Lenguajes de Programación 2020-I
Nota de clase 2, Ejemplo introductorio

Favio E. Miranda Perea Lourdes Del Carmen González Huesca
Facultad de Ciencias UNAM

19 de agosto de 2019

Desarrollamos aquí un ejemplo introductorio del proceso de evaluación de un lenguaje de expresiones aritméticas y booleanas, así como su compilación a una máquina de pila que funciona mediante instrucciones de bajo nivel.

1. Un lenguaje para expresiones aritméticas y booleanas

A continuación presentamos un lenguaje para expresiones aritméticas y booleanas que incluye las operaciones de suma, sucesor, predecesor, el orden, un test para cero y la negación booleana.

Las expresiones del lenguaje se definen como sigue:

$$\begin{aligned} e \quad ::= \quad & x \mid n \mid \mathsf{true} \mid \mathsf{false} \mid \\ & e + e \mid e * e \\ & e < e \mid \mathsf{iszero}\, e \mid \mathsf{not}\, e \end{aligned}$$

Ejemplos de expresiones del lenguaje son:

- $3 + y$
- $\mathsf{not}(\mathsf{iszero}\, 9)$
- $\mathsf{iszero}\, 3 < \mathsf{not}(2 + 4)$
- $\mathsf{not}\, \mathsf{false}$
- $\mathsf{not}(x + 1)$
- $\mathsf{iszero}\, \mathsf{false}$
- $\mathsf{not}\, 7 * y$
- $7 < x + y$

Se observa que todas son expresiones sintácticamente correctas pero algunas de ellas no tienen sentido desde el punto de vista semántico, es decir, no es posible asignarles uno de los tipos posibles para el lenguaje que son Nat o Bool. Por ejemplo $\mathsf{not}(8+2)$ o $\mathsf{iszero}\,\mathsf{false}$. Además hay expresiones para las cuales no es claro si son válidas o no, por ejemplo $x+2$ puede o no ser válida dependiendo si x representa a un número o a un booleano, lo mismo sucede con $\mathsf{not}(\mathsf{iszero}\,x)$ o con $x < y+1$.

1.1. Intérprete

Un intérprete para el lenguaje es simplemente una función que devuelve un natural o un booleano que resulta de evaluar una expresión e. La definición recursiva de esta función

$$\texttt{eval} : Exp \to \mathsf{N} \cup \mathbb{B}$$

donde $\mathsf{N} = \{0, 1, \ldots\}$ y $\mathbb{B} = \{t, f\}$ es:

```
eval n = n
eval true = t
eval false = f
eval (e1 + e2) = eval e1 + eval e2
eval (e1 * e2) = eval e1 * eval e2
eval (e1 < e2) = (eval e1) < (eval e2)
eval (not e) = ¬ (eval e)
eval (iszero e) = isz (eval e)
```

donde los símbolos del lado derecho $+, *, <$ denotan las operaciones usuales mientras que del lado izquierdo son únicamente símbolos que forman parte de las expresiones. Omitímos aquí el uso de variables por simplicidad. Se observa que la función así definida resulta una función parcial puesto que la evaluación de expresiones como $(3 * 5) + \mathsf{not}\,\mathsf{false}$ no tiene sentido puesto que no sabemos sumar $15 + \mathsf{true}$.

1.2. Compilación de expresiones aritméticas y booleanas

Presentamos un ejemplo de compilación del lenguaje de expresiones aritméticas y booleanas a una máquina abstracta de pila.

- Especificación del lenguaje objeto: se trata de un lenguaje de instrucciones primitivas

  $$i ::= n \mid t \mid f \mid + \mid * \mid < \mid \mathsf{not} \mid \mathsf{iszero}$$

  de manera que una instrucción es un valor natural o booleano o bien un operador del lenguaje fuente de expresiones aritméticas y booleanas

- Programas: un programa es una lista[1] de instrucciones $p = [i_1, \ldots, i_n]$

- Memoria: la memoria es una lista de valores $s = [v_1, \ldots, v_n]$ donde $v_i \in \{n, t, f\}$

[1] En realidad es una pila y elegimos implementarla como una lista, lo mismo sucede con la memoria

