Programming Language Theory

Proofs
Looking back, looking forward

- Done: IMP abstract syntax, operational semantics
- Today and next: Detailed proofs (and some wrong turns) of two "theorems"
  - How to prove them
  - Why these theorems are "interesting"
- Future:
  - Pseudo-denotational Semantics (via translation to ML)
  - Equivalence (when are programs the "same"?)
Proofs

Write out proofs (by hand, on the board) for:

- **while 1 skip** diverges
  - Key point: Must get induction hypothesis “just right” — not too strong (false) or too weak (proof doesn’t go through)

- “No negatives” is preserved by evaluation
  - Can define a program property via judgements and inference rules and prove that it is preserved by every step
  - “Inverting assumed derivations” gives you the necessary facts for smaller expressions/statements (e.g., the *while* case)
Motivation for “no negatives” theorem

While “no negatives is preserved” boils down to properties of $+$ and $\ast$, writing out the whole proof ensures that our language has no mistakes or bad interactions.

The theorem is false if we have:

- Overly flexible rules, e.g.:

  $H; c \downarrow c'$

- An “unsafe” language like C:

  $H; e \downarrow c \quad H @ x \leadsto \langle c_0, \ldots, c_{n-1} \rangle \quad (0 > c \lor c \geq n)$

  $H; x[e] := e' \rightarrow H'; s'$
Even more general proofs to come

We defined the semantics.

Given our semantics, we established properties of programs and sets of programs.

More interesting is having multiple semantics:

▶ For what program states are they equivalent?
▶ For what notion of equivalence?

Or having a more abstract semantics (e.g., a type system) and asking if it is preserved under evaluation.

▶ (If $e$ has type $\tau$ and $e$ becomes $e'$, does $e'$ have type $\tau$?)
Review: **IMP** abstract syntax (programs and heaps)

\[ s ::= x := e \mid \text{skip} \mid s ; s \mid \text{if } e s s \mid \text{while } e s \]
\[ e ::= c \mid x \mid e + e \mid e \ast e \]
\[ (c \in \{\ldots, -2, -1, 0, 1, 2, \ldots\}) \]
\[ (x \in \{x_1, x_2, \ldots, y_1, y_2, \ldots, z_1, z_2, \ldots, \ldots\}) \]

\[ H ::= \cdot \mid H, x \mapsto c \]

<table>
<thead>
<tr>
<th>EMPTY</th>
<th>HIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>\cdot @ x \leadsto 0</td>
<td>(H', x \mapsto c @ x \leadsto c)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MISS</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x \neq y') \quad (H' @ x \leadsto c)</td>
</tr>
<tr>
<td>(H', y' \mapsto c' @ x \leadsto c)</td>
</tr>
</tbody>
</table>
Review: **IMP** operational semantics for expressions

\[ H; e \Downarrow c \]

**CONST**

\[ \frac{}{H; c \Downarrow c} \]

**ADD**

\[ \frac{H; e_1 \Downarrow c_1 \quad H; e_2 \Downarrow c_2}{H; e_1 + e_2 \Downarrow c_1 + c_2} \]

**VAR**

\[ \frac{H; @ x \Rightarrow c}{H; x \Downarrow c} \]

**MULT**

\[ \frac{H; e_1 \Downarrow c_1 \quad H; e_2 \Downarrow c_2}{H; e_1 * e_2 \Downarrow c_1 * c_2} \]
Review: **IMP** operational semantics for statements (small-step)

\[ H_1; s_1 \rightarrow H_2; s_2 \]

**Assign**

\[
\frac{H; e \downarrow c}{H; x := e \rightarrow H, x \mapsto c; \text{skip}}
\]

**While**

\[
\frac{H; \text{while } e \ s \rightarrow \ H; \text{if } e \ (s ; \text{while } e \ s) \ \text{skip}}{H; \text{while } e \ s \rightarrow}
\]

**SeqSkip**

\[
\frac{H; \text{skip } ; \ s \rightarrow H; s}{H; \text{skip } ; \ s \rightarrow H; s}
\]

**SeqStep**

\[
\frac{H; s_1 \rightarrow H'; s_1'}{H; s_1 ; \ s_2 \rightarrow H'; s_1' ; \ s_2}
\]

**IfT**

\[
\frac{H; e \downarrow c \quad c > 0}{H; \text{if } e \ s_1 \ s_2 \rightarrow H; s_1}
\]

**IfF**

\[
\frac{H; e \downarrow c \quad c \leq 0}{H; \text{if } e \ s_1 \ s_2 \rightarrow H; s_2}
\]
Review: **IMP** operational semantics for programs (small-step)

\[ H_1; s_1 \rightarrow^n H_2; s_2 \]

\[ H; s \rightarrow^0 H; s \]

\[ H_1; s_1 \rightarrow^* H_2; s_2 \]

\[ H_1; s_1 \rightarrow^m H_2; s_2 \]

\[ H_2; s_2 \rightarrow H_3; s_3 \]

\[ H_1; s_1 \rightarrow^{m+1} H_3; s_3 \]

\[ H; s \rightarrow^* c \]

\[ \cdot; s \rightarrow^* H; \text{skip} \]

\[ H \circ \text{ans} \rightsquigarrow c \]

\[ s \rightarrow^* c \]
“No Negative Constants” Judgements and Inference Rules

\[ \text{noneg}(e) \]

\[
\begin{array}{c}
\frac{c \geq 0}{\text{noneg}(c)} \\
\frac{\text{noneg}(e_1) \quad \text{noneg}(e_2)}{\text{noneg}(e_1 + e_2)}
\end{array}
\]

\[
\begin{array}{c}
\frac{}{\text{noneg}(x)} \\
\frac{\text{noneg}(e_1) \quad \text{noneg}(e_2)}{\text{noneg}(e_1 \times e_2)}
\end{array}
\]
“No Negative Constants” Judgements and Inference Rules

\[
\text{noneg}(s)
\]

\[
\begin{align*}
\text{noneg}(e) & \quad \text{noneg}(x := e) \\
\text{noneg}(\text{skip}) & \\
\text{noneg}(e) & \quad \text{noneg}(s_1) & \quad \text{noneg}(s_2) & \quad \text{noneg}(\text{if } e \ s_1 \ s_2) \\
\text{noneg}(s_1) & \quad \text{noneg}(s_2) & \quad \text{noneg}(s) & \quad \text{noneg}(\text{while } e \ s)
\end{align*}
\]
“No Negative Constants” Judgements and Inference Rules

\[
\text{noneg}(H)
\]

\[
\frac{\text{noneg}(\cdot)}{\text{noneg}(H, x \mapsto c)}
\]

\[
\frac{\text{noneg}(H) \quad c \geq 0}{\text{noneg}(H, x \mapsto c)}
\]