

**CMSC 631 — Program Analysis and Understanding  
Fall 2006**

Operational Semantics

## Formal Program Semantics

- Data flow analysis defines a set of transfer functions that aim to prove facts about a program
- How do we know that the facts we prove are correct?
- First, we need to understand a program's *semantics*; that is, *what it means*

CMSC 631

2

## Today's Lecture

- Semantics of simple arithmetic expressions
  - "small-step" and "big-step"
  - Properties: determinacy and normal forms
  - Proposition of equivalence (without proof)
- Semantics of a more realistic language, with control flow, like what we used for data flow analysis
- Statement of (partial) correctness for dataflow analysis
  - Proof sketch

CMSC 631

3

## Arithmetic Expressions

- $x, y, z \in \text{Var}$
- $i, j, k \in \text{Int}$
- $e \in \text{Exp} ::= x \mid i \mid e_1 + e_2 \mid e_1 * e_2$
- Examples
  - $(x + 1) * y$
  - $(1 + 3) * (x * y)$
  - Etc.

CMSC 631

4

## Semantics of Arithmetic Expressions

- Defined as *transitions* between *abstract machine states*  $(e, s)$ 
  - $e$  is the expression to evaluate
  - $s$  is the program memory
- Formally:
  - $\text{Store} = \text{Var} \rightarrow \text{Int}$ 
    - (a function from variables to integers)
  - $\text{MachineState} = \text{Exp} \times \text{Store}$
  - $\text{Transition} = \text{MachineState} \times \text{MachineState}$ 
    - We write  $MS \rightarrow MS'$  to say that  $(MS, MS') \in \text{relation} \rightarrow$

CMSC 631

5

## Examples

- $(3+4, s) \rightarrow (7, s)$
- $(2*6, s) \rightarrow (12, s)$
- $(x, s) \rightarrow (s(x), s)$
- The number of possible transitions is infinite (Why?)
  - Need a way to specify all possible members of the transition relation
- Solution: inference rules

CMSC 631

6

## Transition Relation: Computation

$$(var) \frac{s(x) = k}{(x,s) \rightarrow (k,s)} \quad \leftarrow \quad \boxed{\text{if } s(x) = k \text{ then } (x,s) \rightarrow (k,s) \text{ is a legal transition}}$$

$$(plus) \frac{}{(i+j,s) \rightarrow (k,s)} \quad (\text{where } k \text{ is the sum of } i \text{ and } j)$$

$$(times) \frac{}{(i*j,s) \rightarrow (k,s)} \quad (\text{where } k \text{ is the product of } i \text{ and } j)$$

- These rules are sufficient to define the transitions we say on the previous slide

- But what about compound expressions, like  $(i+(j+k),s)$ ?

CMSC 631

7

## Transition Relation: Congruence

$$(left\ plus) \frac{(e_1,s) \rightarrow (e_1',s)}{(e_1+e_2,s) \rightarrow (e_1'+e_2,s)}$$

$$(right\ plus) \frac{(e_2,s) \rightarrow (e_2',s)}{(i+e_2,s) \rightarrow (i+e_2',s)}$$

$$(left\ times) \frac{(e_1,s) \rightarrow (e_1',s)}{(e_1*e_2,s) \rightarrow (e_1'*e_2,s)}$$

$$(right\ times) \frac{(e_2,s) \rightarrow (e_2',s)}{(i*e_2,s) \rightarrow (i*e_2',s)}$$

CMSC 631

8

## Transitions as Proofs

- Rules define proofs that a given machine state transitions to another machine state

- Example

$$\begin{array}{c} (var) \frac{s(x) = 21}{(x,s) \rightarrow (21,s)} \\ (right\ times) \frac{}{(2*x,s) \rightarrow (2*21,s)} \\ (right\ plus) \frac{}{(12+2*x,s) \rightarrow (12+2*21,s)} \end{array}$$

- But how can we discover such a proof?

- How can we determine  $(MS \rightarrow MS')$  given  $MS$ ?

CMSC 631

9

## Proof by Goal-directed Search

- Compare the structure of a machine state  $(e,s)$  to the conclusion of each transition rule

- Then apply the rule that matches

- Example

- $(12+(2*x),s) \rightarrow ???$

- Has form  $(i+e_2,s)$

–Where  $i = 12$  and  $e_2 = 2*x$ .