- Ejecución de una instrucción: la función de ejecución de una instrucción `ej` recibe una instrucción `i` y una memoria `s` devolviendo la memoria resultante al ejecutar la instrucción.

```
ej n s = (n:s)
ej true s = (t:s)
ej false s = (f:s)
ej + (v1:v2:s) = (v2+v1) : s
ej * (v1:v2:s) = (v2*v1) : s
ej < (v1:v2:s) = (v2<v1) : s
ej not (v:s) = (¬ v) : s
ej iszero (v:s) = (isz v) : s
```

  donde del lado derecho de las ecuaciones nos referimos a las operaciones binarias $+, * :: \mathsf{N} \times \mathsf{N} \to \mathsf{N}$, $<: \mathsf{N} \times \mathsf{N} \to \mathbb{B}$ y a las operaciones unarias $\neg : \mathbb{B} \to \mathbb{B}$, $isz : \mathsf{N} \to \mathbb{B}$

- Ejecución de un programa: ejecutar un programa p en la memoria dada s devuelve la memoria obtenida al ejecutar en orden todas instrucciones de p.

```
ejp nil s = s
ejp (i:p) s = ejp p (ej i s)
```

- Compilación de una expresión en un programa: el proceso de compilación convierte una expresión e en un programa.

```
comp n = [n]
comp true = [t]
comp false = [f]
comp (e1+e2) = comp e2 ++ comp e1 ++ [+]
comp (e1*e2) = comp e2 ++ comp e1 ++ [*]
comp (e1<e2) = comp e2 ++ comp e1 ++ [<]
comp (not e) = comp e ++ [not]
comp (iszero e) = comp e ++ [iszero]
```

  donde $+\!\!+$ es la función de concatenación de listas. ¿ Porqué en los casos de operadores binarios queda el programa resultado de compilar la segunda expresión e_2 al inicio de la lista?

- Correctud del compilador: la memoria resultado de ejecutar el programa p obtenido al compilar la expresión e con la memoria vacía coincide con la memoria cuyo único valor es la evaluación de la expresión e

```
∀ e (ejp (comp e) nil = [eval e])
```

Por supuesto en todas las definiciones anteriores estamos suponiendo que las expresiones de entrada son coherentes en el sentido de que cualquier operación estará bien definida. Es decir, estamos suponiendo que el proceso de verificación previa fue hecho por el sistema de tipos. Por lo que el cuantificador $\forall e$ se refiere únicamente a todas las expresiones semánticamente correctas.

1.3. Prueba de la correctud (informal)

Para probar la correctud del compilador vamos a probar algo más general:

$$\forall\ \texttt{e}\ \forall\ \texttt{p}\ \forall\ \texttt{s}\ \big(\texttt{ejp (comp e } +\!\!+\ \texttt{p) s = ejp p (eval e:s)}\big)$$

de esta propiedad la correctud del compilador resulta un corolario tomando $p = nil, s = nil$.

La prueba es por inducción sobre las expresiones. Analizamos aquí un caso base y dos casos inductivos:

- Base $e = n$: P.D. $\forall p \forall s(\texttt{ejp}\ (\texttt{comp}\, n +\!\!+ p)\ s = \texttt{ejp}\ p\ (\texttt{eval}\, n : s))$.

$$\begin{array}{rcl} \texttt{ejp}\ (\texttt{comp}\, n +\!\!+ p)\ s & = & \texttt{ejp}\ ([n] +\!\!+ p)\ s \\ & = & \texttt{ejp}\ (n : p)\ s \\ & = & \texttt{ejp}\ p\ (\texttt{ej}\, n\ s) \\ & = & \texttt{ejp}\ p\ (n : s) \\ & = & \texttt{ejp}\ p\ (\texttt{eval}\, n : s) \end{array}$$

- Paso inductivo $e = e_1 * e_2$.
  - I.H.1.: $\forall p \forall s(\texttt{ejp}\ (\texttt{comp}\, e_1 +\!\!+ p)\ s = \texttt{ejp}\ p\ (\texttt{eval}\, e_1 : s))$
  - I.H.2.: $\forall p \forall s(\texttt{ejp}\ (\texttt{comp}\, e_2 +\!\!+ p)\ s = \texttt{ejp}\ p\ (\texttt{eval}\, e_2 : s))$

