Trabajo Práctico 1

Valentina Prato - Daniela Bazan

22 de Septiembre 2021

Análisis de Lenguajes de Programación
Licenciatura en Ciencias de la Computación
Universidad Nacional de Rosario
Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Ejercicio 1

Extendemos las sintaxis abstracta y concreta de LIS para incluir asignaciones de variables como expresiones enteras y el operador ',' para escribir una secuencia de expresiones enteras. Además modificamos la gramática para obtener una sin ambigüedades, suponiendo la siguiente precedencia para los operadores:

$$-_u \; (*/) \; (+-_b) \; (== \; ! = \; < \; >) \; ! \;\; \& \;\; \& \;\; \| = ;$$

Sintaxis Abstracta

intexp := eassgn | eassgn, intexp
eassgn := var = eassgn | expr
expr := term + intexp | term $-_b$ intexp | term
term := factor | factor * term | factor / term
factor := -$_u$ factor | nat | var | (intexp)

boolexp := bool1 | bool1 ∨ boolexp
bool1 := bool2 | bool2 ∧ bool1
bool2 := bool3 | ¬ bool3
bool3 := **true** | **false**
  | (boolexp)
  | intexp == intexp
  | intexp != intexp
  | intexp < intexp
  | intexp > intexp

comm := **skip**
  | var = intexp
  | comm; comm
  | **if** boolexp **then** comm **else** comm
  | **repeat** comm **until** boolexp

Sintaxis Concreta

digit := '0' | '1' | ... | '9'
letter := 'a' | ... | 'z'
nat := digit | digit nat
var := letter | letter var

intexp := eassgn | eassgn ',' intexp
eassgn := var '=' eassgn | expr
expr := term '+' intexp | term '-' intexp | term
term := factor | factor '*' term | factor '/' termn

factor := '-' factor | nat | var | '(' intexp ')'

boolexp := bool1 | bool1 ' & & ' boolexp
bool1 := bool2 | bool2 '||' bool1
bool2 := bool3 | '!' bool3
bool3 := '**true**' | '**false**'
| '(' boolexp ')'
| intexp '==' intexp
| intexp '!=' intexp
| intexp '<' intexp
| intexp '>' intexp

comm := **skip**
| var '=' intexp
| comm ';' comm
| 'if' boolexp '{' comm '}'
| 'if' boolexp '{' comm '}' 'else' '{' comm '}'
| 'repeat' comm 'until' boolexp 'end'

Por la regla gramática de **repeat**, que no tiene los brackets que hay en los comandos **if**, editamos los archivos de ejemplo **div.lis** y **sqrt.lis** eliminando los brackets para que cumplan con la regla como la indica el enunciado. Para evaluarlos correctamente y que nuestra implementación del parser no devuelva error recomendamos hacer lo mismo.

Ejercicio 2

Extendemos la realización de la sintaxis abstracta en Haskell para incluir la asignación como expresiones y el operador ',' para secuencia de expresiones enteras. Los nombres de los constructores son ESeq y EAssgn respectivamente. Para esto modificamos el archivo src\AST.hs incluyendo los constructores mencionados:

```haskell
ESeq :: Exp Int -> Exp Int -> Exp Int
EAssgn :: Variable -> Exp Int -> Exp Int
```

Ejercicio 3

Ver resolución en **src/Parser.hs**.

Ejercicio 4

Modificamos la semántica big-step de expresiones enteras para incluir la asignación de variables como expresiones y el operador ',' para secuencias de expresiones.

$$\frac{\langle e_0, \sigma\rangle \Downarrow_{exp} \langle n_0, \sigma'\rangle \quad \langle e_1, \sigma'\rangle \Downarrow_{exp} \langle n_1, \sigma''\rangle}{\langle e_0, e_1, \sigma\rangle \Downarrow_{exp} \langle n_1, \sigma''\rangle}\,(ESeq)$$

$$\frac{\langle e, \sigma\rangle \Downarrow_{exp} \langle n, \sigma'\rangle}{\langle v := e, \sigma\rangle \Downarrow_{exp} \langle n, [\sigma'|v : n]\rangle}\,(EAssgn)$$

Ejercicio 5

Vamos a demostrar que la relación de evaluación de un paso $\rightsquigarrow$ es determinista.
Queremos demostrar que:
Si t $\rightsquigarrow$ t' y t $\rightsquigarrow$ t'', entonces t' = t''.
Tenemos como hipótesis que la relación $\Downarrow$ es determinista.

