Universidad Nacional de Rosario
Facultad de Ciencias Exactas,
Ingeniería y Agrimensura

Análisis de Lenguajes de Programación

Trabajo Práctico I

Un Lenguaje Imperativo Simple (LIS)

Román Castellarin
Juan Ignacio Suarez

28 de septiembre de 2018

1. Operador Ternario

El *operador ternario* es una construcción del estilo de `if-then-else` que a diferencia de éste, es una expresión entera en lugar de un *statement*.
Formalmente,

1.1. Sintaxis Abstracta

(fragmento)

$$\begin{aligned} intexp ::= & \ ... \\ & \mid boolexp \ ? \ intexp \ : \ intexp \end{aligned}$$

Si la expresión booleana evalúa a verdadero, entonces la expresión completa toma el valor del segundo argumento, si no, del tercero.
Esta sintaxis en Haskell se verá reflejada así, al representarla mediante un constructor:

(fragmento)

```
data IntExp = ...
            | TerCond BoolExp IntExp IntExp
```

1.2. Sintaxis Concreta

(fragmento)

$$\begin{aligned} intexp ::= & \ ... \\ & \mid boolexp \ \texttt{'?'} \ intexp \ \texttt{':'} \ intexp \end{aligned}$$

2. Parser

(ver código adjunto)

3. Semántica operacional

3.1. Semántica natural para Expresiones

Extendemos la semántica operacional de paso grande para incluir al operador ternario...

$$\frac{\langle b, \sigma\rangle \Downarrow_{intexp} \mathbf{true} \quad \langle e_1, \sigma\rangle \Downarrow_{intexp} v}{\langle b \ ? \ e_1 : e_2, \sigma\rangle \Downarrow_{intexp} v} \ \textsc{TerCond-T}$$

$$\frac{\langle b, \sigma\rangle \Downarrow_{intexp} \mathbf{false} \quad \langle e_2, \sigma\rangle \Downarrow_{intexp} v}{\langle b \ ? \ e_1 : e_2, \sigma\rangle \Downarrow_{intexp} v} \ \textsc{TerCond-F}$$

3.2. Semántica op. estructural para Comandos

3.2.1. Prueba de que la relación de evaluación a un paso es determinista

Sean $\langle t, \sigma\rangle \rightsquigarrow \langle t_1, \sigma_1\rangle$ y $\langle t, \sigma\rangle \rightsquigarrow \langle t_2, \sigma_2\rangle$, que la relación de evaluación de un paso ($\rightsquigarrow$) sea determinista significa que $\langle t_1, \sigma_1\rangle = \langle t_2, \sigma_2\rangle$.

Para demostrar esto haremos inducción sobre la derivación $\langle t, \sigma\rangle \rightsquigarrow \langle t_1, \sigma_1\rangle$. Supondremos que tanto $\Downarrow_{intexp}$ como $\Downarrow_{boolexp}$ son deterministas.

Si la última regla utilizada es Ass, entonces t tiene la forma $v := e$ para alguna e. Por lo tanto,

- Existe un único n tal que $\langle e, \sigma\rangle \Downarrow_{intexp} n$. (Por determinismo de $\Downarrow_{intexp}$)
- $t_1 = \textbf{skip}$, $\sigma_1 = [\sigma|v : n]$.
- La última regla utilizada en la derivación $\langle t, \sigma\rangle \rightsquigarrow \langle t_2, \sigma_2\rangle$ no pudo haber sido SEQ_1, SEQ_2, IF_1, IF_2 ni $REPEAT$ por la forma de t. Entonces la última regla utilizada tuvo que haber sido ASS. Finalmente nos queda $t_2 = \textbf{skip} = t_1$ y $\sigma_2 = [\sigma|v : n] = \sigma_1$.

Si es la última regla utilizada es SEQ_1, entonces t tiene la forma $\textbf{skip}; c$ para alguna c. Por lo tanto,

- $t_1 = c$, $\sigma_1 = \sigma$.
- La última regla utilizada en la derivación $\langle t, \sigma\rangle \rightsquigarrow \langle t_2, \sigma_2\rangle$ no pudo haber sido ASS, IF_1, IF_2 ni $REPEAT$ por la forma de t. Para el caso de SEQ_2, no existe regla de derivación para **skip** al ser forma normal. Por lo tanto la última regla utilizada tuvo que haber sido la única restante, SEQ_1, finalmente entonces nos queda $t_2 = c = t_1$ y $\sigma_2 = \sigma = \sigma_1$.

Si es la última regla utilizada es SEQ_2, entonces t tiene la forma $c_0; c_1$. Por lo tanto,

- $t_1 = c_0'; c_1$, $\sigma_1 = \sigma'$.
- La última regla utilizada en la derivación $\langle t, \sigma\rangle \rightsquigarrow \langle t_2, \sigma_2\rangle$ no pudo haber sido ASS, IF_1, IF_2 ni $REPEAT$ por la forma de t.

  Sabemos por hipótesis inductiva que $\langle c_0, \sigma\rangle \rightsquigarrow \langle c_0', \sigma'\rangle$ es determinista. Luego c_0 no puede ser **skip**, en consecuencia la última regla utilizada en la derivación $\langle c_0, \sigma\rangle \rightsquigarrow \langle t_2, \sigma_2\rangle$ no pudo haber sido SEQ_1. La única opción restante es que sea SEQ_2, finalmente nos queda $t_2 = c_0'; c_1 = t_1$ y $\sigma_2 = \sigma' = \sigma_1$.

