

## Operational Semantics

- Winskel, Ch. 2
  - Big-step operational semantics (most related)
- Slonneger and Kurtz Ch 8.4, 8.5, 8.6
  - Small-step operational semantics

CSE 6341

1

## Outline

- **Introduction**
  - Simple Imperative Language (IMP)
- Operational semantics for IMP
  - These are "big-step" semantics
- Proving equivalence using (big-step) operational semantics
- Small-step operational semantics

CSE 6341

2

## Operational vs. Axiomatic

- Operational semantics
  - Explicitly describes the effects of program constructs on program state
  - Shows not only **what** the program does, but also **how** it does it
  - Essentially describes an interpreter
- Axiomatic semantics
  - Describes properties of program state, using first-order logic
  - Concerned with constructing proofs for such properties

CSE 6341

3

## IMP: simple imperative language

- Only integer variables
- No procedures or functions
- No explicit variable declarations

CSE 6341

4

## IMP Syntax

```
<c>1 ::= skip | id := <ae> | <c>2 ; <c>3
      | if <be> then <c>2 else <c>3
      | while <be> do <c>2
<ae>1 ::= id | int | <ae>2 + <ae>3
      | <ae>2 - <ae>3 | <ae>2 * <ae>3
<be>1 ::= true | false
      | <ae>1 = <ae>2 | <ae>1 < <ae>2
      | ¬ <be>2 | <be>2 ∧ <be>3
      | <be>2 ∨ <be>3
```

CSE 6341

5

## IMP Syntax

```
<c>1 ::= skip | id := <ae> | <c>2 ; <c>3
      | if <be> then <c>2 else <c>3
      | while <be> do <c>2
<ae>1 ::= id | int | <ae>2 + <ae>3
      | <ae>2 - <ae>3 | <ae>2 * <ae>3
<be>1 ::= true | false
      | <ae>1 = <ae>2 | <ae>1 < <ae>2
      | ¬ <be>2 | <be>2 ∧ <be>3
      | <be>2 ∨ <be>3
```

What  
about  
division?

CSE 6341

6

## Outline

- Introduction
  - Simple Imperative Language (IMP)
- **Operational semantics for IMP**
  - These are “big-step” semantics
- Proving equivalence using (big-step) operational semantics
- Small-step operational semantics

CSE 6341

7

## State

- State: a function  $\sigma$  from variable names to values
- E.g., program with 2 variables  $x, y$ 
  - $\sigma(x) = 9$
  - $\sigma(y) = 5$
- For simplicity, we will only consider integer variables
  - $\sigma$ : Variables  $\rightarrow \{0, -1, +1, -2, 2, \dots\}$

CSE 6341

8

## Operational Semantics for IMP

- Proof system
- If the state is  $\sigma$  and expression  $e$  is evaluated, what is the resulting value?
  - $\langle ae, \sigma \rangle \rightarrow n$  for arithmetic expression
  - $\langle be, \sigma \rangle \rightarrow bv$  for boolean expressions
    - $ae, be$ : parse trees;  $n$ : integer;  $bv$ : boolean
- If the state is  $\sigma$  and statement  $c$  is executed to termination, what is the resulting state?
  - $\langle c, \sigma \rangle \rightarrow \sigma'$   $c$  is a parse (sub)tree

CSE 6341

9

## Evaluation of Arithmetic Expressions

An arithmetic expression is **int** or **id** or  $a_1+a_2$  or ...

$\langle \text{int}, \sigma \rangle \rightarrow n$   $n$  is the integer value of **int**

$\langle \text{id}, \sigma \rangle \rightarrow \sigma(v)$   $v$  is the variable w/ name **id**

$\frac{\langle a_1, \sigma \rangle \rightarrow n_1 \quad \langle a_2, \sigma \rangle \rightarrow n_2}{\langle a_1+a_2, \sigma \rangle \rightarrow n}$   $n$  is the sum of  $n_1$  and  $n_2$

similarly for  $a_1-a_2$  and  $a_1*a_2$

E.g. if  $\sigma(P) = 4$  and  $\sigma(Q) = 6$ ,  $\langle P+Q, \sigma \rangle \rightarrow 10$

