Programming Language Theory

Equivalence
Looking back, looking forward

Done: IMP

- abstract syntax
- operational semantics (large-step and small-step)
- semantic properties of (sets of) programs — proofs
- “pseudo-denotational” semantics

Today: Equivalence

- equivalence of programs in a semantics
- equivalence of different semantics

Next: $\lambda$-calculus
Equivalence motivation

- Program equivalence (we change the program):
  - code optimizer
  - code maintainer
- Semantics equivalence (we change the language):
  - compiler correctness
  - interpreter optimizer
  - language designer
    - (prove properties for equivalent semantics with easier proof)

Warning: Proofs are easy with the right semantics and lemmas
- (almost never start off with right semantics and lemmas)

Note: Small-step operational semantics often has harder proofs, but models more interesting things
What is equivalence?

Equivalence depends on what is observable!
What is equivalence?

Equivalence depends on what is observable!

- Partial I/O equivalence (if terminates, same ans)
  - `while 1 skip` equivalent to everything
  - not transitive
What is equivalence?

Equivalence depends on what is observable!

- Partial I/O equivalence (if terminates, same ans)
  - while 1 skip equivalent to everything
  - not transitive
- Total I/O equivalence (same termination behavior, same ans)
What is equivalence?

Equivalence depends on what is observable!

- Partial I/O equivalence (if terminates, same ans)
  - while 1 skip equivalent to everything
  - not transitive

- Total I/O equivalence (same termination behavior, same ans)

- Total heap equivalence (same termination behavior, same heaps)
  - heaps are syntactically equal
  - all variables have the same value
  - almost all variables have the same value

Equivalence plus complexity bounds

- Is $O(2^n)$ really equivalent to $O(n)$?

Syntactic equivalence (perhaps with renaming)

- too strict to be interesting
What is equivalence?

Equivalence depends on what is observable!

- Partial I/O equivalence (if terminates, same ans)
  - while 1 skip equivalent to everything
  - not transitive
- Total I/O equivalence (same termination behavior, same ans)
- Total heap equivalence (same termination behavior, same heaps)
  - heaps are syntactically equal
  - all variables have the same value
  - almost all variables have the same value
- Equivalence plus complexity bounds
  - Is $O(2^n)$ really equivalent to $O(n)$?
What is equivalence?

Equivalence depends on **what is observable!**

- **Partial I/O equivalence** (if terminates, same **ans**)
  - **while 1 skip** equivalent to everything
  - not transitive

- **Total I/O equivalence** (same termination behavior, same **ans**)

- **Total heap equivalence** (same termination behavior, same heaps)
  - heaps are syntactically equal
  - all variables have the same value
  - almost all variables have the same value

- **Equivalence plus complexity bounds**
  - Is $O(2^n)$ really equivalent to $O(n)$?

- **Syntactic equivalence** (perhaps with renaming)
  - too strict to be interesting
What is equivalence?

Equivalence depends on what is observable!

- Partial I/O equivalence (if terminates, same ans)
  - while 1 skip equivalent to everything
  - not transitive

- Total I/O equivalence (same termination behavior, same ans)

- Total heap equivalence (same termination behavior, same heaps)
  - heaps are syntactically equal
  - all variables have the same value
  - almost all variables have the same value

- Equivalence plus complexity bounds
  - Is $O(2^n)$ really equivalent to $O(n)$?

- Syntactic equivalence (perhaps with renaming)
  - too strict to be interesting
Program example: Strength reduction

Motivation: Strength reduction
▶ a common compiler optimization due to architecture issues

Theorem:

\[ H;e \ast 2 \downarrow c \text{ if and only if } H;e + e \downarrow c \]

Proof Sketch:
▶ Just need “inversion on derivation” and math
▶ no induction
Program example: Nested strength reduction

Theorem:

If \( e_1 \) has a subexpression of the form \( e \ast 2 \),
then \( H; e_1 \Downarrow c' \) if and only if \( H; e_2 \Downarrow c' \)
where \( e_2 \) is \( e_1 \) with \( e \ast 2 \) replaced by \( e + e \).
Program example: Nested strength reduction

Theorem:

If $e_1$ has a subexpression of the form $e \ast 2$, then $H;e_1 \downarrow c'$ if and only if $H;e_2 \downarrow c'$ where $e_2$ is $e_1$ with $e \ast 2$ replaced by $e + e$.

