

## Proof Techniques Based on Operational Semantics

### Lecture 3 CS263

CS 263

1

#### Induction

- Probably the single most important technique for the study of formal semantics of programming languages
- Of several kinds
  - mathematical induction (the simplest)
  - well-founded induction (the most general)
  - structural induction (the most widely used in PL)

CS 263

3

#### Mathematical Induction. Notes.

- The inductive case looks similar to the overall goal  
 $\forall n \in \mathbb{N}. P(n) \Rightarrow P(n+1)$  vs.  $\forall n \in \mathbb{N}. P(n)$ 
  - but it is simpler because of the assumption that  $P(n)$  holds
- Why does mathematical induction work?
  - The key property of  $\mathbb{N}$  is that there are no infinite descending chains of naturals. It has to stop somewhere.
  - For each  $n$ ,  $P(n)$  can be obtained from the base case and uses of the inductive case

CS 263

5

#### Plan

- We'll study various flavors of induction
  - mathematical induction
  - well-founded induction
  - structural induction

CS 263

2

#### Mathematical Induction

- Goal: prove that  $\forall n \in \mathbb{N}. P(n)$
- Strategy: (2 steps)
  1. Base case: prove that  $P(0)$
  2. Inductive case:
    - pick an arbitrary  $n \in \mathbb{N}$
    - assume that  $P(n)$  holds
    - prove that  $P(n+1)$
    - or, formally prove that  $\forall n \in \mathbb{N}. P(n) \Rightarrow P(n+1)$

CS 263

4

#### Example of Mathematical Induction

- Recall the evaluation rules for IMP commands
- Prove that if  $\sigma(x) \leq 6$  then  
 $\langle \text{while } x \leq 5 \text{ do } x := x + 1, \sigma \rangle \Downarrow \sigma[x := 6]$
- Reformulate the claim:
  - Let  $W = \text{while } x \leq 5 \text{ do } x := x + 1$
  - Let  $\sigma_i = \sigma[x := 6 - i]$
  - Claim:  $\forall i \in \mathbb{N}. \langle W, \sigma_i \rangle \Downarrow \sigma_0$
- Now the claim looks provable by mathematical induction on  $i$

CS 263

6

### Example of Mathematical Induction (Base Case)

- Base case:  $i = 0$  or  $\langle W, \sigma_0 \rangle \Downarrow \sigma_0$ 
  - To prove an evaluation judgment, construct a derivation tree:

$$\frac{\frac{\sigma_0(x) = 6}{\langle x, \sigma_0 \rangle \Downarrow 6} \quad \langle 6 \leq 5, \sigma_0 \rangle \Downarrow \text{false}}{\langle x \leq 5, \sigma_0 \rangle \Downarrow \text{false}} \quad \frac{}{\langle \text{while } x \leq 5 \text{ do } x := x + 1, \sigma_0 \rangle \Downarrow \sigma_0}$$

- This completes the base case

CS 263

7

### Example of Mathematical Induction (Inductive Case)

- Must prove  $\forall i \in \mathbb{N}. \langle W, \sigma_i \rangle \Downarrow \sigma_0 \Rightarrow \langle W, \sigma_{i+1} \rangle \Downarrow \sigma_0$
- The beginning of the proof is straightforward
  - Pick an arbitrary  $i \in \mathbb{N}$
  - Assume that  $\langle W, \sigma_i \rangle \Downarrow \sigma_0$
  - Now prove that  $\langle W, \sigma_{i+1} \rangle \Downarrow \sigma_0$
  - Must construct a derivation tree:

$$\frac{\frac{\langle x, \sigma_{i+1} \rangle \Downarrow 5 - i \quad 5 - i \leq 5}{\langle x \leq 5, \sigma_{i+1} \rangle \Downarrow \text{true}} \quad \frac{\frac{\langle x + 1, \sigma_{i+1} \rangle \Downarrow 6 - i}{\langle x := x + 1, \sigma_{i+1} \rangle \Downarrow \sigma_i} \quad \langle W, \sigma_i \rangle \Downarrow \sigma_0}{\langle x := x + 1, W, \sigma_{i+1} \rangle \Downarrow \sigma_0}}{\langle \text{while } x \leq 5 \text{ do } x := x + 1, \sigma_{i+1} \rangle \Downarrow \sigma_0}$$

