda Calculus (with constants) Syntax and Substitution

\[ e ::= c \mid x \mid \lambda x. \ e \mid e \ e \]

\[ v ::= c \mid \lambda x. \ e \]

Work with terms “up to renaming of bound variables” ("up to alpha-conversion").

\[
FV(c) = \{\} \quad FV(e_f \ e_a) = FV(e_f) \cup FV(e_a) \\
FV(x) = \{x\} \quad FV(\lambda x. \ e_b) = FV(e_b) \setminus \{x\}
\]

\[ e_1[e_2/x] = e_3 \]

\[
\begin{align*}
  c[e/x] &= c \\
  x[e/x] &= e \\
  y \neq x \quad y \not\in FV(e) \quad e_b[e/x] &= e'_b \\
  (\lambda y. \ e_b)[e/x] &= \lambda y. \ e'_b \\
  e_f[e/x] &= e'_f \\
  e_a[e/x] &= e'_a \\
  (e_f \ e_a)[e/x] &= e'_f \ e'_a
\end{align*}
\]

Substitution usually treated as a metafunction, not a judgement.
Lambda Calculus (with constants) Operational Semantics

Small-step, left-to-right, call-by-value (CBV) operational semantics:

\[ e \rightarrow_{\text{cbv}} e' \]

E-AppF
\[ e_f \rightarrow_{\text{cbv}} e'_f \]
\[ e_f e_a \rightarrow_{\text{cbv}} e'_f e_a \]
\[ E-\text{AppA} \]
\[ e_a \rightarrow_{\text{cbv}} e'_a \]
\[ v_f e_a \rightarrow_{\text{cbv}} v_f e'_a \]
\[ E-\text{Apply} \]
\[ (\lambda x. e_b) v_a \rightarrow_{\text{cbv}} e_b[v_a/x] \]

\[ e \rightarrow^*_{\text{cbv}} e' \]

E*-Zero
\[ e \rightarrow^*_{\text{cbv}} e \]

E*-Step
\[ e \rightarrow_{\text{cbv}} e^\dagger \]
\[ e^\dagger \rightarrow^*_{\text{cbv}} e' \]

We say that an expression \( e \) is stuck if

- \( e \) is not a value, and there is no \( e' \) such that \( e \rightarrow_{\text{cbv}} e' \)

We say that an expression \( e \) gets stuck if

- \( e \rightarrow^*_{\text{cbv}} e' \), and \( e' \) is stuck.
Simply-Typed Lambda Calculus (with constants) Type System

Type system to classify (accept or reject) $\lambda$-terms.

$$\tau ::= \text{int} \mid \tau \rightarrow \tau \quad \Gamma ::= \cdot \mid \Gamma, x : \tau$$

\[\Gamma @ x \leadsto \tau\]

\[\Gamma \vdash e : \tau\]

C-Hit

$$\frac{\Gamma, x : \tau_x @ x \leadsto \tau_x}{\Gamma, x : \tau_x @ x \leadsto \tau_x}$$

C-Miss

$$\frac{x \not= y \quad \Gamma' @ x \leadsto \tau}{\Gamma', y : \tau_y @ x \leadsto \tau}$$

T-Const

$$\frac{\Gamma @ x \leadsto \tau}{\Gamma \vdash c : \text{int}}$$

T-VAR

$$\frac{\Gamma @ x \leadsto \tau}{\Gamma \vdash x : \tau}$$

T-Lam

$$\frac{\Gamma, x : \tau_a @ e_b : \tau_r}{\Gamma \vdash \lambda x. e_b : \tau_a \rightarrow \tau_r}$$

T-App

$$\frac{\Gamma \vdash e_f : \tau_a \rightarrow \tau_r \quad \Gamma \vdash e_a : \tau_a}{\Gamma \vdash e_f e_a : \tau_r}$$
Type Soundness: Main Theorem and Lemmas

A program that type checks does not get stuck.

Theorem (Type Safety): If \( \cdot \vdash e : \tau \) and \( e \rightarrow_{cbv}^* e' \), then either \( e' \) is a value or there exists \( e'' \) such that \( e' \rightarrow_{cbv} e'' \).

Follows from two key lemmas:

- Lemma (Progress): If \( \cdot \vdash e' : \tau \), then either \( e' \) is a value or there exists an \( e'' \) such that \( e' \rightarrow_{cbv} e'' \).
- Lemma (Preservation): If \( \cdot \vdash e : \tau \) and \( e \rightarrow_{cbv} e^\dagger \), then \( \cdot \vdash e^\dagger : \tau \).