- Thus matches the rule *(right-plus)*.

- Then we apply the same approach to proving the premises.

CMSC 631

10

## Evaluation

- We are interested in the “final result” of a given arithmetic expression. We call this result a *value*.

- The process of reducing an expression to a value is thus called *evaluation*.

- Define

- $v \in \text{Values} = \text{Int}$

- $\rightarrow^*$  as the reflexive, transitive closure of  $\rightarrow$

- Can recursively apply goal-directed search for each transition

CMSC 631

11

## Example Evaluation

$$\begin{array}{c} (var) \frac{s(x) = 21}{(x,s) \rightarrow (21,s)} \\ (right\ times) \frac{}{(2*x,s) \rightarrow (2*21,s)} \\ (right\ plus) \frac{}{(12+2*x,s) \rightarrow (12+2*21,s)} \end{array}$$

and  $(right\ plus) \frac{(times) \frac{}{(2*21,s) \rightarrow (42,s)}}{(12+2*21,s) \rightarrow (12+42,s)}$

implies  $(12+2*x,s) \rightarrow^* (12+42,s)$

CMSC 631

12

## Properties of $\rightarrow$

• **Determinacy:** Given a machine state  $(e,s)$ , either  $e$  is an integer, or else there exists at most one  $e'$  such that  $(e,s) \rightarrow (e',s)$ .

- Proof: by induction on the structure of  $e$ .

• **Confluence:** if  $(e,s) \rightarrow^* (e_1,s)$  and  $(e,s) \rightarrow^* (e_2,s)$  then there exists some  $e_3$  such that  $(e_1,s) \rightarrow^* (e_3,s)$  and  $(e_2,s) \rightarrow^* (e_3,s)$ .

- Proof: follows from Determinacy.

CMSC 631

13

## Alternative Semantics

$$\text{(left plus)} \frac{(e_1,s) \rightarrow (e_1',s)}{(e_1+e_2,s) \rightarrow (e_1'+e_2,s)}$$

$$\text{(right plus')} \frac{(e_2,s) \rightarrow (e_2',s)}{(e_1+e_2,s) \rightarrow (e_1+e_2',s)}$$

$$\text{(left times)} \frac{(e_1,s) \rightarrow (e_1',s)}{(e_1 * e_2,s) \rightarrow (e_1' * e_2,s)}$$

$$\text{(right times')} \frac{(e_2,s) \rightarrow (e_2',s)}{(e_1 * e_2,s) \rightarrow (e_1 * e_2',s)}$$

CMSC 631

14

## Determinacy Lost

Given  $(y+2*x,s)$ , we now have either

$$\begin{array}{l} \text{(var)} \frac{s(x) = 21}{(x,s) \rightarrow (21,s)} \\ \text{(right times)} \frac{(x,s) \rightarrow (21,s)}{(2*x,s) \rightarrow (2*21,s)} \\ \text{(right plus)} \frac{(2*x,s) \rightarrow (2*21,s)}{(y+2*x,s) \rightarrow (y+2*21,s)} \end{array}$$

or

$$\begin{array}{l} \text{(var)} \frac{s(y) = 12}{(y,s) \rightarrow (12,s)} \\ \text{(left plus)} \frac{(y,s) \rightarrow (12,s)}{(y+2*x,s) \rightarrow (12+2*x,s)} \end{array}$$

CMSC 631

15

## Confluence Retained

• We can still prove **Confluence** for the new semantics.

- The proof is harder. We prove a simpler lemma first, that shows we have confluence for  $\rightarrow$ , then we repeatedly apply this lemma for  $\rightarrow^*$

• **Normal Forms:** if  $(e,s) \rightarrow^* (i,s)$  and  $(e,s) \rightarrow^* (j,s)$  then  $i = j$ .

- Follows from Confluence.
- Thus there is a unique meaning for each expression  $e$ , which is its normal form.

CMSC 631

16

## Big-Step Semantics

• If we are really concerned only with normal forms, we might want to dispense with intermediate states.

