Announcements

• I’m here!
Announcements

Web Site

- My office hours
  - Usually Monday 10–11am and Friday 2–3pm
  - This week only, Thursday 1–2pm instead of Friday

- Grading details
  - Three 24-hour slip days
  - You can use at most two per assignment
  - Lowest score dropped

- Slides and notes posted there from now on (instead of Piazza)

Homework #1

- Due: Today, 11:59pm

Homework #2

- Out: Today
Last time we defined the IMP programming language...

\[
\begin{align*}
a & ::= x \mid n \mid a_1 + a_2 \mid a_1 \times a_2 \\
 b & ::= \text{true} \mid \text{false} \mid a_1 < a_2 \\
c & ::= \text{skip} \\
 & \quad | \quad x ::= a \\
 & \quad | \quad c_1; c_2 \\
 & \quad | \quad \text{if } b \text{ then } c_1 \text{ else } c_2 \\
 & \quad | \quad \text{while } b \text{ do } c
\end{align*}
\]
Again, three relations, one for each syntactic category:

\[
\downarrow_{\text{Aexp}} \subseteq \text{Store} \times \text{Aexp} \times \text{Int} \\
\downarrow_{\text{Bexp}} \subseteq \text{Store} \times \text{Bexp} \times \text{Bool} \\
\downarrow_{\text{Com}} \subseteq \text{Store} \times \text{Com} \times \text{Store}
\]
Large-Step Semantics

\[
\langle \sigma, n \rangle \Downarrow n
\]

\[
\sigma(x) = n
\]

\[
\langle \sigma, x \rangle \Downarrow n
\]

\[
\langle \sigma, e_1 \rangle \Downarrow n_1 \quad \langle \sigma, e_2 \rangle \Downarrow n_2 \quad n = n_1 + n_2
\]

\[
\langle \sigma, e_1 + e_2 \rangle \Downarrow n
\]

\[
\langle \sigma, e_1 \rangle \Downarrow n_1 \quad \langle \sigma, e_2 \rangle \Downarrow n_2 \quad n = n_1 \times n_2
\]

\[
\langle \sigma, e_1 \times e_2 \rangle \Downarrow n
\]
Large-Step Semantics

\[ \langle \sigma, \text{true} \rangle \Downarrow \text{true} \]

\[ \langle \sigma, \text{false} \rangle \Downarrow \text{false} \]

\[ \langle \sigma, a_1 \rangle \Downarrow n_1 \quad \langle \sigma, a_2 \rangle \Downarrow n_2 \quad n_1 < n_2 \]

\[ \langle \sigma, a_1 < a_2 \rangle \Downarrow \text{true} \]

\[ \langle \sigma, a_1 \rangle \Downarrow n_1 \quad \langle \sigma, a_2 \rangle \Downarrow n_2 \quad n_1 \geq n_2 \]

\[ \langle \sigma, a_1 < a_2 \rangle \Downarrow \text{false} \]
Large-Step Semantics

\[
\text{SKIP} \quad \frac{}{\langle \sigma, \text{skip} \rangle \downarrow \sigma}
\]
Large-Step Semantics

\[ \text{ASSGN} \quad \frac{\langle \sigma, e \rangle \downarrow n}{\langle \sigma, x := e \rangle \downarrow \sigma[x \mapsto n]} \]
Large-Step Semantics

\[ \text{SEQ} \quad \frac{\langle \sigma, c_1 \rangle \Downarrow \sigma'}{\langle \sigma, c_1; c_2 \rangle \Downarrow \sigma''} \quad \frac{\langle \sigma', c_2 \rangle \Downarrow \sigma''}{\langle \sigma, c_1; c_2 \rangle \Downarrow \sigma''} \]
Large-Step Semantics

\[\text{IF-T} \quad \frac{\langle \sigma, b \rangle \Downarrow \text{true} \quad \langle \sigma, c_1 \rangle \Downarrow \sigma'}{\langle \sigma, \text{if } b \text{ then } c_1 \text{ else } c_2 \rangle \Downarrow \sigma'}\]

\[\text{IF-F} \quad \frac{\langle \sigma, b \rangle \Downarrow \text{false} \quad \langle \sigma, c_2 \rangle \Downarrow \sigma'}{\langle \sigma, \text{if } b \text{ then } c_1 \text{ else } c_2 \rangle \Downarrow \sigma'}\]
**Large-Step Semantics**

\[ \text{WHILE-F:} \quad \langle \sigma, b \rangle \downarrow \text{false} \]
\[ \quad \Rightarrow \quad \langle \sigma, \text{while } b \text{ do } c \rangle \downarrow \sigma \]

\[ \text{WHILE-T:} \quad \langle \sigma, b \rangle \downarrow \text{true} \]
\[ \quad \quad \langle \sigma, c \rangle \downarrow \sigma' \]
\[ \quad \quad \langle \sigma', \text{while } b \text{ do } c \rangle \downarrow \sigma'' \]
\[ \quad \Rightarrow \quad \langle \sigma, \text{while } b \text{ do } c \rangle \downarrow \sigma'' \]
Command Equivalence

Intuitively, two commands are equivalent if they produce the same result under any store...