First, some useful meta-notation:

\[ C ::= [\cdot] \mid C + e \mid e + C \mid C \ast e \mid e \ast C \]

$C[e]$ is “$C$ with $e$ plugged into the hole”.
Program example: Nested strength reduction

Theorem:

If $e_1$ has a subexpression of the form $e \ast 2$, then $H; e_1 \downarrow c'$ if and only if $H; e_2 \downarrow c'$ where $e_2$ is $e_1$ with $e \ast 2$ replaced by $e + e$.

First, some useful meta-notation:

$$C ::= [\cdot] \mid C + e \mid e + C \mid C \ast e \mid e \ast C$$

$C[e]$ is “$C$ with $e$ plugged into the hole”.

Theorem:

If $(e_1 = C[e \ast 2]$ and $e_2 = C[e + e]$) then $(H; e_1 \downarrow c'$ if and only if $H; e_2 \downarrow c'$).

Proof sketch:

- By structural induction on $C$. 

Matthew Fluet
Programming Language Theory
Lecture 06
Small-step program equivalence

These sort of proofs also work with small-step semantics (e.g., IMP statements), but tend to be more cumbersome, even to state.

Example: The statement-sequence operator is associative. That is,

- For all $n$, if $H; s_1 ; (s_2 ; s_3) \rightarrow^n H'; \text{skip}$, then there exist $H''$ and $n'$ such that $H;(s_1 ; s_2) ; s_3 \rightarrow^{n'} H''; \text{skip}$ and $H''(\text{ans}) = H'(\text{ans})$.

- If for all $n$ there exist $H'$ and $s'$ such that $H; s_1 ; (s_2 ; s_3) \rightarrow^n H'; s'$, then for all $n$ there exist $H''$ and $s''$ such that $H;(s_1 ; s_2) ; s_3 \rightarrow^n H''; s''$.

(Proof needs a much stronger induction hypothesis.)

One way to avoid it: Prove large-step and small-step semantics equivalent, then prove program equivalences in whichever semantics is easier.
Language example

**IMP w/o multiply (large-step):**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CONST</strong></td>
<td>( H; c \downarrow c )</td>
</tr>
<tr>
<td><strong>VAR</strong></td>
<td>( H @ x \sim c )</td>
</tr>
<tr>
<td><strong>ADD</strong></td>
<td>( H; e_1 \downarrow c_1 ) ( H; e_2 \downarrow c_2 ) ( H; e_1 + e_2 \downarrow c_1 + c_2 )</td>
</tr>
</tbody>
</table>

**IMP w/o multiply (small-step):**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SVAR</strong></td>
<td>( H @ x \sim c )</td>
</tr>
<tr>
<td><strong>SADD</strong></td>
<td>( H; c_1 + c_2 \rightarrow c_1 + c_2 )</td>
</tr>
<tr>
<td><strong>SADDL</strong></td>
<td>( H; e_1 \rightarrow e'_1 )</td>
</tr>
<tr>
<td><strong>SADDR</strong></td>
<td>( H; e_2 \rightarrow e'_2 )</td>
</tr>
</tbody>
</table>

\( H; e_1 + e_2 \rightarrow e'_1 + e'_2 \)

Theorem: Semantics are equivalent; \( H; e \downarrow c \) if and only if \( H; e \rightarrow^* c \).

Proof: We prove the two directions separately.
Proof, part 1:

Forward: Assume $H;e \downarrow c$; show $\exists n. H;e \rightarrow^n c$.

Lemma (prove it!): If $H;e \rightarrow n e'$, then $H;e_1 + e \rightarrow n e_1' + e'$ and $H;e_2 \rightarrow n e_2'$. 

▶ (Proof uses $[\text{saddl}]$ and $[\text{saddr}]$.)

Now, prove by structural induction on (the derivation of) $H;e \downarrow c$:

▶ $[\text{const}]$: Derivation is via $[\text{const}]$ and $e$ is $c$, so derive $H;c \rightarrow 0 c$.

