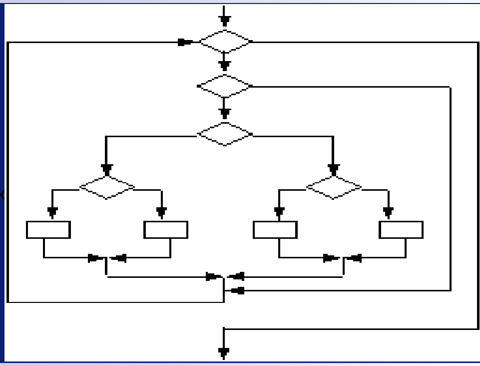
Spring 2008



Program correctness

Program verification and operational semantics

Marcello Bonsangue

Leiden Institute of Advanced Computer Science
Research & Education

System verification

- Model checking verification is
  - model based $M,s \models \phi$
  - fully automatic
  - intended for hardware or software systems with finitely many states
    - control is the main issue
    - no complex data
    - mainly reactive
      - reaction-> computation -> reaction -> …
      - not intended to terminated

System verification

- Program verification:
  - Proof based $\Gamma \vdash \phi$
    - It is impossible to check infinite states !
  - Semi-automatic
  - intended for software systems with possibly infinite states
    - mainly sequential
    - transformational
      - input -> computation -> output
      - like methods of an object

Program verification

The verification framework:

1. Convert an informal specification S in an ‘equivalent’ formula ϕ of some logic
2. Write a program P realizing ϕ (or S)
3. Prove that P satisfies the formula ϕ

A simple language

- Syntactic sets associated to the language:
  - N positive and negative integers n,...
  - B truth values true,false
  - Var program variables x,…
  - Aexp arithmetic expressions a,...
  - Bexp boolean expressions b,...
  - Com commands c, …

Arithmetic expressions

- A ::= n | x | (A+A) | (A-A) | (A*A)

  where n ∈ N and x ∈ Var
- Here * binds more tightly than - and +
- Examples:

2 + 3 * 4 - 5	is	(2 + 3) * (4 - 5)
- 3	is	(0 - 3)
- -5	is	(0 - -5)
2 + x + 5	is	(2 + x) + 5

Boolean expressions

- $B ::= true \mid false \mid \neg B \mid B \wedge B \mid B \vee B \mid A < A$

- Examples:

  $A_1 = A_2$ is $\neg(A_1 < A_2) \wedge \neg(A_2 < A_1)$

  $A_1 \neq A_2$ is $\neg(A_1 = A_2)$

- Boolean expression are built on top of arithmetic expressions
  - $3+5 < 9$
  - $4 = 5$ is a correct boolean expression !!!
  - true < 10 is not a boolean expression

Commands

- C ::= <u>skip</u> |
  x := A |
  C;C |
  <u>if</u> B <u>then</u> C <u>else</u> C <u>fi</u> |
  <u>while</u> B <u>do</u> C <u>od</u>

- Example (Fact1)

```
y := 1;
z := 0;
while z ≠ 0 do
        z := z + 1;
        y := y*z
        od
```

The behaviour

- We need a formal model to understand correctly the behavior of a program

- State $\sigma : Var \rightarrow N$

- An arithmetic expression a in a state σ evaluates to an integer n

$$\underbrace{\langle a, \sigma \rangle}_{\text{configuration}} \rightarrow n$$

Evaluating arithmetic expressions

- $<n, \sigma> \rightarrow n$
- $<x, \sigma> \rightarrow \sigma(x)$
- If n is the sum of n_1 and n_2

$$\frac{<a_1, \sigma> \rightarrow n_1 \quad <a_2, \sigma> \rightarrow n_2}{<a_1 + a_2, \sigma> \rightarrow n}$$

- If n is the subtraction of n_2 from n_1

$$\frac{<a_1, \sigma> \rightarrow n_1 \quad <a_2, \sigma> \rightarrow n_2}{<a_1 - a_2, \sigma> \rightarrow n}$$

- If n is the product of n_1 and n_2

$$\frac{<a_1, \sigma> \rightarrow n_1 \quad <a_2, \sigma> \rightarrow n_2}{<a_1 * a_2, \sigma> \rightarrow n}$$

An Example Derivation

- What is the n such that

$$<(3+4)-(x*2), \sigma> \rightarrow n \ ?$$

Semantics of arithmetic expressions

- Two arithmetic expressions are equivalent if they evaluate to the same value in all states

$$a_1 \approx a_2$$

$$\text{iff}$$

$$(\forall n \in N. \forall \sigma. <a_1, \sigma> \rightarrow n \Leftrightarrow <a_2, \sigma> \rightarrow n)$$

- Examples:
  - $<2+3, \sigma> \rightarrow 5$ and $<3+2, \sigma> \rightarrow 5$ thus $(2+3) \approx (3+2)$
  - 2+x is not equivalent to 2+3 because there are states in which x evaluates to an integer different from 3

