Illinois Institute of Technology Homework 3

# Programs and Their Semantics

*CS 536: Science of Programming; Due Mon Sep 19*

9/14: p.1

***A. Why?***

* Our simple programming language is a model for the kind of constructs seen in actual languages.
* Our programs stand for state transformers.

## B. Outcomes

After this homework, you should be able to

* Translate simple programs into our language
* Calculate the meaning of small programs

## C. Questions [50 points total]

General note: For ASCII versions of α, β, σ, and τ, you can spell them out (alpha, beta, etc.) or abbreviate them using 'a, 'b, 's, 't.

For Problems 1 – 3, translate the given C-like programs into our programming language.

1. [6 points] for (j = m, x = 1; j >= 1; ) x \*= y[--j];

Ans: j := m;

x :=1;

while j ≥ 1

do

j:=j-1;

x=x\*y[j];

od

1. [6 points] j = m; x = 1; while (--j >= 0) x \*= y[j];

Ans: j:=m;

x:=1;

j:=j-1;

while j ≥ 0

do

x:=x\*y[j];

j:=j-1;

od

1. [6 points] x = 1; j = 0; while (j++ < m) x \*= y[j];

Ans: x:=1;

j:=0;

while j < m

do

j:=j+1;

x:=x\*y[j];

od

j:=j+1

For Problems 4 and 5, let σ = {(x, α), (y, β)}. For each *S* below, evaluate 〈*S*, σ〉 to completion. Also give *M*(*S*, σ).

1. [6 points] *S* ≡ t := x ; x := y; y := t

Ans: M(t := x ;x := y;y := t,σ)  
=M(x := y;y := t,M(t := x ,σ))  
=M(x := y;y := t,σ[t ↦α])  
=M(y := t,M(x := y,σ[t ↦α]))  
=M(y := t,σ[t ↦α][x↦β])  
= σ[t ↦  α][x ↦ β][y ↦ α].

1. [6 points] *S* ≡ ***if*** x < 0 ***then*** y := -x ***else*** y := x ***fi***

Ans: Ifα < 0, then M(if x < 0 then y := -x else y := x fi, σ)  
 = M(y := -x, σ) = σ[y ↦ –α]  
 If α ≥ 0, then M(if x < 0 then y := –x else y := x fi, σ)  
 = M(y := x, σ) = σ[y ↦ α].

1. [10 points] Let *S* ≡ s := 0; ***while*** n ≥ 0 ***do*** *S*₁ ***od*** where the loop body *S*₁ ≡ s := s+n; n := n-1.
   1. [2 points] If σ(n) < 0, what are the operational and denotational semantics of *S* in σ? (I.e., what is the sequence 〈*S*, σ〉 → … → 〈*E*, *M*(*S*, σ)〉? What is *M*(*S*, σ)?)

Ans: when σ(n) < 0 〈*S*, σ〉 = 〈s := 0;W, σ〉 = 〈W, σ[s ↦ 0]〉 = 〈E, σ[s ↦ 0]〉

M(S, σ) = σ[s ↦ 0]

* 1. [3 points] Give the operational and denotational semantics of the loop body *S*₁ in an arbitrary state τ.

Ans: 〈*S1*, τ 〉= 〈s := s+n; n := n-1, τ 〉

=〈n := n-1, τ [s ↦  τ(s)+ τ(n)]〉

=〈E, τ` [ n :=  τ`(n)-1]〉 where τ `=τ [s ↦  τ(s)+ τ(n)]

M(*S1*, τ) =〈E, τ` [ n :=  τ`(n)-1]〉 where τ `=τ [s ↦  τ(s)+ τ(n)]

* 1. [5 points] Let σ(n) = 3. Write out the complete execution sequence for 〈*S*, σ〉 and give *M*(*S*, σ). Use multi-step execution: Follow the style of Example 8 in Lecture 6.

