Universdad Nacional Autónoma de México

Lenguajes de Programacion
Examen Parcial IV

- Edgar Montiel Ledesma
  317317794
- Carlos Daniel Cortes Jimenez
  420004846

---

1. Considera el siguiente programa en el lenguaje While :

   new z = 0;
   while (y < x + 1) do
   (z := z + 1;
   x := x − y)
   end

   a) Ejecuta el programa en el estado en el que σ(x) = 17 y σ(y) = 5 ¿Cual es el estado resultante de la evaluacion?
   Inicialmente, tenemos σ(x) = 17 y σ(y) = 5. Vamos a ejecutar el programa paso a paso:

   Iteración 1:
   - z := 0
   - y < x + 1 es verdadero ya que 5 < 17 + 1
   - z := z + 1; z=1
   - x := x - y; x=12

   Iteración 2:
   - y < x + 1 es verdadero ya que 5 < 12 + 1
   - z := z + 1; z=2
   - x := x - y; x=7

   Iteración 3:
   - y ¡ x + 1 es verdadero ya que 5 ¡ 7 + 1
   - z := z + 1; z=3
   - x := x - y; x=2

   Iteración 4:
   - y ¡ x + 1 es verdadero ya que 5 ¡ 2 + 1
   - z := z + 1; z=4
   - x := x - y; x=-3

Iteración 5:

- y ¡ x + 1 es falso ya que 5 ¿= -3 + 1

El estado resultante es σ(x) = -3 y σ(y) = 5.

b) Da un estado σ tal que si se evalua el programa anterior con dicho estado la evaluacion se ciclaria infinitamente.

Para que el programa entre en un bucle infinito, necesitamos un estado en el que la condición del bucle siempre sea verdadera. La condición del bucle es $y < x + 1$. Por lo tanto, si tenemos y igual a cualquier número mayor que x+1, el bucle se ejecutará infinitamente.

Un ejemplo de un estado que causaría una ejecución infinita podría ser σ(x) = 5 y σ(y) = 10. En este caso, la condición $y < x + 1$ siempre será verdadera ($10 < 5 + 1$), y el bucle se ejecutará sin llegar a una condición de salida.

2. Extiende el lenguaje While con el operador:

$$\text{for x := a1 to a2 do c}$$

esto es:

a) Modifica la estructura de la maquina W (agregando marcos, estados o transiciones) para evaluar la expresion for.

La máquina W, que se utiliza para evaluar expresiones en el lenguaje While, podría modificarse para incorporar el operador for. La estructura general de la máquina W podría ampliarse para manejar la nueva construcción for de la siguiente manera:

1. Marcos de pila adicionales: Se pueden agregar marcos de pila para manejar la ejecución del bucle for. Estos marcos contendrían información sobre la variable de iteración, los límites del bucle y la declaración dentro del bucle.
2. Instrucciones adicionales: Se agregarían instrucciones específicas para manejar la inicialización de la variable de iteración, la verificación de la condición del bucle, la ejecución del cuerpo del bucle y la actualización de la variable de iteración.

La máquina W podría tener nuevas instrucciones como FOR_START, FOR_CONDITION, FOR_BODY, y FOR_UPDATE para gestionar las diferentes partes de la construcción for.

b) Da las reglas de semantica estatica para verificacion de tipos para el nuevo operador for.

Para el nuevo operador for, se necesitarían reglas de verificación de tipos para garantizar que las expresiones a1, a2 y c sean de tipos apropiados.

Las reglas podrían incluir:

- a1 y a2 deben ser expresiones enteras.
- c debe ser una expresión que cumpla con ciertos requisitos para ser ejecutada en un bucle.

c) ¿Es posible definir el operador for como azucar sintactica dentro del lenguaje While? justifica tu respuesta.

Sí, es posible definir el operador for como azúcar sintáctica dentro del lenguaje While. La azúcar sintáctica es una forma de escribir código que se traduce a una forma más básica del lenguaje. En este caso, podríamos definir la construcción for en términos de construcciones más básicas del lenguaje While, como while y asignaciones.

Por ejemplo, la construcción for x := a1 to a2 do c podría ser traducida a algo similar a:

```
{
  c;
  x := x + 1;
}
```

En este caso, la estructura for se traduce a un bucle while con una asignación adicional para actualizar la variable de iteración. Esta traducción permite que el operador for sea implementado en términos de las construcciones fundamentales del lenguaje While.

3. Decimos que dos programas en el lenguaje While son equivalentes ($c_1 \equiv_w c_2$) si y solo sı la ejecucion de ambos programas resulta en el mismo estado, es decir, si para todo estado de las variables σ, $\blacklozenge \succ \langle c_1, \sigma\rangle \longrightarrow_W^* \blacklozenge \prec \sigma'$ y $\blacklozenge \succ \langle c_2, \sigma\rangle \longrightarrow_W^* \blacklozenge \prec \sigma'$ entonces $c_1 \equiv_w c_2$.
Con la definicion de equivalencia anterior, demuestra o da un contraejemplo de lo siguiente:

a) $\equiv_w$ realmente es una relacion de equivalencia. Esto es, demuestra que la relacion $\equiv_w$ es transitiva, reflexiva y simetrica.

