

# CS477 Formal Software Development Methods

Elsa L Gunter  
2112 SC, UIUC  
egunter@illinois.edu

<http://courses.engr.illinois.edu/cs477>

Slides based in part on previous lectures by Mahesh Vishwanathan, and  
by Gul Agha

March 21, 2014

## Simple Imperative Programming Language #2

$I \in \text{Identifiers}$

$N \in \text{Numerals}$

$E ::= N \mid I \mid E + E \mid E * E \mid E - E \mid I ::= E$

$B ::= \text{true} \mid \text{false} \mid B \& B \mid B \text{ or } B \mid \text{not } B$   
 $\mid E < E \mid E = E$

$C ::= \text{skip} \mid C; C \mid \{C\} \mid E$   
 $\mid \text{if } B \text{ then } C \text{ else } C \text{ fi}$   
 $\mid \text{while } B \text{ do } C$

# Transition Semantics

- Aka “small step semantics” or “Structured Operational Semantics”
- Defines a relation of “one step” of computation, instead of complete evaluation
  - Determines granularity of atomic computations
- Typically have two kinds of “result”: configurations and final values
- Written  $(C, m) \rightarrow (C', m')$  or  $(C, m) \rightarrow m'$

# Simple Imperative Programming Language #1 (SIMPL1)

$I \in \text{Identifiers}$

$N \in \text{Numerals}$

$E ::= N \mid I \mid E + E \mid E * E \mid E - E$

$B ::= \text{true} \mid \text{false} \mid B \& B \mid B \text{ or } B \mid \text{not } B$   
 $\mid E < E \mid E = E$

$C ::= \text{skip} \mid C; C \mid \{C\} \mid I ::= E$   
 $\mid \text{if } B \text{ then } C \text{ else } C \text{ fi}$   
 $\mid \text{while } B \text{ do } C$

# Commands - in English

- `skip` means done evaluating
- When evaluating an assignment, evaluate expression first
- If the expression being assigned is a value, update the memory with the new value for the identifier
- When evaluating a sequence, work on the first command in the sequence first
- If the first command evaluates to a new memory (ie completes), evaluate remainder with new memory

# Commands

Skip:  $(\text{skip}, m) \longrightarrow m$

Assignment: 
$$\frac{(E, m) \longrightarrow (E', m)}{(I ::= E, m) \longrightarrow (I ::= E', m)}$$

$(I ::= V, m) \longrightarrow m[I \leftarrow V]$

Sequencing:

$$\frac{(C, m) \longrightarrow (C'', m')}{(C; C', m) \longrightarrow (C''; C', m')} \quad \frac{(C, m) \longrightarrow m'}{(C; C', m) \longrightarrow (C', m')}$$

# Block Command

- Choice of level of granularity:
  - Choice 1: Open a block is a unit of work

$$(\{C\}, m) \longrightarrow (C, m)$$

- Choice 2: Blocks are syntactic sugar

$$\frac{(C, m) \longrightarrow (C', m')}{(\{C\}, m) \longrightarrow (C', m')} \quad \frac{(C, m) \longrightarrow m'}{(\{C\}, m) \longrightarrow m'}$$

# If Then Else Command - in English

- If the boolean guard in an `if_then_else` is true, then evaluate the first branch
- If it is false, evaluate the second branch
- If the boolean guard is not a value, then start by evaluating it first.



# If Then Else Command

$$(\text{if true then } C \text{ else } C' \text{ fi, } m) \longrightarrow (C, m)$$

$$(\text{if false then } C \text{ else } C' \text{ fi, } m) \longrightarrow (C', m)$$

$$(B, m) \longrightarrow (B', m)$$

---

$$(\text{if } B \text{ then } C \text{ else } C' \text{ fi, } m) \longrightarrow (\text{if } B' \text{ then } C \text{ else } C' \text{ fi, } m)$$

# While Command

$$\begin{array}{c} (\text{while } B \text{ do } C, m) \\ \longrightarrow \\ (\text{if } B \text{ then } C; \text{while } B \text{ do } C \text{ else skip fi}, m) \end{array}$$

- In English: Expand a **while** into a test of the boolean guard, with the true case being to do the body and then try the while loop again, and the false case being to stop.