CSE 6341

10

## Inference Rules

- We represent the semantics with inference rules
  - Zero or more premises
  - Conclusion
  - Optional condition (shown to the right): the rule applies only if the condition is true
    - e.g. “ $n$  is the sum of  $n_1$  and  $n_2$ ”
- Instances of such rules are applied for a given code fragment, in order to derive (prove) values and states

CSE 6341

11

## Level of Detail

- “ $n$  is the sum of  $n_1$  and  $n_2$ ”
  - This assumes that “sum” is a primitive notion that we will not define
- In some cases, we may decide to define it precisely
  - e.g. “sum” is not trivial for Roman numerals
  - or maybe we are describing a low-level language for some hardware device
- In this class: we will not specify how addition is done

CSE 6341

12

### Evaluation of Boolean Expressions

**true, false**,  $a_1 = a_2$ ,  $a_1 < a_2$ ,  $\neg b$ ,  $b_1 \wedge b_2$ ,  $b_1 \vee b_2$

$\langle \text{true}, \sigma \rangle \rightarrow \text{true}$	$\langle \text{false}, \sigma \rangle \rightarrow \text{false}$
$\frac{\langle a_1, \sigma \rangle \rightarrow n_1 \quad \langle a_2, \sigma \rangle \rightarrow n_2}{\langle a_1 = a_2, \sigma \rangle \rightarrow \text{true}}$	$n_1$ and $n_2$ are equal
$\frac{\langle a_1, \sigma \rangle \rightarrow n_1 \quad \langle a_2, \sigma \rangle \rightarrow n_2}{\langle a_1 = a_2, \sigma \rangle \rightarrow \text{false}}$	$n_1$ and $n_2$ are not equal

CSE 6341 13

### Evaluation of Boolean Expressions

$\frac{\langle a_1, \sigma \rangle \rightarrow n_1 \quad \langle a_2, \sigma \rangle \rightarrow n_2}{\langle a_1 < a_2, \sigma \rangle \rightarrow \text{true}}$	$n_1$ is less than $n_2$
$\frac{\langle a_1, \sigma \rangle \rightarrow n_1 \quad \langle a_2, \sigma \rangle \rightarrow n_2}{\langle a_1 < a_2, \sigma \rangle \rightarrow \text{false}}$	$n_1$ is greater than or equal to $n_2$
$\langle b, \sigma \rangle \rightarrow \text{true}$	$\langle b, \sigma \rangle \rightarrow \text{false}$
$\langle \neg b, \sigma \rangle \rightarrow \text{false}$	$\langle \neg b, \sigma \rangle \rightarrow \text{true}$

CSE 6341 14

### Evaluation of Boolean Expressions

$\frac{\langle b_1, \sigma \rangle \rightarrow t_1 \quad \langle b_2, \sigma \rangle \rightarrow t_2}{\langle b_1 \wedge b_2, \sigma \rangle \rightarrow t}$	$t$ is true iff $t_1$ and $t_2$ are true
$\frac{\langle b_1, \sigma \rangle \rightarrow t_1 \quad \langle b_2, \sigma \rangle \rightarrow t_2}{\langle b_1 \vee b_2, \sigma \rangle \rightarrow t}$	$t$ is false iff $t_1$ and $t_2$ are false

**How about short-circuit evaluation?**  
(in-class exercise)

CSE 6341 15

### Short-circuit Evaluations

- $b_1 \wedge b_2$ : if  $b_1$  evaluates to false, no need to evaluate  $b_2$
- $b_1 \vee b_2$ : if  $b_1$  evaluates to true, no need to evaluate  $b_2$
- Most programming languages do this
- How do we represent this approach as inference rules?