Ans: 〈 *S1*, τ 〉=〈s := s+n; n := n-1, τ 〉

= 〈n := n-1, τ [s ↦  τ(s)+ τ(n)]〉

= 〈E, τ[s ↦  τ(s)+ τ(n)][ n ↦  τ(n)-1]〉

〈S₁, τ〉 → ² 〈E, τ[s ↦  τ(s)+ τ(n)][ n ↦  τ(n)-1]〉

If τ(n ≥ 0)=T then 〈W, σ〉→ 3 〈W, τ[s ↦  τ(s)+ τ(n)][ n ↦  τ(n)-1]〉

If σ(n) = 3

〈S, σ〉 = 〈s := 0 ; W, σ〉

→ 〈W, σ[s ↦ 0]〉

→ 3 〈W, τ[s ↦  τ(s)+ σ(n)][ n ↦  σ(n)-1]〉 τ= σ[s ↦ 0]

= 〈W, τ[s ↦  0+ σ(n)][ n ↦  σ(n)-1]〉

= 〈W, τ[s ↦  0+3][ n ↦ 3-1]〉

= 〈W, τ[s ↦  3][ n ↦ 2]〉

→ 3 〈W, τ`[s ↦  τ`(s)+ σ(n)][ n ↦  σ(n)-1]〉 τ`= τ[s ↦  3][ n ↦ 2]

= 〈W, τ`[s ↦  5][n ↦  1]〉

→ 3 〈W, τ2[s ↦  τ2(s)+ τ2(n)][ n ↦  τ2(n)-1]〉 τ2= τ`[s ↦  5][n ↦  1]

= 〈W, τ2[s ↦ 6][ n ↦  0]〉

→ 3 〈W, τ3[s ↦  τ3(s)+ τ3(n)][ n ↦  τ3(n)-1]〉 τ3= τ2[s ↦ 6][ n ↦  0]

= 〈W, τ3[s ↦ 6][ n ↦  -1]〉

→ 〈E, τ3[s ↦ 6][ n ↦  -1]〉

*M*(*S*, σ)= 〈E, τ3[s ↦ 6][ n ↦  -1]〉 where τ3= τ2[s ↦ 6][ n ↦  0]

1. [10 points] Give a definition for a predicate function Reversed(b, i, j, m) that works as follows: If you take the m-long subsequence of b starting at b[i] and compare it to the m-long subsequence of b starting at b[j], you find the two are reverses of each other. E.g., if b = (1, 2, 3, 4, 5, 4, 5, 4, 3, 2, 3), then
   * Reversed(b, 1, 6, 4) is true (comparing 2, 3, 4, 5 vs 5, 4, 3, 2).
   * Reversed(b, 7, 2, 2) is true (comparing 4, 3 vs 3, 4).
   * Reversed(b, 3, 5, 1) is true (comparing 4 vs 4).
   * Reversed(b, 5, 3, 3) is true (comparing 4, 5, 4 vs 4, 5, 4).
   * Reversed(b, 0, 9, 2) is false (comparing 1, 2 vs  2, 6).
   * Reversed(b, i, j, 0) is true if i and j are both legal indexes.
   * Reversed(b, i, j, m) is false if m < 0 or if i or j is an illegal index.
   * Reversed(b, 10, 2, 2) is false (there aren’t 2 elements to the right of b[i]). Similarly, Reversed(b, 2, 10, 2) is false.

Your definition can be recursive or non-recursive, your choice. Remember, this is supposed to be a predicate function, so its definition must be a predicate, not a program. E.g., OK: Reversed(b, i, j, m) ≡ m ≥ 0 → ∀ etc.

WRONG: Reversed(b, i, j, m) ≡ if (m < 0) return 1; else…

Ans: Reversed(b, i, j, m) ≡  
 m > 0 → Index(i) ∧ Index(j+m-1) ∧ b[i] = b[j+m-1]  
 ∧ Reversed(b, i+1, j, m-1)