▶ $[\text{var}]$: Derivation is via $[\text{var}]$ and $e$ is $x$ and $H@x;c$, so derive, by $[\text{svar}]$ using $H@x;c$, $H;x \rightarrow 1 c$.

▶ $[\text{add}]$: Derivation is via $[\text{add}]$ and $e$ is $e_1 + e_2$, $H;e_1 \downarrow c_1$, $H;e_2 \downarrow c_2$, and $c$ is $c_1 + c_2$. By induction, $\exists n_1. H;e_1 \rightarrow n_1 c_1$ and $\exists n_2. H;e_2 \rightarrow n_2 c_2$. By our lemma, $H;e_1 + e_2 \rightarrow n_1 c_1 + e_2$ and $H;c_1 + e_2 \rightarrow n_2 c_1 + c_2$. Derive, by $[\text{sadd}]$ and $c$ is $c_1 + c_2$, $H;c_1 + c_2 \rightarrow c$.

So derive $H;e_1 + e_2 \rightarrow n_1 + n_2 c$.
Proof, part 1:

Forward: Assume $H; e \Downarrow c$; show $\exists n. H; e \rightarrow^n c$.

Lemma (prove it!): If $H; e \rightarrow^n e'$, then $H; e_1 + e \rightarrow^n e_1 + e'$ and $H; e + e_2 \rightarrow^n e' + e_2$.

$\blacktriangleright$ (Proof uses [SADDR] and [SADDR].)
Proof, part 1:

Forward: Assume $H;e \Downarrow c$; show $\exists n. H;e \rightarrow^n c$.

Lemma (prove it!): If $H;e \rightarrow^n e'$, then $H;e_1 + e \rightarrow^n e_1 + e'$ and $H;e + e_2 \rightarrow^n e' + e_2$.

▶ (Proof uses [SADDL] and [SADDR].)

Now, prove by structural induction on (the derivation of) $H;e \Downarrow c$: 
Proof, part 1:

Forward: Assume \( H;e \Downarrow c \); show \( \exists n. \, H;e \rightarrow^n c \).

Lemma (prove it!): If \( H;e \rightarrow^n e' \),
then \( H;e_1 + e \rightarrow^n e_1 + e' \) and \( H;e + e_2 \rightarrow^n e' + e_2 \).

▷ (Proof uses \([SADDL]\) and \([SADDR]\).)

Now, prove by structural induction on (the derivation of) \( H;e \Downarrow c \):

▷ \([\text{CONST}]\): Derivation is via \([\text{CONST}]\) and \( e \) is \( c \), so derive \( H;c \rightarrow^0 c \).
Proof, part 1:

Forward: Assume $H;e \Downarrow c$; show $\exists n. H;e \rightarrow^n c$.

Lemma (prove it!): If $H;e \rightarrow^n e'$, then $H;e_1 + e \rightarrow^n e_1 + e'$ and $H;e + e_2 \rightarrow^n e' + e_2$.

▶ (Proof uses [SADDL] and [SADDR].)

Now, prove by structural induction on (the derivation of) $H;e \Downarrow c$:

▶ [CONST]: Derivation is via [CONST] and $e$ is $c$, so derive $H;c \rightarrow^0 c$.

▶ [VAR]: Derivation is via [VAR] and $e$ is $x$ and $H @ x \sim c$, so derive, by [SVAR] using $H @ x \sim c$, $H;x \rightarrow^1 c$. 
Proof, part 1:

Forward: Assume $H;e \Downarrow c$; show $\exists n. H;e \rightarrow^n c$.

Lemma (prove it!): If $H;e \rightarrow^n e'$, then $H;e_1 + e \rightarrow^n e_1 + e'$ and $H;e + e_2 \rightarrow^n e' + e_2$.

▶ (Proof uses [SADDL] and [SADDR].)

Now, prove by structural induction on (the derivation of) $H;e \Downarrow c$:

▶ [CONST]: Derivation is via [CONST] and $e$ is $c$, so derive $H;c \rightarrow^0 c$.