Definition (Equivalence of commands)

Two commands $c$ and $c'$ are equivalent (written $c \sim c'$) if, for any stores $\sigma$ and $\sigma'$, we have

$$
\langle \sigma, c \rangle \Downarrow \sigma' \iff \langle \sigma, c' \rangle \Downarrow \sigma'.
$$
For example, we can prove that every \texttt{while} command is equivalent to its “unrolling”:

\textbf{Theorem}

\textit{For all }$b \in \texttt{Bexp}$\textit{ and }$c \in \texttt{Com}$\textit{ we have}

\[ \texttt{while } b \texttt{ do } c \sim \texttt{if } b \texttt{ then } (c; \texttt{while } b \texttt{ do } c) \texttt{ else skip}. \]

\textbf{Proof.}

We show each implication separately...
IMP Questions

- Q: Can you write a program that doesn’t terminate?
IMP Questions

- Q: Can you write a program that doesn’t terminate?
  - A: `while true do skip`

- Q: Does this mean that IMP is Turing complete?
  - A: Not quite... we also need to check the language is not finite state... but IMP has real mathematical integers.

- Q: What if we replace `Int` with `Int64`?
  - A: Then we would lose Turing completeness.

- Q: How much space do we need to represent configurations during execution of an IMP program?
  - A: Can calculate a fixed bound!
IMP Questions

- Q: Can you write a program that doesn’t terminate?
  - A: `while true do skip`

- Q: Does this mean that IMP is Turing complete?
  - A: Not quite... we also need to check the language is not finite state... but IMP has real mathematical integers.
IMP Questions

- Q: Can you write a program that doesn’t terminate?
  A: `while true do skip`

- Q: Does this mean that IMP is Turing complete?
  A: Not quite... we also need to check the language is not finite state... but IMP has real mathematical integers.
IMP Questions

- Q: Can you write a program that doesn’t terminate?
  - A: `while true do skip`

- Q: Does this mean that IMP is Turing complete?
  - A: Not quite... we also need to check the language is not finite state... but IMP has real mathematical integers.

- Q: What if we replace `Int` with `Int64`?
Q: Can you write a program that doesn’t terminate?
A: while true do skip

Q: Does this mean that IMP is Turing complete?
A: Not quite... we also need to check the language is not finite state... but IMP has real mathematical integers.

Q: What if we replace Int with Int64?
A: Then we would lose Turing completeness.
Q: Can you write a program that doesn’t terminate?
A: `while true do skip`

Q: Does this mean that IMP is Turing complete?
A: Not quite... we also need to check the language is not finite state... but IMP has real mathematical integers.

Q: What if we replace `Int` with `Int64`?
A: Then we would lose Turing completeness.

Q: How much space do we need to represent configurations during execution of an IMP program?
IMP Questions

- Q: Can you write a program that doesn’t terminate?
  - A: `while true do skip`

- Q: Does this mean that IMP is Turing complete?
  - A: Not quite... we also need to check the language is not finite state... but IMP has real mathematical integers.

- Q: What if we replace `Int` with `Int64`?
  - A: Then we would lose Turing completeness.

- Q: How much space do we need to represent configurations during execution of an IMP program?
  - A: Can calculate a fixed bound!
Determinism

Theorem

∀c ∈ \textbf{Com}, σ, σ', σ'' ∈ \textbf{Store}.

if ⟨σ, c⟩ ↓ σ' and ⟨σ, c⟩ ↓ σ'' then σ' = σ''.
Theorem

\[ \forall c \in \text{Com}, \sigma, \sigma', \sigma'' \in \text{Store}.
\text{if } (\sigma, c) \downarrow \sigma' \text{ and } (\sigma, c) \downarrow \sigma'' \text{ then } \sigma' = \sigma''. \]

Proof.

By structural induction on c...
**Theorem**

\[ \forall c \in \textbf{Com}, \sigma, \sigma', \sigma'' \in \textbf{Store}. \]

*if* \( \langle \sigma, c \rangle \downarrow \sigma' \) *and* \( \langle \sigma, c \rangle \downarrow \sigma'' \) *then* \( \sigma' = \sigma'' \).