  Queremos demostrar que:

$$\forall p \forall s(\texttt{ejp}\ (\texttt{comp}(e_1 * e_2) +\!\!+ p)\ s = \texttt{ejp}\ p\ (\texttt{eval}(e_1 * e_2) : s))$$

$$\begin{array}{rcl} \texttt{ejp}\ (\texttt{comp}(e_1 * e_2) +\!\!+ p)\ s & = & \texttt{ejp}\ ((\texttt{comp}\, e_2 +\!\!+ \texttt{comp}\, e_1 +\!\!+ [*]) +\!\!+ p)\ s \\ & = & \texttt{ejp}\ ((\texttt{comp}\, e_2 +\!\!+ (\texttt{comp}\, e_1 +\!\!+ [*] +\!\!+ p))\ s \\ & =_{Asoc +\!\!+} & \texttt{ejp}\ ((\texttt{comp}\, e_2 +\!\!+ (\texttt{comp}\, e_1 +\!\!+ [*] +\!\!+ p))\ s \\ & =_{I.H.2} & \texttt{ejp}\ (\texttt{comp}\, e_1 +\!\!+ [*] +\!\!+ p))\ (\texttt{eval}\, e_2 : s) \\ & =_{Asoc +\!\!+} & \texttt{ejp}\ (\texttt{comp}\, e_1 +\!\!+ ([*] +\!\!+ p))\ (\texttt{eval}\, e_2 : s) \\ & =_{I.H.1} & \texttt{ejp}\ ([*] +\!\!+ p)\ (\texttt{eval}\, e_1 : \texttt{eval}\, e_2 : s) \\ & = & \texttt{ejp}\ (* : p)\ (\texttt{eval}\, e_1 : \texttt{eval}\, e_2 : s) \\ & = & \texttt{ejp}\ p\ (\texttt{ej}\ [*]\ (\texttt{eval}\, e_1 : \texttt{eval}\, e_2 : s)) \\ & = & \texttt{ejp}\ p\ (\texttt{eval}\, e_1 * \texttt{eval}\, e_2 : s) \\ & = & \texttt{ejp}\ p\ (\texttt{eval}(e_1 * e_2) : s) \end{array}$$

- Paso inductivo $e = \texttt{not}\, e_1$.

- I.H: $\forall p \forall s(\texttt{ejp}\ (\texttt{comp}\, e_1 +\!\!+ p)\ s = \texttt{ejp}\ \ p\ (\texttt{eval}\, e_1 : s))$

Queremos demostrar que:

$$\forall p \forall s(\texttt{ejp}\ (\texttt{comp}(\texttt{not}\, e_1) +\!\!+ p)\ s = \texttt{ejp}\ \ p\ (\texttt{eval}(\texttt{not}\, e_1) : s))$$

$$\begin{array}{lll}
\texttt{ejp}\ (\texttt{comp}(\texttt{not}\, e_1) +\!\!+ p)\ s & = & \texttt{ejp}\ ((\texttt{comp}\, e_1 +\!\!+ [\neg]) +\!\!+ p)\ s \\
& =_{Asoc +\!\!+} & \texttt{ejp}\ ((\texttt{comp}\, e_1 +\!\!+ ([\neg] +\!\!+ p))\ s \\
& =_{I.H} & \texttt{ejp}\ ([\neg] +\!\!+ p)\ (\texttt{eval}\, e_1 : s) \\
& = & \texttt{ejp}\ (\neg : p)\ (\texttt{eval}\, e_1 : s) \\
& = & \texttt{ejp}\ \ p\ (\texttt{ej}\ [\neg]\ (\texttt{eval}\, e_1 : s)) \\
& = & \texttt{ejp}\ \ p\ (\neg(\texttt{eval}\, e_1) : s) \\
& = & \texttt{ejp}\ \ p\ (\texttt{eval}(\texttt{not}\, e_1) : s)
\end{array}$$

A continuación damos una idea de la formalización de esta prueba en el sistema de deducción natural.