# Example

$(y := i; \text{while } i > 0 \text{ do } \{i := i - 1; y := y * i\}, \langle i \mapsto 3 \rangle)$

$\longrightarrow \underline{\quad ? \quad}$

# Alternate Semantics for SIMPL1

- Can mix Natural Semantics with Transition Semantics to get larger atomic computations
- Use  $(E, m) \Downarrow v$  and  $(B, m) \Downarrow b$  for arithmetics and boolean expressions
- Revise rules for commands

# Revised Rules for SIMPL1

Skip:  $(\text{skip}, m) \longrightarrow m$

Assignment: 
$$\frac{(E, m) \Downarrow v}{(I ::= E, m)} \longrightarrow m[I \leftarrow V]$$

Sequencing:

$$\frac{(C, m) \longrightarrow (C'', m')}{(C; C', m) \longrightarrow (C''; C', m')} \quad \frac{(C, m) \longrightarrow m'}{(C; C', m) \longrightarrow (C', m')}$$

Blocks:

$$\frac{(C, m) \longrightarrow (C', m')}{(\{C\}, m) \longrightarrow (C', m')} \quad \frac{(C, m) \longrightarrow m'}{(\{C\}, m) \longrightarrow m'}$$

# If Then Else Command

$$\frac{(B, m) \Downarrow \text{true}}{(\text{if } B \text{ then } C \text{ else } C' \text{ fi}, m) \longrightarrow (C, m)}$$

$$\frac{(B, m) \Downarrow \text{false}}{(\text{if } B \text{ then } C \text{ else } C' \text{ fi}, m) \longrightarrow (C', m)}$$

# While Command

$$\frac{(B, m) \Downarrow \text{true}}{(\text{while } B \text{ do } C, m) \longrightarrow (C; \text{while } B \text{ do } C, m)}$$

$$\frac{(B, m) \Downarrow \text{false}}{(\text{while } B \text{ do } C, m) \longrightarrow m}$$

- Other more fine grained options exist (eg rule given before)

# Transition Semantics for SIMPL2?

- What are the choices and consequences for giving a transition semantics for the Simple Concurrent Imperative Programming Language #2, SIMP2?
- For finest grain transitions, summary:
  - Each rule for arithmetic or boolean expression must propagate changes to memory; instead of transitioning to a value, go to a value - memory pair



# Transition Semantics for SIMPL2

- Second assignment rule returns value:

$$(I ::= V, m) \longrightarrow (V, m[I \leftarrow V])$$

- Expressions as commands need two rules:

$$\frac{(E, m) \longrightarrow (E', m')}{(E, m) \longrightarrow (E', m')} \qquad \frac{(E, m) \longrightarrow (V, m')}{(E, m) \longrightarrow m'}$$

$$\text{Exp. as Comm.: } \frac{(E, m) \longrightarrow (E', m')}{(E, m) \longrightarrow (E', m')}$$

# Simple Concurrent Imperative Programming Language (SCIMP1)

$I \in \text{Identifiers}$

$N \in \text{Numerals}$

$E ::= N \mid I \mid E + E \mid E * E \mid E - E$

$B ::= \text{true} \mid \text{false} \mid B \& B \mid B \text{ or } B \mid \text{not } B$   
 $\mid E < E \mid E = E$

$C ::= \text{skip} \mid C; C \mid \{C\} \mid I ::= E \mid C \parallel C'$   
 $\mid \text{if } B \text{ then } C \text{ else } C \text{ fi}$   
 $\mid \text{while } B \text{ do } C$