**Proof.**

By structural induction on \( c \)...

**Proof.**

By induction on the derivation of \( \langle \sigma, c \rangle \downarrow \sigma' \)...

---

**Determinism**
Derivations

Write $\mathcal{D} \vdash y$ if the conclusion of derivation $\mathcal{D}$ is $y$. 
Write $D \models y$ if the conclusion of derivation $D$ is $y$.

**Example:**

Given the derivation,

\[
\begin{align*}
\langle \sigma, 6 \rangle & \downarrow 6 \\
\langle \sigma, 7 \rangle & \downarrow 7 \\
\langle \sigma, 6 \times 7 \rangle & \downarrow 42 \\
\langle \sigma, i := 6 \times 7 \rangle & \downarrow \sigma[i \mapsto 42]
\end{align*}
\]

we would write: $D \models \langle \sigma, i := 42 \rangle \downarrow \sigma[i \mapsto 42]$
Induction on Derivations

Given a set of axioms and inference rules, the set of derivations is itself an inductively defined set!
Induction on Derivations

Given a set of axioms and inference rules, the set of derivations is itself an inductively defined set!

This means we can prove properties by induction on derivations!
Induction on Derivations

Given a set of axioms and inference rules, the set of derivations is itself an inductively defined set!

This means we can prove properties by induction on derivations!

A derivation $\mathcal{D}'$ is an immediate subderivation of $\mathcal{D}$ if $\mathcal{D}' \vdash z$ where $z$ is one of the premises used of the final rule of derivation $\mathcal{D}$. 
Induction on Derivations

Given a set of axioms and inference rules, the set of derivations is itself an inductively defined set!

This means we can prove properties by induction on derivations!

A derivation $\mathcal{D}'$ is an immediate subderivation of $\mathcal{D}$ if $\mathcal{D}' \vdash z$ where $z$ is one of the premises used of the final rule of derivation $\mathcal{D}$.

In a proof by induction on derivations, for every axiom and inference rule, assume that the property $P$ holds for all immediate subderivations, and show that it holds of the conclusion.
### Large-Step Semantics

<table>
<thead>
<tr>
<th>Rule</th>
<th>Precondition</th>
<th>Postcondition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Skip</strong></td>
<td>$\langle \sigma, \text{skip} \rangle \downarrow \sigma$</td>
<td>$\langle \sigma, a \rangle \Downarrow n$</td>
</tr>
<tr>
<td><strong>Assign</strong></td>
<td>$\langle \sigma, x := a \rangle \Downarrow \sigma[x \mapsto n]$</td>
<td>$\langle \sigma, c_1 \rangle \Downarrow \sigma'$</td>
</tr>
<tr>
<td><strong>Seq</strong></td>
<td>$\langle \sigma, c_1 \rangle \Downarrow \sigma'$ $\langle \sigma', c_2 \rangle \Downarrow \sigma''$</td>
<td>$\langle \sigma, c_1 ; c_2 \rangle \Downarrow \sigma''$</td>
</tr>
<tr>
<td><strong>If-T</strong></td>
<td>$\langle \sigma, b \rangle \Downarrow \text{true}$ $\langle \sigma, c_1 \rangle \Downarrow \sigma'$</td>
<td>$\langle \sigma, \text{if } b \text{ then } c_1 \text{ else } c_2 \rangle \Downarrow \sigma'$</td>
</tr>
<tr>
<td><strong>If-F</strong></td>
<td>$\langle \sigma, b \rangle \Downarrow \text{false}$ $\langle \sigma, c_2 \rangle \Downarrow \sigma'$</td>
<td>$\langle \sigma, \text{if } b \text{ then } c_1 \text{ else } c_2 \rangle \Downarrow \sigma'$</td>
</tr>
<tr>
<td><strong>While-T</strong></td>
<td>$\langle \sigma, b \rangle \Downarrow \text{true}$ $\langle \sigma, c \rangle \Downarrow \sigma'$ $\langle \sigma', \text{while } b \text{ do } c \rangle \Downarrow \sigma''$</td>
<td>$\langle \sigma, \text{while } b \text{ do } c \rangle \Downarrow \sigma''$</td>
</tr>
<tr>
<td><strong>While-F</strong></td>
<td>$\langle \sigma, b \rangle \Downarrow \text{false}$</td>
<td>$\langle \sigma, \text{while } b \text{ do } c \rangle \Downarrow \sigma$</td>
</tr>
</tbody>
</table>