▶ [VAR]: Derivation is via [VAR] and $e$ is $x$ and $H @ x \rightsquigarrow c$, so derive, by [SVAR] using $H @ x \rightsquigarrow c$, $H;x \rightarrow^1 c$.

▶ [ADD]: Derivation is via [ADD] and $e$ is $e_1 + e_2$, $H;e_1 \Downarrow c_1$, $H;e_2 \Downarrow c_2$, and $c$ is $c_1 + c_2$.

By induction, $\exists n_1. H;e_1 \rightarrow^{n_1} c_1$ and $\exists n_2. H;e_2 \rightarrow^{n_2} c_2$.

By our lemma, $H;e_1 + e_2 \rightarrow^{n_1} c_1 + e_2$ and $H;c_1 + e_2 \rightarrow^{n_2} c_1 + c_2$.

Derive, by [SADD] and $c$ is $c_1 + c_2$, $H;c_1 + c_2 \rightarrow c$.

So derive $H;e_1 + e_2 \rightarrow^{n_1+n_2+1} c$. 
Proof, part 2:

Backward: Assume $\exists n. H;e \rightarrow^n c$; show $H;e \downarrow c$. 
Proof, part 2:

Backward: Assume $\exists n. \, H; e \rightarrow^n c$; show $H; e \downarrow c$.

Prove by induction on $n$:
Proof, part 2:

Backward: Assume $\exists n. \ H;e \rightarrow^n c$; show $\ H;e \downarrow c$.

Prove by induction on $n$:

$\triangleright \ n = 0$: $e$ is $c$ and $[\text{CONST}]$ lets us derive $\ H;c \downarrow c$. 
Proof, part 2:

Backward: Assume $\exists n. \; H;e \rightarrow^n c$; show $H;e \downarrow c$.

Prove by induction on $n$:

- $n = 0$: $e$ is $c$ and $[\text{CONST}]$ lets us derive $H;c \downarrow c$.
- $n = m + 1$: $\exists e'. \; H;e \rightarrow e'$ and $H;e' \rightarrow^m c$.

By induction (on $H;e' \rightarrow^m c$), we have $H;e' \downarrow c$.

So this lemma suffices: If $H;e \rightarrow e'$ and $H;e' \downarrow c$, then $H;e \downarrow c$. 
Proof, part 2:

Backward: Assume $\exists n. \ H;e \rightarrow^n c$; show $H;e \downarrow c$.

Prove by induction on $n$:

- $n = 0$: $e$ is $c$ and $[\text{CONST}]$ lets us derive $H;c \downarrow c$.
- $n = m + 1$: $\exists e'. \ H;e \rightarrow e'$ and $H;e' \rightarrow^m c$.
  
  By induction (on $H;e' \rightarrow^m c$), we have $H;e' \downarrow c$.
  
  So this lemma suffices: If $H;e \rightarrow e'$ and $H;e' \downarrow c$, then $H;e \downarrow c$.

  Prove the lemma by structural induction on (the derivation of) $H;e \rightarrow e'$:
Proof, part 2:

Backward: Assume $\exists n. H;e \xrightarrow{n} c$; show $H;e \Downarrow c$.

Prove by induction on $n$:

- $n = 0$: $e$ is $c$ and $[\text{CONST}]$ lets us derive $H;c \Downarrow c$.

- $n = m + 1$: $\exists e'. H;e \xrightarrow{} e'$ and $H;e' \xrightarrow{m} c$.
  
  By induction (on $H;e' \xrightarrow{m} c$), we have $H;e' \Downarrow c$.
  
  So this lemma suffices: If $H;e \xrightarrow{} e'$ and $H;e' \Downarrow c$, then $H;e \Downarrow c$.

Prove the lemma by structural induction on (the derivation of) $H;e \xrightarrow{} e'$:

- $[\text{SVAR}]$: Derivation is via $[\text{SVAR}]$ and $e$ is $x$, $H@x \leadsto c$, and $e'$ is $c$,
  so derive, by $[\text{VAR}]$ and $H@x \leadsto c$, $H;x \Downarrow c$. 
Proof, part 2:

Backward: Assume ∃\(n\). \(H;e \rightarrow^n c\); show \(H;e \downarrow c\).

