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 $\oplus$ 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:
  \[
  \begin{align*}
  H;e \Downarrow c & \quad H @ x \leadsto \langle c_0, \ldots, c_{n-1} \rangle & (0 > c \lor c \geq n) \\
  H;x[e] := e' & \rightarrow H';s'
  \end{align*}
  \]
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)

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

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

\[
H @ x \leadsto c
\]

**EMPTY**

\[
\begin{array}{c}
\cdot @ x \leadsto 0
\end{array}
\]

**HIT**

\[
\begin{array}{c}
H', x \mapsto c @ x \leadsto c
\end{array}
\]

**MISS**

\[
\begin{array}{c}
x \neq y' \quad H' @ x \leadsto c \\
H', y' \mapsto c' @ x \leadsto c
\end{array}
\]
Review: **IMP** operational semantics for expressions

\[ H; e \Downarrow c \]

**CONST**

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

**VAR**

\[
H \@ x \rightsquigarrow c
\]

\[
H; x \Downarrow c
\]

**ADD**

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

**MULT**

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

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

**SEQ_SKIP**

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

**SEQ_STEP**

\[
\frac{H; s_1 \rightarrow H'; s'_1}{H; s_1 \ ; \ s_2 \rightarrow H'; s'_1 \ ; \ s_2}
\]

**IF_T**

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

**IF_F**

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

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

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

\[ H_1; s_1 \rightarrow^m H_2; s_2 \quad \text{and} \quad H_2; s_2 \rightarrow H_3; s_3 \]

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

\[ s \rightarrow^* \text{c} \]

\[ H; \text{skip} \]

\[ H \oplus \text{ans} \leadsto \text{c} \]

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

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

\[ \begin{array}{c}
  c \geq 0 \\
  \hline
  \text{noneg}(c)
\end{array} \]

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

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

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

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

\[
\begin{align*}
\text{noneg}(H) & \quad \text{noneg}(H) \quad c \geq 0 \\
\text{noneg}(\cdot) & \quad \text{noneg}(H, x \mapsto c)
\end{align*}
\]