• Define a relation  $(e,s) \Downarrow i$

- Read: expression  $e$  evaluates to  $i$  in state  $s$

CMSC 631

17

## Big-Step Inference Rules

$$\text{(var)} \frac{s(x) = k}{(x,s) \Downarrow k} \quad \text{(id)} \frac{}{(i,s) \Downarrow i}$$

$$\text{(plus)} \frac{(e_1,s) \Downarrow i \quad (e_2,s) \Downarrow j}{(e_1+e_2,s) \Downarrow k} \quad \text{(where } k \text{ is the sum of } i \text{ and } j)$$

$$\text{(times)} \frac{(e_1,s) \Downarrow i \quad (e_2,s) \Downarrow j}{(e_1 * e_2,s) \Downarrow (k,s)} \quad \text{(where } k \text{ is the product of } i \text{ and } j)$$

CMSC 631

18

## Equivalence of the Semantics

- Define  $eval((e,s),i)$  iff  $(e,s) \rightarrow^* i$ .
- **Equivalence:**  $eval((e,s),i)$  iff  $(e,s) \Downarrow i$ .
  - Proof: Must show  $(e,s) \Downarrow i$  implies  $eval((e,s),i)$  and  $eval((e,s),i)$  implies  $(e,s) \Downarrow i$ .
  - The first is by induction on the structure of the derivation  $(e,s) \Downarrow i$ .
  - The second is comes as a corollary of the lemma: if  $(e,s) \rightarrow^n (e',s)$  and  $(e',s) \Downarrow i$ , then  $(e,s) \Downarrow i$ 
    - where  $(e,s) \rightarrow^n (e',s)$  means
      - $(e,s) \rightarrow (e_1,s) \rightarrow \dots \rightarrow (e_n,s) \rightarrow (e',s)$
      - The proof is by induction on the length  $n$

CMSC 631

19

## A Language of Commands

- Extend the language to include commands  $c$
- $c ::=$
- skip
- $x := e$
- $\text{if}_{\text{not } 0} e \text{ then } c_1 \text{ else } c_2$
- $\text{while}_{\text{not } 0} e \text{ do } c$
- $c_1; c_2$

CMSC 631

20

## Semantics of Commands

- Two small-step relations; normal form:
  - $(c,s) \rightarrow s'$
  - $(c,s) \rightarrow (c',s')$
- Reflexive, transitive closure of both relations:
  - $(c,s) \rightarrow^* s'$
- One big-step relation
  - $(c,s) \Downarrow s'$

CMSC 631

21

## Small-step Computation Rules

$$\text{(assign)} \frac{}{(x := i, s) \rightarrow s[i \backslash x]} \quad \text{(skip)} \frac{}{(\text{skip}, s) \rightarrow s}$$

$$\text{(if0)} \frac{}{(\text{if}_{\text{not } 0} 0 \text{ then } c_1 \text{ else } c_2, s) \rightarrow (c_2, s)}$$

$$\text{(ifn)} \frac{}{(\text{if}_{\text{not } 0} i \text{ then } c_1 \text{ else } c_2, s) \rightarrow (c_1, s)} \quad (\text{where } i \neq 0)$$

CMSC 631

22

## Semantics of While

$$\text{(while)} \frac{}{(\text{if}_{\text{not } 0} e \text{ then } (c; \text{while}_{\text{not } 0} e \text{ do } c) \text{ else skip}, s) \rightarrow (\text{while}_{\text{not } 0} e \text{ do } c, s)}$$

- Semantics by “unrolling” the loop once and evaluating that
- Different from other rules: not in terms of sub-components

CMSC 631

23

## Small-step Congruence Rules

$$\text{(cong1)} \frac{(e,s) \rightarrow (e',s)}{(x := e, s) \rightarrow (x := e', s)}$$

$$\text{(cong2)} \frac{(e,s) \rightarrow (e',s)}{(\text{if}_{\text{not } 0} e \text{ then } c_1 \text{ else } c_2, s) \rightarrow (\text{if}_{\text{not } 0} e' \text{ then } c_1 \text{ else } c_2, s)}$$

$$\text{(seq-l)} \frac{(c_1, s) \rightarrow (c_1', s')}{(c_1; c_2, s) \rightarrow (c_1'; c_2, s')}$$

$$\text{(seq-r)} \frac{(c_1, s) \rightarrow s'}{(c_1; c_2, s) \rightarrow (c_2, s')}$$