Prove by induction on \(n\):

- \(n = 0\): \(e\) is \(c\) and \([\text{CONST}]\) lets us derive \(H;c \downarrow c\).

- \(n = m + 1\): ∃\(e'\). \(H;e \rightarrow e'\) and \(H;e' \rightarrow^m c\).
  
  By induction (on \(H;e' \rightarrow^m c\)), we have \(H;e' \downarrow c\).
  
  So this lemma suffices: If \(H;e \rightarrow e'\) and \(H;e' \downarrow c\), then \(H;e \downarrow c\).

Prove the lemma by structural induction on (the derivation of) \(H;e \rightarrow e'\):

- \([\text{SVAR}]\): Derivation is via \([\text{SVAR}]\) and \(e\) is \(x\), \(H @ x \rightsquigarrow c\), and \(e'\) is \(c\), so derive, by \([\text{VAR}]\) and \(H @ x \rightsquigarrow c\), \(H;x \downarrow c\).

- \([\text{SADD}]\): Derivation is via \([\text{SADD}]\) and \(e\) is \(c_1 + c_2\) and \(e'\) is \(c_1+c_2\), so derive, by \([\text{CONST}]\) and \([\text{ADD}]\), \(H;c_1 + c_2 \downarrow c_1+c_2\).
Proof, part 2:

Backward: Assume $\exists n. H;e \rightarrow^n c$; show $H;e \Downarrow c$.

Prove by induction on $n$:

$n = 0$: $e$ is $c$ and $[\text{CONST}]$ lets us derive $H;c \Downarrow c$.

$n = m + 1$: $\exists e'$. $H;e \rightarrow e'$ and $H;e' \rightarrow^m c$.
By induction (on $H;e' \rightarrow^m c$), we have $H;e' \Downarrow c$.
So this lemma suffices: If $H;e \rightarrow e'$ and $H;e' \Downarrow c$, then $H;e \Downarrow c$.

Prove the lemma by structural induction on (the derivation of) $H;e \rightarrow e'$:

$s\text{VAR}$: Derivation is via $[\text{SVAR}]$ and $e$ is $x$, $H @ x \rightsquigarrow c$, and $e'$ is $c$, so derive, by $[\text{VAR}]$ and $H @ x \rightsquigarrow c$, $H;x \Downarrow c$.

$s\text{ADD}$: Derivation is via $[\text{SADD}]$ and $e$ is $c_1 + c_2$ and $e'$ is $c_1 + c_2$, so derive, by $[\text{CONST}]$ and $[\text{ADD}]$, $H;c_1 + c_2 \Downarrow c_1 + c_2$.

$s\text{ADDR}$: . . .

$s\text{ADDL}$: . . .
Proof, part 2 (cont’d):

- \( n = m + 1 \): \( \exists e'. H;e \rightarrow e' \) and \( H;e' \rightarrow^m c \).

  By induction (on \( H;e' \rightarrow^m c \)), we have \( H;e' \Downarrow c \).

  So this lemma suffices: If \( H;e \rightarrow e' \) and \( H;e' \Downarrow c \), then \( H;e \Downarrow c \).

Prove the lemma by structural induction on (the derivation of) \( H;e \rightarrow e' \):

- [SADDL]: Derivation is via [SADDL]
  and \( e \) is \( e_1 + e_2 \), \( H;e_1 \rightarrow e_1' \), and \( e' \) is \( e_1' + e_2 \).
Proof, part 2 (cont’d):

- \( n = m + 1 \): \( \exists e'. \ H;e \rightarrow e' \) and \( H;e' \rightarrow^m c. \)

  By induction (on \( H;e' \rightarrow^m c \)), we have \( H;e' \Downarrow c \).

  So this lemma suffices: If \( H;e \rightarrow e' \) and \( H;e' \Downarrow c \), then \( H;e \Downarrow c \).

Prove the lemma by structural induction on (the derivation of) \( H;e \rightarrow e' \):

- \([\text{saddl}]:\) Derivation is via \([\text{saddl}]\)
  and \( e \) is \( e_1 + e_2 \), \( H;e_1 \rightarrow e'_1 \), and \( e' \) is \( e'_1 + e_2 \).