CS 263

8

### Discussion

- A proof is more powerful than running the code and observing the result. Why?
- The proof relied on a loop invariant
  - $x \leq 6$  in all iterations
- ... and a loop variant
  - $6 - x$  is positive and decreasing
- Picking the loop invariant and variant is typically the hardest part of a proof

CS 263

9

### Discussion

- We proved termination and correctness. This is called total correctness
- Mathematical induction is good when we prove properties of natural numbers
  - In PL analysis we most often prove properties of expressions, commands, programs, input data, etc.
  - We need a more powerful induction principle

CS 263

10

### Well-Founded Induction

- A relation  $\prec \subseteq A \times A$  is well-founded if there are no infinite descending chains in  $A$ 
  - Example:  $\prec_1 = \{ (x, x+1) \mid x \in \mathbb{N} \}$ 
    - the predecessor relation
  - Example:  $\prec = \{ (x, y) \mid x, y \in \mathbb{N} \text{ and } x < y \}$
- Well-founded induction:
  - To prove  $\forall x \in A. P(x)$  it is enough to prove  $\forall x \in A. [\forall y \prec x \Rightarrow P(y)] \Rightarrow P(x)$
- If  $\prec$  is  $\prec_1$  then we obtain mathematical induction as a special case

CS 263

11

### Well-Founded Induction. Notes.

- Why does well-founded induction work?
  - Assume we proved that  $\forall x \in A. (\forall y \prec x \Rightarrow P(y)) \Rightarrow P(x)$
  - Define  $D_n$  to be the subset of all elements of  $A$  whose longest descending chain is of length at most  $n$ 

$$\forall x \in A. \exists n \geq 1. x \in D_n \text{ (no infinite descending chains)}$$
  - Show by mathematical induction that  $\forall n \geq 1. (\forall x \in D_n. P(x))$
  - Base case: Show  $\forall x \in D_1. P(x)$ 
    - It is vacuously true that if  $x \in D_1$  then  $\forall y \prec x \Rightarrow P(y)$
  - Inductive case:
    - Assume  $\forall x \in D_n. P(x)$
    - Show that  $\forall x \in D_{n+1}. P(x)$
    - But if  $y \prec x$  then  $y \in D_n$
    - Thus  $\forall y \prec x \Rightarrow P(y)$
  - See Winskel (Chapter 3) for another proof (the correct one!)

CS 263

12

### Well-Founded Induction. Examples.

- Consider  $< \subseteq \mathbb{N} \times \mathbb{N}$  with  $x < y$  iff  $x + 2 = y$ 
  - $\forall x \in \mathbb{N}. (\forall y < x \Rightarrow P(y)) \Rightarrow P(x)$  is equivalent to
  - $P(0) \wedge P(1) \wedge \forall n \in \mathbb{N}. (P(n) \Rightarrow P(n+2))$
- Consider  $< \subseteq \mathbb{Z} \times \mathbb{Z}$  with  $x < y$  iff
  - $(y < 0 \text{ and } y = x - 1) \text{ or } (y > 0 \text{ and } y = x + 1)$
  - $P(0) \wedge \forall x \leq 0. P(x) \Rightarrow P(x - 1) \wedge \forall x \geq 0. P(x) \Rightarrow P(x + 1)$
- Consider  $< \subseteq (\mathbb{N} \times \mathbb{N}) \times (\mathbb{N} \times \mathbb{N})$  and  $(x_1, y_1) < (x_2, y_2)$  iff
  - $x_2 = x_1 + 1 \vee (x_1 = x_2 \wedge y_2 = y_1 + 1)$
  - This leads to the induction principle
  - $P(0,0) \wedge \forall x,y,y'. (P(x,y) \Rightarrow P(x+1,y') \wedge P(x,y+1))$
  - This is sometimes called lexicographic induction

