CSE 505: Programming Languages

Lecture 12 — Safely Extending STLC: Progress, Preservation, Lets, Branches

Zach Tatlock
Fall 2013
Review

\[ e ::= \lambda x. e \mid x \mid e \, e \mid c \quad \tau ::= \text{int} \mid \tau \rightarrow \tau \]

\[ v ::= \lambda x. e \mid c \quad \Gamma ::= \cdot \mid \Gamma, x : \tau \]

\[ (\lambda x. \, e) \, v \rightarrow e[v/x] \]

\[ e_1 \rightarrow e'_1 \quad e_1 \, e_2 \rightarrow e'_1 \, e_2 \quad e_2 \rightarrow e'_2 \quad v \, e_2 \rightarrow v \, e'_2 \]

\[ e[e'/x]: \text{capture-avoiding substitution of } e' \text{ for free } x \text{ in } e \]

\[ \Gamma \vdash c : \text{int} \quad \Gamma \vdash x : \Gamma(x) \quad \Gamma \vdash \lambda x. \, e : \tau_1 \rightarrow \tau_2 \]

\[ \Gamma \vdash e_1 : \tau_2 \rightarrow \tau_1 \quad \Gamma \vdash e_2 : \tau_2 \]

\[ \Gamma \vdash e_1 \, e_2 : \tau_1 \]

Preservation: If \( \cdot \vdash e : \tau \) and \( e \rightarrow e' \), then \( \cdot \vdash e' : \tau \).

Progress: If \( \cdot \vdash e : \tau \), then \( e \) is a value or \( \exists e' \) such that \( e \rightarrow e' \).
Adding Stuff

Time to use STLC as a foundation for understanding other common language constructs

We will add things via a *principled methodology* thanks to a *proper education*

- Extend the syntax

- Extend the operational semantics
  - Derived forms (syntactic sugar), or
  - Direct semantics

- Extend the type system

- Extend soundness proof (new stuck states, proof cases)

In fact, extensions that add new types have even more structure
Let bindings (CBV)

\[ e ::= \ldots \mid \text{let } x = e_1 \text{ in } e_2 \]

\[ e_1 \rightarrow e_1' \]

\[
\begin{align*}
\text{let } x = e_1 \text{ in } e_2 & \rightarrow \text{let } x = e_1' \text{ in } e_2 \\
\text{let } x = v \text{ in } e & \rightarrow e[v/x]
\end{align*}
\]

\[ \Gamma \vdash e_1 : \tau' \quad \Gamma, x : \tau' \vdash e_2 : \tau \]

\[ \Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau \]

(Also need to extend definition of substitution...)

Progress: If \( e \) is a let, 1 of the 2 new rules apply (using induction)

Preservation: Uses Substitution Lemma

Substitution Lemma: Uses Weakening and Exchange
Derived forms

let seems just like \( \lambda \), so can make it a derived form

- \textbf{let } \( x = e_1 \textbf{ in } e_2 \) “a macro” / “desugars to” \( (\lambda x. \ e_2) \ e_1 \)

- A “derived form”

(Harder if \( \lambda \) needs explicit type)

Or just define the semantics to replace let with \( \lambda \):

\begin{align*}
\textbf{let } x = e_1 \textbf{ in } e_2 & \rightarrow (\lambda x. \ e_2) \ e_1 \\
\end{align*}

These 3 semantics are different in the state-sequence sense

\( (e_1 \rightarrow e_2 \rightarrow \ldots \rightarrow e_n) \)

- But (totally) equivalent and you could prove it (not hard)

Note: ML type-checks let and \( \lambda \) differently (later topic)

Note: Don’t desugar early if it hurts error messages!
Booleans and Conditionals

\[
e \ ::= \ldots \mid \text{true} \mid \text{false} \mid \text{if } e_1 \ e_2 \ e_3
\]

\[
v \ ::= \ldots \mid \text{true} \mid \text{false}
\]

\[
\tau \ ::= \ldots \mid \text{bool}
\]

\[
e_1 \rightarrow e'_1 \\
\frac{e_1 \rightarrow e'_1}{\text{if } e_1 \ e_2 \ e_3 \rightarrow \text{if } e'_1 \ e_2 \ e_3}
\]

\[
\frac{\text{if } \text{true} \ e_2 \ e_3 \rightarrow e_2}{
\frac{\Gamma \vdash e_1 : \text{bool}}{
\frac{\Gamma \vdash e_2 : \tau \quad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \text{if } e_1 \ e_2 \ e_3 : \tau}}}
\]

\[
\frac{\Gamma \vdash \text{true} : \text{bool}}{
\frac{\Gamma \vdash \text{false} : \text{bool}}{
\text{if } \text{false} \ e_2 \ e_3 \rightarrow e_3}}}
\]

Also extend definition of substitution (will stop writing that)...

Notes: CBN, new Canonical Forms case, all lemma cases easy