  By \( H;e'_1 + e_2 \Downarrow c \) and inversion,

  \( H;e'_1 \Downarrow c_1 \), \( H;e_2 \Downarrow c_2 \), and \( c \) is \( c_1 + c_2 \).
Proof, part 2 (cont’d):

- \( n = m + 1 \): \( \exists e'. \; H;e \rightarrow e' \) and \( H;e' \rightarrow^m c \).

  By induction (on \( H;e' \rightarrow^m c \)), we have \( H;e' \downarrow c \).

  So this lemma suffices: If \( H;e \rightarrow e' \) and \( H;e' \downarrow c \), then \( H;e \downarrow c \).

Prove the lemma by structural induction on (the derivation of) \( H;e \rightarrow e' \):

- \([\text{SADDL}]\): Derivation is via \([\text{SADDL}]\)
  and \( e \) is \( e_1 + e_2 \), \( H;e_1 \rightarrow e'_1 \), and \( e' \) is \( e'_1 + e_2 \).

  By \( H;e'_1 + e_2 \downarrow c \) and inversion,
  \( H;e'_1 \downarrow c_1 \), \( H;e_2 \downarrow c_2 \), and \( c \) is \( c_1 + c_2 \).

  By the inductive hypothesis with \( H;e_1 \rightarrow e'_1 \) and \( H;e'_1 \downarrow c_1 \),
  \( H;e_1 \downarrow c_1 \).
Proof, part 2 (cont’d):

▶ \( n = m + 1: \exists e'. H;e \rightarrow e' \) and \( H;e' \rightarrow^m c. \)

By induction (on \( H;e' \rightarrow^m c \)), we have \( H;e' \Downarrow c. \)
So this lemma suffices: If \( H;e \rightarrow e' \) and \( H;e' \Downarrow c. \), then \( H;e \Downarrow c. \)

Prove the lemma by structural induction on (the derivation of) \( H;e \rightarrow e' \):

▶ \[SADDL]\: Derivation is via \[SADDL]\]
and \( e \) is \( e_1 + e_2 \), \( H;e_1 \rightarrow e'_1 \), and \( e' \) is \( e'_1 + e_2 \).
By \( H;e'_1 + e_2 \Downarrow c \) and inversion, \( H;e'_1 \Downarrow c_1 \), \( H;e_2 \Downarrow c_2 \), and \( c \) is \( c_1 + c_2 \).
By the inductive hypothesis with \( H;e_1 \rightarrow e'_1 \) and \( H;e'_1 \Downarrow c_1 \), \( H;e_1 \Downarrow c_1 \).
So derive, by \[ADD\] with \( H;e_1 \Downarrow c_1 \) and \( H;e_2 \Downarrow c_2 \), \( H;e_1 + e_2 \Downarrow c \).
Proof, part 2 (cont’d):

$n = m + 1$: \( \exists e'. \; H; e \rightarrow e' \) and \( H; e' \rightarrow^m c. \)

By induction (on \( H; e' \rightarrow^m c. \)), we have \( H; e' \Downarrow c. \)
So this lemma suffices: If \( H; e \rightarrow e' \) and \( H; e' \Downarrow c. \), then \( H; e \Downarrow c. \).

Prove the lemma by structural induction on (the derivation of) \( H; e \rightarrow e' \):

- **[SADDL]**: Derivation is via [SADDL]
  and \( e \) is \( e_1 + e_2 \), \( H; e_1 \rightarrow e'_1 \), and \( e' \) is \( e'_1 + e_2 \).
  By \( H; e'_1 + e_2 \Downarrow c \) and inversion,
  \( H; e'_1 \Downarrow c_1 \), \( H; e_2 \Downarrow c_2 \), and \( c \) is \( c_1 + c_2 \).
  By the inductive hypothesis with \( H; e_1 \rightarrow e'_1 \) and \( H; e'_1 \Downarrow c_1 \),
  \( H; e_1 \Downarrow c_1 \).
  So derive, by [ADD] with \( H; e_1 \Downarrow c_1 \) and \( H; e_2 \Downarrow c_2 \),
  \( H; e_1 + e_2 \Downarrow c \).