CS 263

13

### Structural Induction

- Recall  $e ::= n \mid e_1 + e_2 \mid e_1 * e_2 \mid x$
- Define  $< \subseteq \text{Aexp} * \text{Aexp}$  such that
  - $e_1 < e_1 + e_2$
  - $e_2 < e_1 + e_2$
  - $e_1 < e_1 * e_2$
  - $e_2 < e_1 * e_2$
  - and no other elements of  $\text{Aexp} * \text{Aexp}$  are related by  $<$
- To prove  $\forall e \in \text{Aexp}. P(e)$ 
  1. Prove  $\forall n \in \mathbb{Z}. P(n)$
  2. Prove  $\forall x \in L. P(x)$
  3. Prove  $\forall e_1, e_2 \in \text{Aexp}. P(e_1) \wedge P(e_2) \Rightarrow P(e_1 + e_2)$
  4. Prove  $\forall e_1, e_2 \in \text{Aexp}. P(e_1) \wedge P(e_2) \Rightarrow P(e_1 * e_2)$

CS 263

14

### Structural Induction. Notes.

- Called structural induction because the proof is guided by the structure of the expression
- As many cases as there are expression forms
  - Atomic expressions (with no subexpressions) are all base cases
  - Composite expressions are the inductive case
- This is the most useful form of induction in PL

CS 263

15

### Example of Induction on Structure of Expressions

- Let
  - $L(e)$  be the number of literals and variable occurrences in  $e$
  - $O(e)$  be the number of operators in  $e$
- Prove that  $\forall e \in \text{Aexp}. L(e) = O(e) + 1$
- By induction on the structure of  $e$ 
  - Case  $e = n$ .  $L(e) = 1$  and  $O(e) = 0$
  - Case  $e = x$ .  $L(e) = 1$  and  $O(e) = 0$
  - Case  $e = e_1 + e_2$ .
    - $L(e) = L(e_1) + L(e_2)$  and  $O(e) = O(e_1) + O(e_2) + 1$
    - By induction hypothesis  $L(e_1) = O(e_1) + 1$  and  $L(e_2) = O(e_2) + 1$
    - Thus  $L(e) = O(e) + 1$
  - Case  $e = e_1 * e_2$ . Same as the case for +

CS 263

16

### Other Proofs by Structural Induction on Expressions

- Most proofs for Aexp sublanguage of IMP
- Small-step and natural semantics
  - $\forall e \in \text{Exp}. \forall n \in \mathbb{N}. e \rightarrow^* n \Leftrightarrow e \Downarrow n$
- Structural induction on expressions works here because all of the semantics are syntax directed

CS 263

17

### Another Proof

- Prove that IMP is deterministic
  - $\forall e \in \text{Aexp}. \forall \sigma \in \Sigma. \forall n, n' \in \mathbb{N}. \langle e, \sigma \rangle \Downarrow n \wedge \langle e, \sigma \rangle \Downarrow n' \Rightarrow n = n'$
  - $\forall b \in \text{Bexp}. \forall \sigma \in \Sigma. \forall t, t' \in \mathbb{B}. \langle b, \sigma \rangle \Downarrow t \wedge \langle b, \sigma \rangle \Downarrow t' \Rightarrow t = t'$
  - $\forall c \in \text{Comm}. \forall \sigma, \sigma' \in \Sigma. \langle c, \sigma \rangle \Downarrow \sigma' \wedge \langle c, \sigma \rangle \Downarrow \sigma'' \Rightarrow \sigma' = \sigma''$
- No immediate way to use mathematical induction
- For commands we cannot use induction on the structure of the command
  - Consider the rule for while. Its evaluation does not depend only on the evaluation of its strict subexpressions

$$\frac{\langle b, \sigma \rangle \Downarrow \text{true} \quad \langle c, \sigma \rangle \Downarrow \sigma' \quad \langle \text{while } b \text{ do } c, \sigma \rangle \Downarrow \sigma''}{\langle \text{while } b \text{ do } c, \sigma \rangle \Downarrow \sigma''}$$

CS 263

18

### Induction on the Structure of Derivations

- Key idea: The hypothesis does not assume just a  $c \in \text{Comm}$  but the existence of a derivation of  $\langle c, \sigma \rangle \Downarrow \sigma'$
- Derivation trees are also defined inductively, just like expression trees
- A derivation is built of subderivations:

$$\frac{\frac{\langle x, \sigma_{i+1} \rangle \Downarrow 5 - i \quad 5 - i \leq 5}{\langle x \leq 5, \sigma_{i+1} \rangle \Downarrow \text{true}} \quad \frac{\frac{\langle x + 1, \sigma_{i+1} \rangle \Downarrow 6 - i}{\langle x := x + 1, \sigma_{i+1} \rangle \Downarrow \sigma_i} \quad \langle W, \sigma_i \rangle \Downarrow \sigma_0}{\langle x := x + 1; W, \sigma_{i+1} \rangle \Downarrow \sigma_0}}{\langle \text{while } x \leq 5 \text{ do } x := x + 1, \sigma_{i+1} \rangle \Downarrow \sigma_0}$$