Para demostrar que $\equiv_w$ es una relación de equivalencia, necesitamos probar tres propiedades: reflexividad, simetría y transitividad.

Reflexividad: Para cualquier programa c en While, $c \equiv_w c$.

- Para demostrar la reflexividad, consideremos un estado σ arbitrario.
  - Si $\blacklozenge \succ \langle c, \sigma\rangle \longrightarrow_W^* \blacklozenge \prec \sigma'$, entonces el mismo programa c resulta en el estado σ'.
  - Por lo tanto, $c \equiv_w c$, ya que ambos programas producen el mismo estado σ'.

Simetría: Si $c_1 \equiv_w c_2$, entonces $c_2 \equiv_w c_1$.

- Supongamos que para cualquier estado σ, $\blacklozenge \succ \langle c_1, \sigma\rangle \longrightarrow_W^* \blacklozenge \prec \sigma'$ y $\blacklozenge \succ \langle c_2, \sigma\rangle \longrightarrow_W^* \blacklozenge \prec \sigma'$.
- Esto implica que c_1 y c_2 conducen al mismo estado σ'.
- Si $c_1 \equiv_w c_2$, entonces $c_2 \equiv_w c_1$ debido a que ambos programas resultan en el mismo estado σ'.

Transitividad: Si $c_1 \equiv_w c_2$ y $c_2 \equiv_w c_3$, entonces $c_1 \equiv_w c_3$.

- Supongamos que para cualquier estado σ, $\blacklozenge \succ \langle c_1, \sigma\rangle \longrightarrow_W^* \blacklozenge \prec \sigma'$ y $\blacklozenge \succ \langle c_2, \sigma\rangle \longrightarrow_W^* \blacklozenge \prec \sigma'$, también $\blacklozenge \succ \langle c_2, \sigma\rangle \longrightarrow_W^* \blacklozenge \prec \sigma''$ y $\blacklozenge \succ \langle c_3, \sigma\rangle \longrightarrow_W^* \blacklozenge \prec \sigma''$.
- Esto implica que c_1 y c_2 conducen al mismo estado σ' y c_2 y c_3 conducen al mismo estado σ''.
- Por lo tanto, c_1 y c_3 producen el mismo estado σ'', lo que demuestra la transitividad de $\equiv_w$.

Entonces, podemos decir que $\equiv_w$ es una relación de equivalencia en el lenguaje While.

b) c;skip $\equiv_w c$

Para probar que c;skip $\equiv_w c$ en el lenguaje While, necesitamos demostrar que para cualquier estado σ:

1.- Si $\blacklozenge \succ \langle c, \text{skip}, \sigma\rangle \longrightarrow_W^* \blacklozenge \prec \sigma'$, entonces $\blacklozenge \succ \langle c, \sigma\rangle \longrightarrow_W^* \blacklozenge \prec \sigma'$.
2.- Si $\blacklozenge \succ \langle c, \sigma\rangle \longrightarrow_W^* \blacklozenge \prec \sigma'$, entonces $\blacklozenge \succ \langle c, \text{skip}, \sigma\rangle \longrightarrow_W^* \blacklozenge \prec \sigma'$.

Primero resolvemos 1:

Si $\blacklozenge \succ \langle c, \text{skip}, \sigma\rangle \longrightarrow_W^* \blacklozenge \prec \sigma'$, esto significa que $\blacklozenge \succ \langle c, \sigma\rangle \longrightarrow_W^* \blacklozenge \prec \sigma'$. Esto se debe a que la semántica operacional de While establece que la ejecución de un comando "skip"no altera el estado. Por lo tanto, si ejecutamos c; skip básicamente estamos ejecutando solo c, lo que lleva al mismo estado σ'.

Ahora resolvemos 2:

Si $\blacklozenge \succ \langle c, \sigma\rangle \longrightarrow_W^* \blacklozenge \prec \sigma'$, esto significa que $\blacklozenge \succ \langle c, \text{skip}, \sigma\rangle \longrightarrow_W^* \blacklozenge \prec \sigma'$. Por lo tanto, si ejecutamos c y luego skip, el estado sigue siendo σ', ya que skip no realiza cambios en el estado.

