Nome: Rodrigo Ferreira Araújo
Matrícula: 2020006990

2: Semântica Formal

1) z := 1;
   x := n;
   y := m;
   WHILE ¬(y=0) DO(z := z * x;
   y := y - 1)

2) B[¬(x = 1)]s ?= B[(x = 1)]s
   ?= A[x]s = A[1]s
   ?= sx = N[1]
   ?= 3 = 1
   ?= ff (false)
   ?= B[¬(ff)] = tt (true)

3) true[y → a0] = tt
   false[y → a0] = ff
   (¬a1)[y → a0] = ¬(b[y → a0])
   (a1 = a2)[y → a0] = a1[y → a0] = a2[y → a0]
   (a1 ≤ a2)[y → a0] = a1[y → a0] ≤ a2[y → a0]
   (a1 ∧ a2)[y → a0] = b(a1[y → a0]) ∧ b(a2[y 7→ a0])

4) ?

5) while ¬(x=1) do (y:=y*x; x:=x-1): Não é possível determinar se esse programa sempre termina ou não. Note que isso é totalmente dependente do valor inicial de x: caso x < 1, a condição do while sempre será verdadeira com a execução de "x:=x-1", provocando um loop infinito. Caso contrário (x >= 1), com a execução de "x:=x-1", temos a certeza que em algum momento a condição de parada (x == 1) do WHILE será atingida.

   while 1 ≤ x do (y:=y*x; x:=x-1): Esse programa sempre irá terminar, pois a condição (1 <= x) sempre será violada em algum momento independentemente do valor inicial de x, dado a execução de "x:=x-1": caso x >= 1, x eventualmente alcançará um valor menor que 1; caso contrário (x < 1), o loop termina instantaneamente.

while true do skip: Esse programa nunca terminará. O comando skip sozinho nunca será capaz de negar a condição do loop, um whilett.

6)

a) (While(b, stm)) => if (evalB b s) then let val S = (evalStm stm s)
in (evalStm stm S)
end
else s

b) (S, s) -> s', (repeat S until b)s' -> s'' if B[b]s' = ff (Continue no loop)
(repeat S until b)s -> s''

(S, s) -> s' if B[b]s' = tt (Saia do loop)
(repeat S until b)s -> s''

c) ?

d) ?

3: Bindings e Escopos

1)

a) Saída: x = 1. Com escopo estático, a variável x escrita por write(x) na linha 12 sofreu alterações somente no escopo em que foi criada, no caso, no escopo da função “r”. Assim, x é inicializado com 1 e é escrito por write, sendo que a função “q”, estando fora do escopo de criação do x, não o altera.

b) Saída: x = 2. Com escopo dinâmico, quaisquer sobrescritas da variável x, independentemente de escopo, serão registradas. Nesse sentido, primeiro, inicializamos x := 2 no escopo mais externo, então atribuímos x := 1 no escopo de “f”, alteramos novamente para x := 2 incrementando-o de 1 na função “q” e, por fim, escrevemos x := 2.

2)

a)
Escopo de g => Bloco1;
Escopo do primeiro let => Bloco 2 (onde a função f é declarada);
Escopo de f => Bloco 3
Escopo de h => Bloco 4
Escopo do segundo let => Bloco 5 (onde a função f é chamada);

b) Nomes: g, x, inc(linha 3 e linha 7), f, y, h, z,

c)

[g,x] → Bloco 1
[inc(linha 3), f, h] → Bloco 2
y → Bloco 3
z → Bloco 4
inc(linha 7) → Bloco 5

d)

g(5) := 6. A cadeia de execução será: g(5) → h(5) → f(5) → return 5 + inc. Como SML possui escopo estático, o retorno de g(5) dependerá do escopo de definição de definição de inc e da função f. Como o inc da linha 3 e a função f pertencem ao mesmo escopo (Bloco 2) inc assumirá valor := 1, portanto, 5 + 1 = 6. Caso SML possuísse escopo dinâmico, a última sobrescrita de inc (linha 7) prevalece sobre a primeira definição, o que produz 5 + 2 = 7 na chamada da função f no escopo do segundo let.

3) VPL

4) VPL

5)

a) A função cresce de forma exponencial: um valor de x é comparado com todos os outros valores x da lista para todos os valores da lista.

b)
```
fun max [] = 0
    | max (h::[]) = h
    | max (h::t) = let val x = (max t) in if h > x then h else x end;
```

6) VPL

7)

```
fun expr() = let val x = 1
             in (let val x = 2 in x + 1 end)
             +
             (let val y = x + 2 in y + 1 end)
             end;
```