- Adapt the structural induction principle to work on the structure of derivations

CS 263

19

### Induction on Derivations

- To prove that for all derivations  $D$  of a judgment, property  $P$  holds

1. For each derivation rule of the form

$$\frac{H_1 \dots H_n}{C}$$

2. Assume that  $P$  holds for derivations of  $H_i$  ( $i = 1, \dots, n$ )
3. Prove the the property holds for the derivation obtained from the derivations of  $H_i$  using the given rule

CS 263

20

### Example of Induction on Derivations (I)

- Prove that evaluation of commands is deterministic:  
 $\langle c, \sigma \rangle \Downarrow \sigma' \Rightarrow \forall \sigma'' \in \Sigma. \langle c, \sigma \rangle \Downarrow \sigma'' \Rightarrow \sigma' = \sigma''$
- Pick arbitrary  $c, \sigma, \sigma'$  and  $D :: \langle c, \sigma \rangle \Downarrow \sigma'$
- To prove:  $\forall \sigma'' \in \Sigma. \langle c, \sigma \rangle \Downarrow \sigma'' \Rightarrow \sigma' = \sigma''$
- Proof by induction on the structure of the derivation  $D$
- Case: last rule used in  $D$  was the one for skip

$$D :: \frac{}{\langle \text{skip}, \sigma \rangle \Downarrow \sigma}$$

- This means that  $c = \text{skip}$ , and  $\sigma' = \sigma$
- By inversion  $\langle c, \sigma \rangle \Downarrow \sigma''$  uses the rule for skip. Thus  $\sigma'' = \sigma$
- This is a base case in the induction

CS 263

21

### Example of Induction on Derivations (II)

- Case: the last rule used in  $D$  was the one for sequencing

$$D :: \frac{D_1 :: \langle c_1, \sigma \rangle \Downarrow \sigma_1 \quad D_2 :: \langle c_2, \sigma_1 \rangle \Downarrow \sigma'}{\langle c_1; c_2, \sigma \rangle \Downarrow \sigma'}$$

- Pick arbitrary  $\sigma''$  such that  $D'' :: \langle c_1; c_2, \sigma \rangle \Downarrow \sigma''$ .
  - by inversion  $D''$  uses the rule for sequencing
  - and has subderivations  $D''_1 :: \langle c_1, \sigma \rangle \Downarrow \sigma''_1$  and  $D''_2 :: \langle c_2, \sigma''_1 \rangle \Downarrow \sigma''$
- By induction hypothesis on  $D_1$  (with  $D''_1$ ):  $\sigma_1 = \sigma''_1$ 
  - Now  $D''_2 :: \langle c_2, \sigma_1 \rangle \Downarrow \sigma''$
- By induction hypothesis on  $D_2$  (with  $D''_2$ ):  $\sigma' = \sigma''$
- This is a simple inductive case

CS 263

22

### Example of Induction on Derivations (III)

- Case: the last rule used in  $D$  was the one for while true

$$D :: \frac{D_1 :: \langle b, \sigma \rangle \Downarrow \text{true} \quad D_2 :: \langle c, \sigma \rangle \Downarrow \sigma_1 \quad D_3 :: \langle \text{while } b \text{ do } c, \sigma_1 \rangle \Downarrow \sigma'}{\langle \text{while } b \text{ do } c, \sigma \rangle \Downarrow \sigma'}$$

- Pick arbitrary  $\sigma''$  such that  $D'' :: \langle \text{while } b \text{ do } c, \sigma \rangle \Downarrow \sigma''$ 
  - by inversion and determinism of boolean expressions,  $D''$  also uses the rule for while true
  - and has subderivations  $D''_2 :: \langle c, \sigma \rangle \Downarrow \sigma''_1$  and  $D''_3 :: \langle W, \sigma''_1 \rangle \Downarrow \sigma''$
- By induction hypothesis on  $D_2$  (with  $D''_2$ ):  $\sigma_1 = \sigma''_1$ 
  - Now  $D''_3 :: \langle \text{while } b \text{ do } c, \sigma_1 \rangle \Downarrow \sigma''$
- By induction hypothesis on  $D_3$  (with  $D''_3$ ):  $\sigma' = \sigma''$