Por lo que, si ejecutamos c;skip, simplemente c en el lenguaje While, se llega al mismo estado σ'. Por lo tanto c;skip $\equiv_w c$.

c) $c_1; c_2 \equiv_w c_2; c_1$

Para probar que $c_1; c_2 \equiv_w c_2; c_1$ en el lenguaje While, necesitamos demostrar que para cualquier estado σ:

1.- Si $\blacklozenge \succ \langle c_1, c_2, \sigma\rangle \longrightarrow_W^* \blacklozenge \prec \sigma'$, entonces $\blacklozenge \succ \langle c_2, c_1, \sigma\rangle \longrightarrow_W^* \blacklozenge \prec \sigma'$.
2.- Si $\blacklozenge \succ \langle c_2, c_1, \sigma\rangle \longrightarrow_W^* \blacklozenge \prec \sigma'$, entonces $\blacklozenge \succ \langle c_1, c_2, \sigma\rangle \longrightarrow_W^* \blacklozenge \prec \sigma'$.

Primero resolvemos 1:

Si $\blacklozenge \succ \langle c_1, c_2, \sigma\rangle \longrightarrow_W^* \blacklozenge \prec \sigma'$, entonces $\blacklozenge \succ \langle c_2, c_1, \sigma\rangle \longrightarrow_W^* \blacklozenge \prec \sigma'$. Esto se debe a la regla de la secuencia en la máquina W: cuando evaluamos una secuencia, primero evaluamos el primer comando y luego, si termina, evaluamos el segundo comando. Por lo tanto, al ejecutar $c_1; c_2$, c_1 se evalúa primero, seguido por c_2. En consecuencia, al intercambiar el orden de ejecución a $c_2;c_1$, ambos comandos se ejecutarán en el mismo estado σ', ya que la secuencia no afecta el estado.

Ahora resolvemos 2:

Si $\blacklozenge \succ \langle c_2, c_1, \sigma\rangle \longrightarrow_W^* \blacklozenge \prec \sigma'$, entonces $\blacklozenge \succ \langle c_1, c_2, \sigma\rangle \longrightarrow_W^* \blacklozenge \prec \sigma'$. De manera similar al razonamiento anterior, intercambiar el orden de ejecución de c_2 y c_1 no afecta el estado σ'. La ejecución de c_2 seguido por c_1 o c_1 seguido por c_2 resultará en el mismo estado final σ'.

Por lo tanto, se cumple que $c_1;\ c_2 \equiv_w c_2;\ c_1$ en el lenguaje While.

d) $c_1;\ (c_2;\ c_3) \equiv_w (c_1;\ c_2);\ c_3$

Para probar que $c_1;\ (c_2;\ c_3) \equiv_w (c_1;\ c_2);\ c_3$ en el lenguaje While, necesitamos verificar lo siguiente:

1.- Si $\blacklozenge \succ \langle c_1;(c_2;c_3),\ \sigma\rangle \longrightarrow_W^* \blacklozenge \prec \sigma'$, entonces $\blacklozenge \succ \langle (c_1;c_2);c_3,\ \sigma\rangle \longrightarrow_W^* \blacklozenge \prec \sigma'$.
2.- Si $\blacklozenge \succ \langle (c_1;c_2);c_3,\ \sigma\rangle \longrightarrow_W^* \blacklozenge \prec \sigma'$, entonces $\blacklozenge \succ \langle c_1;(c_2;c_3),\ \sigma\rangle \longrightarrow_W^* \blacklozenge \prec \sigma'$.

Primero resolvemos 1:

Si $\blacklozenge \succ \langle c_1;(c_2;c_3),\ \sigma\rangle \longrightarrow_W^* \blacklozenge \prec \sigma'$, entonces $\blacklozenge \succ \langle (c_1;c_2);c_3,\ \sigma\rangle \longrightarrow_W^* \blacklozenge \prec \sigma'$. Esta relación se mantiene debido a la asociatividad de la secuencia en la máquina W. Independientemente de cómo se agrupen los comandos dentro de una secuencia, si todos se ejecutan en secuencia, el estado final será el mismo. Por lo tanto, ejecutar primero c_1 y luego c_2 seguido por c_3 es equivalente a ejecutar primero c_1 seguido por c_2 y luego c_3.

Ahora resolvemos 2:

Si $\blacklozenge \succ \langle (c_1;c_2);c_3,\ \sigma\rangle \longrightarrow_W^* \blacklozenge \prec \sigma'$, entonces $\blacklozenge \succ \langle c_1;(c_2;c_3),\ \sigma\rangle \longrightarrow_W^* \blacklozenge \prec \sigma'$. Esta relación también se mantiene debido a la asociatividad de la secuencia en la máquina W. Ejecutar c_1 seguido por c_2 y luego c_3 es equivalente a agrupar primero c_2 y c_3 ejecutarlos juntos, seguidos por c_1.

Por lo tanto, dado que ambas partes se sostienen, se demuestra que $c_1;\ (c_2;\ c_3) \equiv_w (c_1;\ c_2);\ c_3$ en el lenguaje While.