- $C_1 \parallel C_2$  means that the actions of  $C_1$  and done at the same time as, “in parallel” with, those of  $C_2$
- True parallelism hard to model; must handle collisions on resources
  - What is the meaning of

$$x := 1 \parallel x := 0$$

- True parallelism exists in real world, so important to model correctly

# Interleaving Semantics

- Weaker alternative: interleaving semantics
- Each process gets a turn to commit some atomic steps; no preset order of turns, no preset number of actions
- No collision for  $x := 1 \parallel x := 0$ 
  - Yields only  $\langle x \mapsto 1 \rangle$  and  $\langle x \mapsto 0 \rangle$ ; no collision
- No simultaneous substitution:  $x := y \parallel y := x$  results in  $x$  and  $y$  having the same value; not in swapping their values.

# Coarse-Grained Interleaving Semantics for SCIMPL1 Commands

- Skip, Assignment, Sequencing, Blocks, If\_Then\_Else, While unchanged
- Need rules for  $\parallel$

$$\frac{(C_1, m) \longrightarrow (C'_1, m')}{(C_1 \parallel C_2, m) \longrightarrow (C'_1 \parallel C_2, m')} \qquad \frac{(C_1, m) \longrightarrow m'}{(C_1 \parallel C_2, m) \longrightarrow (C_2, m')}$$

$$\frac{(C_2, m) \longrightarrow (C'_2, m')}{(C_1 \parallel C_2, m) \longrightarrow (C_1 \parallel C'_2, m')} \qquad \frac{(C_2, m) \longrightarrow m'}{(C_1 \parallel C_2, m) \longrightarrow (C_1, m')}$$

# Simple Concurrent Imperative Programming Language #2 (SCIMP2)

$I \in \text{Identifiers}$

$N \in \text{Numerals}$

$E ::= N \mid I \mid E + E \mid E * E \mid E - E$

$B ::= \text{true} \mid \text{false} \mid B \& B \mid B \text{ or } B \mid \text{not } B$   
 $\mid E < E \mid E = E$

$C ::= \text{skip} \mid C; C \mid \{C\} \mid I ::= E \mid C \parallel C' \mid \text{sync}(E)$   
 $\mid \text{if } B \text{ then } C \text{ else } C \text{ fi}$   
 $\mid \text{while } B \text{ do } C$

# Informal Semantics of sync

- $\text{sync}(E)$  evaluates  $E$  to a value  $v$
- Waits for another parallel command waiting to synchronize on  $v$
- When two parallel commands are both waiting to synchronize on a value  $v$ , they may both stop waiting, move past the synchronization, and carry on with whatever commands they each have left
- Only two processes may synchronize at a time (in this version).
- Problem: How to formalize?

# Labeled Transition System (LTS)

A **labeled transition system (LTS)** is a 4-tuple  $(Q, \Sigma, \delta, I)$  where

- $Q$  set of states
  - $Q$  finite or countably infinite
- $\Sigma$  set of labels (aka actions)
  - $\Sigma$  finite or countably infinite
- $\delta \subseteq Q \times \Sigma \times Q$  transition relation
- $I \subseteq Q$  initial states

Note: Write  $q \xrightarrow{\alpha} q'$  for  $(q, \alpha, q') \in \delta$ .



# Example: Candy Machine

- $Q = \{\text{Start, Select, GetMarsBar, GetKitKatBar}\}$
- $I = \{\text{Start}\}$
- $\Sigma = \{\text{Pay, ChooseMarsBar, ChooseKitKatBar, TakeCandy}\}$
- $\delta = \left\{ \begin{array}{l} (\text{Start, Pay, Select}) \\ (\text{Select, ChooseMarsBar, GetMarsBar}) \\ (\text{Select, ChooseKitKatBar, GetKitKatBar}) \\ (\text{GetMarsBar, TakeCandy, Start}) \\ (\text{GetKitKatBar, TakeCandy, Start}) \end{array} \right\}$