Proof of Type Safety given Progress and Preservation:

- By induction on (the derivation) \( e \rightarrow_{cbv}^* e' \).
  - \( e \rightarrow_{cbv}^* e' \equiv e \rightarrow_{cbv} e \rightarrow_{cbv} e' \): By Progress.
  - \( e \rightarrow_{cbv}^* e' \equiv e \rightarrow_{cbv} e \rightarrow_{cbv} e^\dagger \rightarrow_{cbv} e' \): By Preservation and IH.
Type Soundness: Auxiliary Lemmas

Lemma (Canonical Forms): If $\cdot \vdash \mathbf{v} : \tau$, then

1. if $\tau = \text{int}$, then $\mathbf{v} = \mathbf{c}$ (for some $\mathbf{c}$)
2. if $\tau = \tau_1 \rightarrow \tau_2$, then $\mathbf{v} = \lambda x. \mathbf{e}$ (for some $\lambda x. \mathbf{e}$)

Lemma (Substitution): If $\Gamma, x : \tau_x \vdash \mathbf{e}_1 : \tau$ and $\Gamma \vdash \mathbf{e}_2 : \tau_x$, then $\Gamma \vdash \mathbf{e}_1[\mathbf{e}_2/x] : \tau$.

Lemma (Exchange): If $\Gamma, x : \tau_x, y : \tau_y \vdash \mathbf{e} : \tau$ and $x \neq y$, then $\Gamma, y : \tau_y, x : \tau_x \vdash \mathbf{e} : \tau$.

Lemma (Weakening): If $\Gamma \vdash \mathbf{e} : \tau$ and $x \notin \text{Dom}(\Gamma)$, then $\Gamma, x : \tau_x \vdash \mathbf{e} : \tau$. 
Lemma dependencies

Safety (evaluation never gets stuck)
- Progress (well-typed not stuck yet)
  - Canonical Forms (primitive reductions apply)
- Preservation (to stay well-typed)
  - Substitution ($\beta$-reduction stays well-typed)
    - Weakening (substituting under nested $\lambda$s well-typed)
    - Exchange (technical point)

Comments:
- Substitution strengthened to open terms for the proof
- When we (and by “we”, I mean “you”) add heaps, Preservation will use Weakening directly
Summary

What may seem a weird lemma pile is a powerful recipe:

Soundness: We don’t get stuck because our induction hypothesis (well typedness) holds (Preservation) and stuck terms aren’t well typed (contrapositive of Progress).

Preservation holds by induction on typing because we replace a subterm with another subterm of same type; for the tricky subterm replacement (\(\beta\)-reduction), we use Substitution.

- Substitution must work over open terms and requires Weakening and Exchange.

Progress holds by induction on typing because either a subexpression progresses or we can make a primitive reduction (using Canonical Forms).
Induction on derivations – Another Look

The app cases are really elegant and worth mastering: $e = e_f \ e_a$.
For Preservation, lemma assumes $\Gamma \vdash e_f \ e_a : \tau$ and $e_f \ e_a \rightarrow_{cbv} e'$.

Inverting the typing derivation ensures that it has the form:

\[
\frac{D_f}{\Gamma \vdash e_f : \tau_a \rightarrow \tau} \quad \frac{D_a}{\Gamma \vdash e_a : \tau_a} \quad \frac{D}{\Gamma \vdash e_f \ e_a : \tau}
\]

One subcase: If $e_f \ e_a \rightarrow_{cbv} e'_f \ e_a$, then inverting that derivation ensures:

\[
\frac{D_s}{e_f \rightarrow_{cbv} e'_f} \quad \frac{e_f \ e_a \rightarrow_{cbv} e'_f \ e_a}
\]
continued...

The inductive hypothesis (applied to $\Gamma \vdash e_f : \tau_a \rightarrow \tau$ and $e_f \rightarrow_{cbv} e'_f$) gives a derivation of this form:

$$
\begin{align*}
D_{f'} \\
\Gamma \vdash e'_f : \tau_a \rightarrow \tau
\end{align*}
$$

Therefore, a derivation of this form exists:

$$
\begin{align*}
D_{f'} & \quad D_a \\
\Gamma \vdash e'_f : \tau_a \rightarrow \tau & \quad \Gamma \vdash e_a : \tau_a \\
\Gamma \vdash e'_f \ e_a : \tau
\end{align*}
$$

(The app case of the Substitution Lemma is similar but we apply induction twice to get the new derivation.)