Evaluating Boolean expressions

- $<true, \sigma> \rightarrow T$
- $<false, \sigma> \rightarrow F$
- $$\frac{<b, \sigma> \rightarrow T}{<\neg b, \sigma> \rightarrow F} \qquad \frac{<b, \sigma> \rightarrow F}{<\neg b, \sigma> \rightarrow T}$$
- $$\frac{<b_1, \sigma> \rightarrow t_1 \quad <b_2, \sigma> \rightarrow t_2}{<b_1 \wedge b_2, \sigma> \rightarrow t}$$

where t = T if both t_1 = T and t_2 =T, otherwise t = F

Evaluating boolean expressions

- $$\frac{\langle b_1, \sigma\rangle \to t_1 \qquad \langle b_2, \sigma\rangle \to t_2}{\langle b_1 \vee b_2, \sigma\rangle \to t}$$

  where $t = T$ if $t_1 = T$ or $t_2 = T$, and $t = F$ otherwise

- If n_1 is less than n_2 then

  $$\frac{\langle a_1, \sigma\rangle \to n_1 \qquad \langle a_2, \sigma\rangle \to n_2}{\langle a_1 < a_2, \sigma\rangle \to T}$$

- If n_1 is greater than or equal to n_2 then

  $$\frac{\langle a_1, \sigma\rangle \to n_1 \qquad \langle a_2, \sigma\rangle \to n_2}{\langle a_1 < a_2, \sigma\rangle \to F}$$

Semantics of Boolean expressions

- Two Boolean expressions are equivalent if they evaluate to the same truth value in all states

$$b_1 \approx b_2$$

iff

$$(\forall\sigma.\ \langle b_1, \sigma\rangle \rightarrow T \Leftrightarrow \langle b_2, \sigma\rangle \rightarrow T)$$

- We could improve the evaluation of Boolean expressions using
  - a left-first sequential strategy
  - a parallel strategy

The command behaviour

- A program may
  - terminate in a final state or
  - diverge and never yield a final state

- We denote by

$$\langle c, \sigma \rangle \to \sigma'$$

the execution of a command c in an initial state σ and terminating in a final state σ'

- Recall: $$\sigma[n/x](y) = \begin{cases} n & \text{if } x = y \\ \sigma(y) & \text{if } x \neq y \end{cases}$$

Executing commands I

- $\langle \text{skip}, \sigma \rangle \to \sigma$
- $$\frac{\langle a, \sigma \rangle \to n}{\langle x := a, \sigma \rangle \to \sigma[n/x]}$$
- $$\frac{\langle c_1, \sigma \rangle \to \sigma'' \qquad \langle c_2, \sigma'' \rangle \to \sigma'}{\langle c_1; c_2, \sigma \rangle \to \sigma'}$$
- $$\frac{\langle b, \sigma \rangle \to T \quad \langle c_1, \sigma \rangle \to \sigma'}{\langle \underline{\text{if}}\ b\ \underline{\text{then}}\ c_1\ \underline{\text{else}}\ c_2\ \underline{\text{fi}}, \sigma \rangle \to \sigma'}$$
- $$\frac{\langle b, \sigma \rangle \to F \quad \langle c_2, \sigma \rangle \to \sigma'}{\langle \underline{\text{if}}\ b\ \underline{\text{then}}\ c_1\ \underline{\text{else}}\ c_2\ \underline{\text{fi}}, \sigma \rangle \to \sigma'}$$

Example: MAX

- What is the final state σ' of

  <if x < y then z:=y else z:= x fi, σ> → σ'

  for σ(x) = 2, σ(y) = 1 and σ(z) = 0 ?

Executing commands II

- $$\frac{\langle b, \sigma\rangle \rightarrow F}{\langle \underline{\text{while}}\ b\ \underline{\text{do}}\ c\ \underline{\text{od}}, \sigma\rangle \rightarrow \sigma}$$

- $$\frac{\langle b, \sigma\rangle \rightarrow T \qquad \langle c, \sigma\rangle \rightarrow \sigma'' \qquad \langle \underline{\text{while}}\ b\ \underline{\text{do}}\ c\ \underline{\text{od}}, \sigma''\rangle \rightarrow \sigma'}{\langle \underline{\text{while}}\ b\ \underline{\text{do}}\ c\ \underline{\text{od}}, \sigma\rangle \rightarrow \sigma'}$$

Semantics of commands

- Two commands are equivalent if when executed from the same initial state they terminate in the same final state

$$c_1 \approx c_2$$

$$\text{iff}$$

$$(\forall \sigma, \sigma'.\ \langle c_1, \sigma\rangle \to \sigma' \ \Leftrightarrow\ \langle c_2, \sigma\rangle \to \sigma')$$

- Examples
  - x := x ≈ <u>skip</u>
  - <u>while</u> b <u>do</u> c of ≈ <u>if</u> b <u>then</u> c; <u>while</u> b <u>do</u> c od
    <u>else</u> <u>skip</u>
    fi

Execution of Commands

- The order of evaluation is important and explicit.
  - c_1 is evaluated before c_2 in c_1; c_2
  - c_2 is not evaluated in <u>if</u> true <u>then</u> c_1 <u>else</u> c_2 <u>fi</u>
  - b is evaluated first in <u>if</u> b <u>then</u> c_1 <u>else</u> c_2 <u>fi</u>
  - c is not evaluated in "<u>while</u> false <u>do</u> c <u>od</u>

- The execution rules suggest an interpreter but abstract from a concrete one

- Execution is deterministic: only one rule can be applied at time.