Demostración

Para probarlo usamos **inducción sobre la derivación** $\rightsquigarrow$. Sean t, t' y t'' tal que t $\rightsquigarrow$ t' y t $\rightsquigarrow$ t'', tomamos como **hipótesis inductiva** que todas las transiciones $\rightsquigarrow$ sobre sub-expresiones de t son deterministas.

- Si la última regla utilizada es **Ass**, entonces t = $\langle v = e, \sigma\rangle$. Al usar la regla **Ass**, quiere decir que parte de una transición $\langle e, \sigma\rangle \Downarrow_{exp} \langle \mathbf{n}, \sigma'\rangle$ para algún valor n y estado σ'. Y como por hipótesis $\Downarrow$ es determinista, significa que $\langle \mathbf{n}, \sigma'\rangle$ es único. Por lo tanto existe un único $t' = \langle \mathbf{skip}, [\sigma'|v : n]\rangle$ tal que $t \rightsquigarrow t'$. Es decir, para todo t'' tal que t $\rightsquigarrow$ t'' con última regla **Ass**, t' == t''.

- Si la última regla utilizada es **SEQ1**, entonces t = $\langle \mathbf{skip}; c_1, \sigma\rangle$ y t' = $\langle c_1, \sigma\rangle$. Por lo tanto t' es la única expresión tal que $t \rightsquigarrow t'$, y entonces para todo t'' tal que t $\rightsquigarrow$ t'' con última regla **SEQ1**, t' == t''.

- Si la última regla utilizada es **SEQ2**, entonces t = $\langle c_0; c_1, \sigma\rangle$. Al usar la regla **SEQ2**, quiere decir que parte de una transición $\langle c_0, \sigma\rangle \Downarrow_{exp} \langle c_0', \sigma'\rangle$ para alguna expresión c_0' y estado σ'. Y como c_0 es sub-expresión de t, por **hipótesis inductiva** toda transición $\rightsquigarrow$ sobre c_0 es determinista. Es decir, $\langle c_0', \sigma'\rangle$ es único y por lo tanto hay un único $t' = \langle c_0'; c_1, \sigma'\rangle$ tal que $t \rightsquigarrow t'$. De esta forma, para todo t'' tal que $t \rightsquigarrow t''$ con última regla **SEQ2**,
t' == t''.

- Si la última regla utilizada es **IF1**, entonces t = $\langle$**if** b **then** c_0 **else** $c_1, \sigma\rangle$. Al usar la regla **IF1**, quiere decir que parte de una derivación $\langle b, \sigma\rangle \Downarrow_{exp} \langle \mathbf{true}, \sigma'\rangle$. Como por **hipótesis inductiva** toda transición $\rightsquigarrow$ sobre b es determinista, entonces el estado σ' es único. Así que hay un único $t' = \langle c_0, \sigma'\rangle$ tal que $t \rightsquigarrow t'$. Por lo tanto, para todo t'' tal que $t \rightsquigarrow t''$ con última regla **IF2**, t' == t''.

- Si la última regla utilizada es **IF2**, entonces t = $\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma\rangle$. Al usar la regla **IF2**, quiere decir que parte de una derivación $\langle b, \sigma\rangle \Downarrow_{exp} \langle \mathbf{false}, \sigma'\rangle$.