CSE 6341 16

### Execution of Statements

- $\sigma[m/x]$  is the same as  $\sigma$  except for  $x$ 
  - $\sigma[m/x](y) = \sigma(y)$  if  $y$  is not  $x$
  - $\sigma[m/x](x) = m$
  - Also written as  $\sigma[x \leftarrow m]$

$\langle e, \sigma \rangle \rightarrow m$	$v$ is the variable w/name <b>id</b>
$\langle \text{id} := e, \sigma \rangle \rightarrow \sigma[m/v]$	
$\frac{\langle c_1, \sigma \rangle \rightarrow \sigma' \quad \langle c_2, \sigma' \rangle \rightarrow \sigma''}{\langle c_1; c_2, \sigma \rangle \rightarrow \sigma''}$	$\langle \text{skip}, \sigma \rangle \rightarrow \sigma$

CSE 6341 17

### In-Class Exercise

$$\langle x := z + 2 ; y := x + 3, \sigma \rangle \rightarrow \sigma'$$

$$\sigma = \{ z=1 \}$$

CSE 6341 18

### Execution of Statements

$\langle b, \sigma \rangle \rightarrow \text{true}$	$\langle c_1, \sigma \rangle \rightarrow \sigma'$
$\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \rightarrow \sigma'$	
$\langle b, \sigma \rangle \rightarrow \text{false}$	$\langle c_2, \sigma \rangle \rightarrow \sigma'$
$\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \rightarrow \sigma'$	
$\langle b, \sigma \rangle \rightarrow \text{false}$	
$\langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma$	
$\langle b, \sigma \rangle \rightarrow \text{true}$	$\langle c, \sigma \rangle \rightarrow \sigma'$
$\langle \text{while } b \text{ do } c, \sigma' \rangle \rightarrow \sigma''$	
$\langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma''$	

CSE 6341

19

### In-Class Exercise

$\langle x := 1 ; \text{while } x < 2 \text{ do } x := x + 3, \sigma \rangle \rightarrow \sigma'$

CSE 6341

20

### Outline

- Introduction
  - Simple Imperative Language (IMP)
- Operational semantics for IMP
  - These are "big-step" semantics
- Proving equivalence using (big-step) operational semantics
- Small-step operational semantics

CSE 6341

21

### Equivalence

- Expressions  $x$  and  $y$  are equivalent if for any  $\sigma$  and any  $z$ ,  $\langle x, \sigma \rangle \rightarrow z$  iff  $\langle y, \sigma \rangle \rightarrow z$ 
  - e.g.  $a+b$  is equivalent to  $b-5+a+5$
- Statements  $x$  and  $y$  are equivalent if for any  $\sigma$  and  $\sigma'$ ,  $\langle x, \sigma \rangle \rightarrow \sigma'$  iff  $\langle y, \sigma \rangle \rightarrow \sigma'$ 
  - e.g. statement " $c:=a+b; d:=c;$ " is equivalent to statement " $d:=b-5+a+5; c:=d;$ "
- Essential for ensuring the correctness of compiler optimizations
  - Optimized code vs. the original code

CSE 6341

22

### Example

- Loop peeling: transform "while  $b$  do  $c$ "
- if  $b$  then ( $c$ ; while  $b$  do  $c$ ) else skip
  - Take the first iteration out of the loop
  - Common compiler optimization
- Can we prove that this transformation is semantics-preserving?
  - i.e., are these statements equivalent?

CSE 6341

23

### First Direction

If  $\langle \text{while } \dots, \sigma \rangle \rightarrow \sigma'$  is derivable, so is  $\langle \text{if } \dots, \sigma \rangle \rightarrow \sigma'$

There must be some derivation, leading to

$\langle b, \sigma \rangle \rightarrow \text{false}$  ( $\sigma$  and  $\sigma'$  are the same state), or

$\langle b, \sigma \rangle \rightarrow \text{true}$   $\langle c, \sigma \rangle \rightarrow \sigma''$   $\langle \text{while } b \text{ to } c, \sigma'' \rangle \rightarrow \sigma'$

Case 1:  $\langle b, \sigma \rangle \rightarrow \text{false}$ , and  $\langle \text{skip}, \sigma \rangle \rightarrow \sigma'$ , so

$\langle \text{if } b \text{ then } \dots \text{ else skip}, \sigma \rangle \rightarrow \sigma'$

Case 2:  $\langle b, \sigma \rangle \rightarrow \text{true}$  and  $\langle c; \text{while } \dots, \sigma \rangle \rightarrow \sigma'$ , so