CMSC 631

24

## Big-step Rules

$$\text{(assign)} \frac{(e,s) \Downarrow i}{(x := e,s) \Downarrow s[i \backslash x]} \quad \text{(skip)} \frac{}{(\text{skip},s) \Downarrow s}$$

$$\text{(seq)} \frac{(c_1,s) \Downarrow s' \quad (c_2,s') \Downarrow s''}{(c_1;c_2,s) \Downarrow s''}$$

$$\text{(if0)} \frac{(e,s) \Downarrow (0,s) \quad (c_2,s) \Downarrow s'}{(\text{if}_{\text{not}0} e \text{ then } c_1 \text{ else } c_2,s) \Downarrow s'}$$

$$\text{(ifn)} \frac{(e,s) \Downarrow (i,s) \quad (c_1,s) \Downarrow s'}{(\text{if}_{\text{not}0} e \text{ then } c_1 \text{ else } c_2,s) \Downarrow s'} \quad (\text{where } i \neq 0)$$

CMSC 631

25

## Semantics of while

$$\text{(while)} \frac{(\text{if}_{\text{not}0} e \text{ then } (c; \text{while}_{\text{not}0} e \text{ do } c) \text{ else skip},s) \Downarrow s'}{(\text{while}_{\text{not}0} e \text{ do } c,s) \Downarrow s'}$$

- Similar to small-step while, but we evaluate the expansion in the premise

CMSC 631

26

## Non-terminating Programs

- If the meaning of a command is its final store, what is the meaning of non-terminating program?
  - Nothing!
  - Non-terminating programs have infinitely-sized derivation trees, and thus do not appear in the  $\Downarrow$  relation.
- Small step semantics is still useful, since we can prove things about intermediate computations.

CMSC 631

27

## What are these semantics good for?

- We started down this path for a reason:
  - How can I prove that a particular dataflow analysis (say) is correct?
- Now that we know what programs mean, we can prove that facts we learn about them through analysis are actually sensible.

CMSC 631

28

## Hybrid Semantics

- For the purposes of this proof, it is convenient to combine the small and big-step semantics
  - Take big-steps for expressions
  - Take small steps for commands (could have side-effects)
- Will make it simpler to set up the inductive hypothesis in our proof.

CMSC 631

29

## Hybrid Rules

$$\text{(assign)} \frac{(e,s) \Downarrow (i,s)}{(x := e,s) \rightarrow s[i \backslash x]} \quad \text{(skip)} \frac{}{(\text{skip},s) \rightarrow s}$$

$$\text{(if0)} \frac{(e,s) \Downarrow (0,s)}{(\text{if}_{\text{not}0} e \text{ then } c_1 \text{ else } c_2,s) \rightarrow (c_2,s)}$$

$$\text{(ifn)} \frac{(e,s) \Downarrow (i,s)}{(\text{if}_{\text{not}0} e \text{ then } c_1 \text{ else } c_2,s) \rightarrow (c_1,s)} \quad (\text{where } i \neq 0)$$

CMSC 631

30

## Semantics of while

$$\begin{array}{c} \text{(while0)} \frac{(e,s) \Downarrow (0,s)}{(\text{while}_{\text{not0}} e \text{ do } c,s) \rightarrow s} \\ \\ \text{(whilen)} \frac{(e,s) \Downarrow (i,s)}{(\text{while}_{\text{not0}} e \text{ do } c,s) \rightarrow (c; \text{while}_{\text{not0}} e \text{ do } c, s)} \\ \text{(where } i \neq 0) \end{array}$$

CMSC 631

31

## Example: Prove LV analysis is correct

- The **live variable analysis** determined
  - For each command  $c$ , those variables  $V$  for which there exists a path from  $c$  to a use of  $x \in V$  that does not assign to  $x$ .
  - Call  $LV(c)$  the set of live variables at command  $c$ .
- What do we want to prove?
  - Hint: we want to discover an invariant that is preserved by the small-step transition relation.