Como por **hipótesis inductiva** toda transición $\rightsquigarrow$ sobre b es determinista, entonces el estado σ' es único. Así que hay un único $t' = \langle c_1, \sigma' \rangle$ tal que $t \rightsquigarrow t'$. Por lo tanto, para todo t” tal que $t \rightsquigarrow t''$ con última regla **IF2**, t’ == t”.

- Si la útima regla utilizada es **REPEAT**, entonces t = ⟨**repeat** c **until** b, σ⟩ y t’ = ⟨c; **if** b **then skip else repeat** b **until** c, σ⟩. Por lo tanto t’ es la única expresión tal que $t \rightsquigarrow t'$, y entonces para todo t” tal que t $\rightsquigarrow$ t” con última regla **REPEAT**, t’ == t”.

Finalmente, como la propiedad vale para todas las reglas definidas, podemos concluir que para toda expresión t, t’ y t”, si t $\rightsquigarrow$ t’ y t $\rightsquigarrow$ t”, entonces t’ == t”. Es decir, la relación de transición $\rightsquigarrow$ es determinista.

Ejercicio 6

Utilizando las reglas de inferencia, construimos un árbol de derivación para probar que el siguiente juicio es válido:
$\langle x = y = 1; \textbf{repeat}\ \text{x = x - y}\ \textbf{until}\ \text{x == 0}, [[\sigma \mid x : 2] \mid y : 2]\rangle \rightsquigarrow *\langle \textbf{skip}, [[\sigma \mid x : 0] \mid y : 1]\rangle$

Primer Paso:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\langle 1, [[\sigma \mid x : 2] \mid y : 2]\rangle \Downarrow_{Exp} \langle \mathbf{1}, [[\sigma \mid x : 2] \mid y : 2]\rangle}\,\text{NVal}}{\langle y = 1, [[\sigma \mid x : 2] \mid y : 2]\rangle \Downarrow_{Exp} \langle \mathbf{1}, [[\sigma \mid x : 2] \mid y : 1]\rangle}\,\text{EAssgn}}{\langle x = y = 1, [[\sigma \mid x : 2] \mid y : 2]\rangle \rightsquigarrow \langle \textbf{skip}, [[\sigma \mid x : 1] \mid y : 1]\rangle}\,\text{Ass}}{\langle x = y = 1; \underbrace{\textbf{repeat}\ x = x - y\ \textbf{until}\ x == 0}_{c_1}, [[\sigma \mid x : 2] \mid y : 2]\rangle \rightsquigarrow \langle \textbf{skip}, c_1, [[\sigma \mid x : 1] \mid y : 1]\rangle}\,\text{SEQ2}$$

Segundo Paso:

$$\dfrac{}{\langle \textbf{skip}, \textbf{repeat}\ x = x - y\ \textbf{until}\ x == 0, [[\sigma \mid x : 1] \mid y : 1]\rangle \rightsquigarrow \langle \textbf{repeat}\ x = x - y\ \textbf{until}\ x == 0, [[\sigma \mid x : 1] \mid y : 1]\rangle}\,\text{SEQ1}$$

Tercer Paso:

$$\dfrac{}{\langle \underbrace{\textbf{repeat}\ x = x - y\ \textbf{until}\ x == 0}_{c_1}, [[\sigma \mid x : 1] \mid y : 1]\rangle \rightsquigarrow \langle x = x - y;\ \textbf{if}\ x == 0\ \textbf{then skip else}\ c_1, [[\sigma \mid x : 1] \mid y : 1]\rangle}\,\text{repeat}$$