$\langle \text{if } b \text{ then } c; \text{while } \dots \text{ else } \dots, \sigma \rangle \rightarrow \sigma'$

CSE 6341

24

## Second Direction

If  $\langle \text{if } \dots, \sigma \rangle \rightarrow \sigma'$  is derivable, so is  $\langle \text{while } \dots, \sigma \rangle \rightarrow \sigma'$

There must be some derivation, leading to

$\langle b, \sigma \rangle \rightarrow \text{false}$   $\langle \text{skip}, \sigma \rangle \rightarrow \sigma'$  (so  $\sigma = \sigma'$ ) or

$\langle b, \sigma \rangle \rightarrow \text{true}$   $\langle c; \text{while } \dots, \sigma \rangle \rightarrow \sigma'$

Case 1:  $\langle b, \sigma \rangle \rightarrow \text{false}$ , so  $\langle \text{while } b \text{ do } \dots, \sigma \rangle \rightarrow \sigma'$

Case 2:  $\langle b, \sigma \rangle \rightarrow \text{true}$  and  $\langle c; \text{while } \dots, \sigma \rangle \rightarrow \sigma'$ , so must have had  $\langle c, \sigma \rangle \rightarrow \sigma''$  and  $\langle \text{while } \dots, \sigma'' \rangle \rightarrow \sigma'$ , and therefore  $\langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma'$

CSE 6341

25

## Another Example

- Partial redundancy elimination
  - In its general form, an advanced compiler optimization
- if b then x:=e1 else y:=e2 fi; x:=e1
- if b then x:=e1 else y:=e2; x:=e1; fi
- Under what conditions are these two code fragments semantically equivalent?
  - Try this at home ...

CSE 6341

26

## Outline

- Introduction
  - Simple Imperative Language (IMP)
- Operational semantics for IMP
  - These are "big-step" semantics
- Proving equivalence using (big-step) operational semantics
- Small-step operational semantics

CSE 6341

27

## Big-Step vs. Small-Step Semantics

- Until now: "coarse" semantics
  - Abstracts away some details about the individual steps taken during execution
  - "Big-step" semantics: based on the productions of the underlying grammar
- Alternative "small-step" semantics: captures smaller steps in the execution
  - Sequence of code rewriting steps

CSE 6341

28

## Small-Step Semantics

- Expressions:  $\langle e, \sigma \rangle \rightarrow \langle e', \sigma \rangle$ 
  - The complete evaluation of an expression  $e$  in a state  $\sigma$  is a sequence  $\langle e, \sigma \rangle \rightarrow \langle e', \sigma \rangle \rightarrow \langle e'', \sigma \rangle \rightarrow \dots \rightarrow \langle \text{value}, \sigma \rangle \rightarrow \text{value}$
  - Example:  $\langle \text{id}, \sigma \rangle \rightarrow \langle \sigma(v), \sigma \rangle$  where  $v$  is the variable with name  $\text{id}$
- Statements:  $\langle c, \sigma \rangle \rightarrow \langle c', \sigma \rangle$ 
  - The execution of the program is a sequence  $\langle \text{program}, \sigma_{\text{initial}} \rangle \rightarrow \langle c', \sigma \rangle \rightarrow \langle c'', \sigma \rangle \rightarrow \dots \rightarrow \sigma_{\text{final}}$

CSE 6341

29

## Small-Step Evaluation of Expressions

$\langle e, \sigma \rangle \rightarrow \langle e', \sigma \rangle$	$\langle e', \sigma \rangle \rightarrow \langle e'', \sigma \rangle$	transitivity
$\langle e, \sigma \rangle \rightarrow \langle e'', \sigma \rangle$		

$\langle \text{id}, \sigma \rangle \rightarrow \langle \sigma(v), \sigma \rangle$  evaluation of a variable

$\langle n, \sigma \rangle \rightarrow n$  evaluation of an integer constant

- Binary expressions: the left argument is evaluated first, the right arguments is evaluated next; big-step semantics does not capture this