CMSC 631

32

## Live Variables Analysis as Equations

- Kill
  - $(x := e) = \{x\}$
  - $(\text{skip}) = \emptyset$
  - $(e) = \emptyset$
- Gen
  - $(x := e) = FV(e)$
  - $(\text{skip}) =$
  - $(e) = FV(e)$

$$\begin{array}{l} LV_{\text{out}}(c) = \emptyset \quad \text{if } \text{succ}(c) = \emptyset \\ LV_{\text{out}}(c) = \bigcup_{c' \in \text{succ}(c)} LV_{\text{in}}(c') \\ \\ LV_{\text{in}}(c) = \\ (LV_{\text{out}}(c) \setminus \text{Kill}(c)) \cup \text{Gen}(c) \end{array}$$

CMSC 631

33

## Make Equations into Constraints

$$\begin{array}{l} LV_{\text{out}}(c) \supseteq \emptyset \quad \text{if } \text{succ}(c) = \emptyset \\ LV_{\text{out}}(c) \supseteq \bigcup_{c' \in \text{succ}(c)} LV_{\text{in}}(c') \\ \\ LV_{\text{in}}(c) \supseteq \\ (LV_{\text{out}}(c) \setminus \text{Kill}(c)) \cup \text{Gen}(c) \end{array}$$

- Write  $live_{\text{in}}$  and  $live_{\text{out}}$  as solutions to these equations (maps commands to variable sets)
  - Refer to both solutions together as *live*
  - Write  $N$  and  $X$  as shorthand for  $live_{\text{in}}$  and  $live_{\text{out}}$

CMSC 631

34

## Useful Lemmas

- **Lemma:** the least solution of the constraint system and the least solution of the equation system coincide.
- **Lemma:** If *live* is a solution for some program  $c$ , then *live* is a solution to all  $c'$  that are “sub-programs” of  $c$ .
  - A sub-program has a subset of the statements and edges in the corresponding flow graph.
  - This implies that a correct solution is preserved during evaluation, since the flow graph either remains the same or shrinks.

CMSC 631

35

## Proof by Induction

- Need to construct the inductive hypothesis
  - When does the result of LV coincide with the actual evaluation of the program?
    - When the variables that are considered live are not assigned to
- Define
  - $s \sim_V s'$  iff  $\forall x \in V. s(x) = s'(x)$
  - States that two stores agree on the variables in  $V$ 
    - In the proof, these will be the variables that should be live.

CMSC 631

36

## Correctness

- If *live* is a solution to LV for program *c* then
    - If  $(c, s_1) \rightarrow (c', s_1')$  and  $s_1 \sim_{N(\text{init}(c))} s_2$  then  $\exists s_2'$  such that  $(c, s_2) \rightarrow (c', s_2')$  and  $s_1' \sim_{N(\text{init}(c))} s_2'$ .
    - If  $(c, s_1) \rightarrow s_1'$  and  $s_1 \sim_{X(\text{init}(c))} s_2$  then  $\exists s_2'$  such that  $(c, s_2) \rightarrow s_2'$  and  $s_1' \sim_{X(\text{init}(c))} s_2'$ .
  - **Corollary:** If *live* is a solution for *c* then
    - If  $(c, s_1) \rightarrow^* (c', s_1')$  and  $s_1 \sim_{N(\text{init}(c))} s_2$  then  $\exists s_2'$  such that  $(c, s_2) \rightarrow^* (c', s_2')$  and  $s_1' \sim_{N(\text{init}(c))} s_2'$ .
    - If  $(c, s_1) \rightarrow^* s_1'$  and  $s_1 \sim_{X(\text{init}(c))} s_2$  then  $\exists s_2'$  such that  $(c, s_2) \rightarrow^* s_2'$  and  $s_1' \sim_{X(\text{init}(c))} s_2'$ .
- Proof by induction on length of the derivation sequence.

CMSC 631

37

## Sample case

- (assign)
  - $(x := e, s_1) \rightarrow s_1[x|i]$  where  $(e, s_1) \Downarrow i$
  - $(x := e, s_2) \rightarrow s_2[x|j]$  where  $(e, s_2) \Downarrow j$
  - $N(c) = X(c) \setminus \{x\} \cup FV(e)$
- Thus
  - $s_1 \sim_{N(c)} s_2$  implies  $i = j$
- So, take  $s_2' = s_2[x|i]$  and we have  $s_1' \sim_{N(c)} s_2'$  as required.

CMSC 631

38