CS 263

23

### Induction on Derivation. Notes.

- If we have to prove  $\forall x \in A. P(x) \Rightarrow Q(x)$ 
  - with  $A$  inductively defined and  $P(x)$  rule-defined
  - we pick arbitrary  $x \in A$  and  $D :: P(x)$
  - we could do induction on both facts
    - $x \in A$  leads to induction on the structure of  $x$
    - $D :: P(x)$  leads to induction on the structure of  $D$
  - Generally, the induction on the structure of the derivation is more powerful and a safer bet
- In many situations there are several choices for induction
  - choosing the right one is a trial-and-error process
  - a bit of practice can help a lot

CS 263

24

## Equivalence

- Two expressions (commands) are equivalent if they yield the same result from all states

$$e_1 = e_2 \text{ iff } \forall \sigma \in \Sigma. \forall n \in \mathbb{N}. \langle e_1, \sigma \rangle \Downarrow n \text{ iff } \langle e_2, \sigma \rangle \Downarrow n$$

and for commands

$$c_1 = c_2 \text{ iff } \forall \sigma, \sigma' \in \Sigma. \langle c_1, \sigma \rangle \Downarrow \sigma' \text{ iff } \langle c_2, \sigma \rangle \Downarrow \sigma'$$

CS 263

25

## Notes on Equivalence

- Equivalence is like validity
  - must hold in all states
  - $2 = 1 + 1$  is like " $2 = 1 + 1$  is valid"
  - $2 = 1 + x$  might or might not hold.
    - So, 2 is not equivalent to  $1 + x$
- Equivalence (for IMP) is undecidable
  - If it were we could solve the halting problem for IMP. How?
- Equivalence justifies code transformations
  - compiler optimizations
  - code instrumentation
  - abstract modeling
- Semantics is the basis for proving equivalence.

CS 263

26

## Equivalence Examples

- skip;  $c = c$
- $(x := e_1; x := e_2) = x := e_2$ . When is this true?
- while b do c = if b then c; while b do c else skip
- If  $e_1 = e_2$  then  $x := e_1 = x := e_2$
- while true do skip = while true do  $x := x + 1$
- If c is
  - while  $x \neq y$  do
  - if  $x \geq y$  then  $x := x - y$  else  $y := y - x$then  $(x := 221; y := 527; c) = (x := 17; y := 17)$

CS 263

27

## Proving An Equivalence

- Prove that "skip;  $c = c$ " for all c
- Assume that  $D :: \langle \text{skip}; c, \sigma \rangle \Downarrow \sigma'$
- By inversion (twice) we have that

$$D :: \frac{\langle \text{skip}, \sigma \rangle \Downarrow \sigma \quad D_1 :: \langle c, \sigma \rangle \Downarrow \sigma'}{\langle \text{skip}; c, \sigma \rangle \Downarrow \sigma'}$$

- Thus, we have  $D_1 :: \langle c, \sigma \rangle \Downarrow \sigma'$
- The other direction is similar

CS 263

28

## Proving An Inequivalence

- Prove that  $x := y \neq x := z$  when  $y \neq z$
- It suffices to exhibit a  $\sigma$  in which the two commands yield different results
- Let  $\sigma(y) = 0$  and  $\sigma(z) = 1$
- Then  $\langle x := y, \sigma \rangle \Downarrow \sigma[x := 0]$
- and  $\langle x := z, \sigma \rangle \Downarrow \sigma[x := 1]$

CS 263

29

## Summary of Operational Semantics

- Precise specification of dynamic semantics
  - order of evaluation (or that it doesn't matter)
  - error conditions (sometimes implicitly, by rule applicability)
- Simple and abstract (vs. implementations)
  - no low-level details such as stack and memory management, data layout, etc.
- Often not compositional (see while)
- Basis for some proofs about the language
- Basis for some reasoning about programs
- Point of reference for other semantics

CS 263

30