Cuarto Paso:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\langle x, [[\sigma \mid x : 1] \mid y : 1]\rangle \Downarrow_{Exp} \langle \mathbf{1}, [[\sigma \mid x : 1] \mid y : 1]\rangle}\,\text{Var} \quad \dfrac{}{\langle y, [[\sigma \mid x : 1] \mid y : 1]\rangle \Downarrow_{Exp} \langle \mathbf{1}, [[\sigma \mid x : 1] \mid y : 1]\rangle}\,\text{Var}}{\langle x - y, [[\sigma \mid x : 1] \mid y : 1]\rangle \Downarrow_{Exp} \langle 0, [[\sigma \mid x : 1] \mid y : 1]\rangle}\,-_b}{\langle x = x - y, [[\sigma \mid x : 1] \mid y : 1]\rangle \rightsquigarrow \langle \textbf{skip}, [[\sigma \mid x : 0] \mid y : 1]\rangle}\,\text{Ass}}{\langle x = x - y;\ \underbrace{\textbf{if}\ x == 0\ \textbf{then skip else repeat}\ x = x - y\ \textbf{until}\ x == 0}_{c_2}, [[\sigma \mid x : 1] \mid y : 1]\rangle \rightsquigarrow \langle \textbf{skip}; c_2, [[\sigma \mid x : 0] \mid y : 1]}\,\text{SEQ2}$$

Quinto Paso:

$$\dfrac{}{\langle \textbf{skip}; \textbf{if } x == 0 \textbf{ then skip else } \underbrace{\textbf{repeat } x = x - y \textbf{ until } x == 0}_{c_1}, [[\sigma \mid x : 0] \mid y : 1] \rightsquigarrow \langle \textbf{if } x == 0 \textbf{ then skip else } c_1, [[\sigma \mid x : 0] \mid y : 1]}\ \text{SEQ1}$$

Sexto Paso:

$$\dfrac{\dfrac{\dfrac{}{\langle x, [[\sigma \mid x : 0] \mid y : 1]\rangle \Downarrow_{Exp} \langle 0, [[\sigma \mid x : 0] \mid y : 1]\rangle}\ \text{Var} \qquad \dfrac{}{\langle 0, [[\sigma \mid x : 0] \mid y : 1]\rangle \Downarrow_{Exp} \langle \mathbf{0}, [[\sigma \mid x : 0] \mid y : 1]\rangle}\ \text{NVal}}{\langle x == 0, [[\sigma \mid x : 0] \mid y : 1]\rangle \Downarrow_{Exp} \langle \textbf{true}, [[\sigma \mid x : 0] \mid y : 1]\rangle}\ \text{EQ}}{\langle \textbf{if } x == 0 \textbf{ then skip else repeat } x = x - y \textbf{ until } x == 0, [[\sigma \mid x : 0] \mid y : 1]\rangle \rightsquigarrow \langle \textbf{skip}, [[\sigma \mid x : 0] \mid y : 1]\rangle}\ \text{IF1}$$

Ejercicios 7, 8 y 9

Ver resolución en **src/Eval1.hs**, **src/Eval2.hs** y **src/Eval3.hs**.

Ejercicio 10

Agregamos una regla de producción a la gramática abstracta de LIS y extendemos la semántica operacional de comandos para el comando for.

Sintaxis Abstracta

intexp := expr | expr,intexp
expr := eassgn | term + intexp | term - intexp | term
eassgn := var = expr
term := factor | factor * term | factor / termn
factor := $-_u$ factor | nat | var | (intexp)

boolexp := bool1 | bool1 ∨ boolexp
bool1 := bool2 | bool2 ∧ bool1
bool2 := bool3 | ¬ bool3
bool3 := **true** | **false**
  | (boolexp)
  | intexp == intexp
  | intexp != intexp
  | intexp < intexp
  | intexp > intexp

comm := **skip**
  | var = intexp
  | comm,comm
  | **if** boolexp **then** comm **else** comm
  | **repeat** comm **until** boolexp
  | **for** (intexp;boolexp;intexp) comm

Semántica Operacional

$$\frac{}{\langle \textbf{for}\ (e_1; e_2; e_3)\ c, \sigma\rangle \rightsquigarrow \langle e_1; \textbf{if}\ e_2\ \textbf{then repeat}\ c; e_3\ \textbf{until}\ \neg e_2\ \textbf{else skip}, \sigma\rangle}\ \text{FOR}$$