



UNIVERSIDAD 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 $\sigma(x) = 17$ y $\sigma(y) = 5$ ¿Cual es el estado resultante de la evaluación?

Inicialmente, tenemos $\sigma(x) = 17$ y $\sigma(y) = 5$. 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:

- $y \nless x + 1$ es verdadero ya que $5 \nless 7 + 1$
- $z := z + 1$; $z=3$
- $x := x - y$; $x=2$

Iteración 4:

- $y \nless x + 1$ es verdadero ya que $5 \nless 2 + 1$
- $z := z + 1$; $z=4$
- $x := x - y$; $x=-3$

Iteración 5:

- y $x + 1$ es falso ya que $5 \neq -3 + 1$

El estado resultante es $\sigma(x) = -3$ y $\sigma(y) = 5$.

- b) Da un estado σ tal que si se evalúa el programa anterior con dicho estado la evaluación se ciclaría 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 $\sigma(x) = 5$ y $\sigma(y) = 10$. 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:

for $x := a1$ to $a2$ do c

esto es:

- a) Modifica la estructura de la máquina W (agregando marcos, estados o transiciones) para evaluar la expresión 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 semántica estática para verificación 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 azúcar sintáctica 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 ($c1 \equiv w c2$) si y solo si la ejecución de ambos programas resulta en el mismo estado, es decir, si para todo estado de las variables σ , $\Diamond \succ \langle c1, \sigma \rangle \rightarrow * W \Diamond \prec \sigma'$ y $\Diamond \succ \langle c2, \sigma \rangle \rightarrow * W \Diamond \prec \sigma'$ entonces $c1 \equiv w c2$.

Con la definición de equivalencia anterior, demuestra o da un contraejemplo de lo siguiente:

- a) $\equiv w$ realmente es una relación de equivalencia. Esto es, demuestra que la relación $\equiv w$ es transitiva, reflexiva y simétrica.
- b) $c; \text{skip} \equiv w c$
- c) $c1; c2 \equiv w c2; c1$
- d) $c1; (c2; c3) \equiv w (c1; c2); c3$