Si la última regla utilizada es IF_1, entonces t tiene la forma $\textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1$, para algunas b, c_0, c_1. Por lo tanto,

- $\langle b, \sigma\rangle \Downarrow_{boolexp} \textbf{true}$ (únicamente, por determinismo de $\Downarrow_{boolexp}$).
- $t_1 = c_0$ y $\sigma_1 = \sigma$.

- La última regla utilizada en la derivación $\langle t, \sigma\rangle \rightsquigarrow \langle t_2, \sigma_2\rangle$ no pudo haber sido ASS, SEQ_1, SEQ_2 ni $REPEAT$ por la forma de t. Tenemos además que $\langle b, \sigma\rangle \Downarrow_{boolexp}$ **true**, entonces IF_2 por su definición queda descartada como opción. La única opción restante es IF_1, finalmente nos queda $t_2 = c_0 = t_1$ y $\sigma_2 = \sigma = \sigma_1$.

Si la última regla utilizada es IF_2, la demostración es análoga al caso anterior de IF_1.

Si la última regla utilizada es $REPEAT$, entonces t tiene la forma **repeat** c **until** b, para algún par c, b. Por lo tanto,

- $t_1 = c;$ **if** b **then skip else repeat** c **until** b y $\sigma_1 = \sigma$.
- La última regla utilizada en la derivación $\langle t, \sigma\rangle \rightsquigarrow \langle t_2, \sigma_2\rangle$ no pudo haber sido ASS, SEQ_1, SEQ_2, IF_1 ni IF_2 por la forma de t. Por lo tanto, la última regla utilizada debió haber sido $REPEAT$. Finalmente entonces nos queda
  $t_2 = c;$ **if** b **then skip else repeat** c **until** $b = t_1$ y $\sigma_2 = \sigma = \sigma_1$.

3.2.2. Ejemplo

Vale que

$$\langle x := x + 1;\ \textbf{if } x > 0 \textbf{ then skip else } x := x - 1, [\sigma|x : 0]\rangle \rightsquigarrow^*, \langle \textbf{skip}, [\sigma|x : 1]\rangle$$

Demostración:
Para alivianar la cantidad de texto en el árbol de prueba, definamos:

$t_1 = \langle x := x + 1;\ \textbf{if } x > 0 \textbf{ then skip else } x := x - 1, [\sigma|x : 0]\rangle$
$t_2 = \langle \textbf{skip};\ \textbf{if } x > 0 \textbf{ then skip else } x := x - 1, [\sigma|x : 1]\rangle$
$t_3 = \langle\ \textbf{if } x > 0 \textbf{ then skip else } x := x - 1, [\sigma|x : 1]\rangle$
$t_4 = \langle \textbf{skip}, [\sigma|x : 1]\rangle$

Luego resulta,

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\langle x, [\sigma|x : 0]\rangle \Downarrow_{intexp} 0}\ \textsc{Var} \quad \dfrac{}{\langle 1, [\sigma|x : 0]\rangle \Downarrow_{intexp} 1}\ \textsc{Nval}}{\langle x + 1, [\sigma|x : 0]\rangle \Downarrow_{intexp} 1}\ \textsc{Plus}}{\langle x := x + 1, [\sigma|x : 0]\rangle \rightsquigarrow \langle \textbf{skip}, [\sigma|x : 1]\rangle}\ \textsc{Ass}}{t_1 \rightsquigarrow t_2}\ \textsc{Seq}_2$$

Adicionalmente:

$$\dfrac{}{t_2 \rightsquigarrow t_3}\ \textsc{Seq}_1$$

Luego vemos:

$$\dfrac{\dfrac{\overline{\langle x, [\sigma|x:1]\rangle \Downarrow_{intexp} 1}\ \textsc{Var} \quad \overline{\langle 0, [\sigma|x:1]\rangle \Downarrow_{intexp} 0}\ \textsc{Nval}}{\langle x > 0, [\sigma|x:1]\rangle \Downarrow_{boolexp} \mathbf{true}}\ \textsc{Gt}}{t_3 \rightsquigarrow t_4}\ \textsc{If}_1$$

Por lo que ya probamos $t_1 \rightsquigarrow t_2 \rightsquigarrow t_3 \rightsquigarrow t_4$.
Recordando que $\rightsquigarrow^*$ es la clausura transitiva de $\rightsquigarrow$, tenemos:

$$\dfrac{\dfrac{\dfrac{t_1 \rightsquigarrow t_2}{t_1 \rightsquigarrow^* t_2} \quad \dfrac{t_2 \rightsquigarrow t_3}{t_2 \rightsquigarrow^* t_3}}{t_1 \rightsquigarrow^* t_3} \quad \dfrac{t_3 \rightsquigarrow t_4}{t_3 \rightsquigarrow^* t_4}}{t_1 \rightsquigarrow^* t_4}$$

Que es lo que queríamos demostrar.

4. Bucle while

La instrucción `while` es similar a aquella de `repeat`.
El efecto que produce **while** b **do** c es el de ejecutar el comando c mientras la condición b se cumpla, pero primero se evalúa la condición, y solo si esta da verdadera, se ejecutara el .

4.1. Sintaxis Abstracta

(fragmento)

$$\begin{aligned} comm ::= & \ ... \\ & \mid \mathbf{while}\ boolexp\ comm \end{aligned}$$

4.2. Semántica op. estructural para Comandos

$$\dfrac{}{\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma\rangle \rightsquigarrow \langle \mathbf{if}\ b\ \mathbf{then}\ c; \mathbf{while}\ b\ \mathbf{do}\ c\ \mathbf{else\ skip}, \sigma\rangle}\ \textsc{While}$$