CSE 6341

30

### Example: Small-Step Addition

$\langle a_1, \sigma \rangle \rightarrow \langle a_1', \sigma \rangle$	$\langle a_2, \sigma \rangle \rightarrow \langle a_2', \sigma \rangle$
$\langle a_1 + a_2, \sigma \rangle \rightarrow \langle a_1' + a_2, \sigma \rangle$	$\langle n + a_2, \sigma \rangle \rightarrow \langle n + a_2', \sigma \rangle$

$\langle n + m, \sigma \rangle \rightarrow \langle p, \sigma \rangle$  where the sum of m and n is p

- If one step in the evaluation of  $a_1$  leads to  $a_1'$ , then one step in the evaluation of  $a_1 + a_2$  leads to  $a_1' + a_2$  (evaluate  $a_1$  first)
- After  $a_1$  is evaluated, evaluate  $a_2$
- Does the order matter for this particular simple language?

CSE 6341

31

### Example

Evaluate  $a+b+c$ , with  $\sigma(a)=1$ ,  $\sigma(b)=2$ ,  $\sigma(c)=4$

$\langle a, \sigma \rangle \rightarrow \langle 1, \sigma \rangle$  therefore  $\langle a+b, \sigma \rangle \rightarrow \langle 1+b, \sigma \rangle$

$\langle b, \sigma \rangle \rightarrow \langle 2, \sigma \rangle$  therefore  $\langle 1+b, \sigma \rangle \rightarrow \langle 1+2, \sigma \rangle$

By the axiom:  $\langle 1+2, \sigma \rangle \rightarrow \langle 3, \sigma \rangle$

transitivity:  $\langle a+b, \sigma \rangle \rightarrow \langle 3, \sigma \rangle$

therefore  $\langle a+b+c, \sigma \rangle \rightarrow \langle 3+c, \sigma \rangle$

$\langle c, \sigma \rangle \rightarrow \langle 4, \sigma \rangle$  therefore  $\langle 3+c, \sigma \rangle \rightarrow \langle 3+4, \sigma \rangle$

By the axiom:  $\langle 3+4, \sigma \rangle \rightarrow \langle 7, \sigma \rangle$

transitivity:  $\langle a+b+c, \sigma \rangle \rightarrow \langle 7, \sigma \rangle \rightarrow 7$

CSE 6341

32

### Small-Step Execution of Statements

$\langle e, \sigma \rangle \rightarrow \langle e', \sigma \rangle$	$\langle m, \sigma \rangle \rightarrow m$
$\langle id := e, \sigma \rangle \rightarrow \langle id := e', \sigma \rangle$	$\langle id := m, \sigma \rangle \rightarrow \sigma[m/v]$

$\langle c_1, \sigma \rangle \rightarrow \langle c_1', \sigma' \rangle$	$\langle c_1, \sigma \rangle \rightarrow \sigma'$
$\langle c_1; c_2, \sigma \rangle \rightarrow \langle c_1'; c_2, \sigma' \rangle$	$\langle c_1; c_2, \sigma \rangle \rightarrow \langle c_2, \sigma' \rangle$

$\langle c, \sigma \rangle \rightarrow \langle c', \sigma' \rangle$	$\langle c', \sigma' \rangle \rightarrow \langle c'', \sigma'' \rangle$	transitivity
$\langle c, \sigma \rangle \rightarrow \langle c'', \sigma'' \rangle$		

CSE 6341

33

### Example

$\langle x:=5; y:=1, \sigma \rangle \rightarrow ?$

$\langle 5, \sigma \rangle \rightarrow 5$  and  $\langle x:=5, \sigma \rangle \rightarrow \sigma[5/x]$

therefore

$\langle x:=5; y:=1, \sigma \rangle \rightarrow \langle y:=1, \sigma[5/x] \rangle$

also  $\langle y:=1, \sigma[5/x] \rangle \rightarrow \sigma[5/x][1/y]$

through transitivity

$\langle x:=5; y:=1, \sigma \rangle \rightarrow \sigma[5/x][1/y]$

CSE 6341

34

### Outline

- Introduction
  - Simple Imperative Language (IMP)
- Operational semantics for IMP
  - These are "big-step" semantics
- Proving equivalence using (big-step) operational semantics
- Small-step operational semantics

CSE 6341

35