- **[SADDR]**: Analogous to [SADDL].
A nice payoff

Theorem: The small-step semantics is deterministic.

- if \( H;e \to^* c_1 \) and \( H;e \to^* c_2 \), then \( c_1 = c_2 \).

Not obvious (see [SADDL] and [SADDR]), nor do I know a direct proof.

- Given \(((1 + 2) + (3 + 4)) + (5 + 6)) + (7 + 8)\)
  there are many execution sequences, all of which produce 36, but with different intermediate expressions.

(Indirect) Proof:

- Large-step evaluation is deterministic (easy proof by induction).
- Small-step and large-step are equivalent (just proved that).
- So small-step is deterministic.
- (Convince yourself that a deterministic and a nondeterministic semantics can’t be equivalent with our definition of equivalence.)
Conclusions

- Equivalence is a subtle concept.
- Proofs “seem obvious” only when the definitions are right.
- Some other language-equivalence claims:
Conclusions

- Equivalence is a subtle concept.
- Proofs “seem obvious” only when the definitions are right.
- Some other language-equivalence claims:

Replace \([\text{WHILE}]\) rule with

\[
\frac{H;e \Downarrow c \quad c \leq 0}{H;\text{while } e \ s \rightarrow H;\text{skip}} \quad \frac{H;e \Downarrow c \quad c > 0}{H;\text{while } e \ s \rightarrow H; s \ ; \ \text{while } e \ s}
\]

Theorem: Languages are equivalent.
Conclusions

- Equivalence is a subtle concept.
- Proofs “seem obvious” only when the definitions are right.
- Some other language-equivalence claims:

  Replace \([\text{WHILE}]\) rule with

  \[
  \begin{align*}
  H;e & \Downarrow c & c \leq 0 \\
  H;\text{while } e \; s & \rightarrow H;\text{skip} \\
  \end{align*}
  \]

  \[
  \begin{align*}
  H;e & \Downarrow c & c > 0 \\
  H;\text{while } e \; s & \rightarrow H;\text{s} \; ; \; \text{while } e \; s \\
  \end{align*}
  \]

  Theorem: Languages are equivalent. (True)
Conclusions

- Equivalence is a subtle concept.
- Proofs “seem obvious” only when the definitions are right.
- Some other language-equivalence claims:

Replace [WHILE] rule with

\[
\begin{align*}
H;e \downarrow c & \quad c \leq 0 \\
\text{while } e \ s & \rightarrow H;\text{skip}
\end{align*}
\]

\[
\begin{align*}
H;e \downarrow c & \quad c > 0 \\
\text{while } e \ s & \rightarrow H;s \ ; \ \text{while } e \ s
\end{align*}
\]

Theorem: Languages are equivalent. (True)

Change syntax of heap and replace [ASSGN] and [VAR] rules with

\[
\begin{align*}
H;x := e & \rightarrow H, x \mapsto e;\text{skip}
\end{align*}
\]

\[
\begin{align*}
H @ x & \rightsquigarrow e \\
H;e \downarrow c
\end{align*}
\]

Theorem: Languages are equivalent.
Conclusions

- Equivalence is a subtle concept.
- Proofs “seem obvious” only when the definitions are right.
- Some other language-equivalence claims:

Replace \texttt{[WHILE]} rule with

\[
\begin{array}{c}
H;e \Downarrow c \\
H;\text{while } e \; s \rightarrow H;\text{skip}
\end{array}
\quad
\begin{array}{c}
H;e \Downarrow c \\
H;\text{while } e \; s \rightarrow H; s \; ; \; \text{while } e \; s
\end{array}
\]

$c \leq 0$

Theorem: Languages are equivalent. \hspace{1cm} (True)

Change syntax of heap and replace \texttt{[ASSGN]} and \texttt{[VAR]} rules with

\[
\begin{array}{c}
H @ x \rightsquigarrow e \\
H; x := e \rightarrow H, x \mapsto e; \text{skip}
\end{array}
\quad
\begin{array}{c}
H;e \Downarrow c \\
H; x \Downarrow c
\end{array}
\]

Theorem: Languages are equivalent. \hspace{1cm} (False)