Lenguajes de Programación, 2019-2
Nota de laboratorio 5: Análisis semántico

Manuel Soto Romero

4 de marzo de 2019

Facultad de Ciencias, UNAM

Una vez implementado el análisis léxico y sintáctico, el análisis semántico consiste en recorrer el árbol de sintaxis abstracta dándole un significado, es decir, evaluando las expresiones escritas por el programador. En esta nota se revisa la implementación de la función `interp` para evaluar expresiones del lenguaje `LDP`.

1. Análisis semántico de `LDP`

A continuación se revisa el comportamiento de cada una de las primitivas del lenguaje.

Identificadores Interpretar un identificador resulta en un error, pues no tienen un valor asociado por sí mismos, es decir, es una presencia libre.

```
(λ) a
Free identifier
```

Números Los números se interpretan a sí mismos, al ser una primitiva, no hay más procesos que realizar.

```
(λ) 1729
1729
```

Operaciones binarias Debe interpretarse el lado izquierdo, interpretarse el lado derecho y finalmente aplicar la función correspondiente.

```
(λ) {+ {* 2 3} {/ 10 2}}
(λ) {+ 6 {/ 10 2}}
(λ) {+ 6 5}
11
```

Asignaciones locales Para interpretar una asignación local, debe *sustituirse* el cada presencia del identificador en el cuerpo por el valor que tiene asociado e interpretar el cuerpo después de realizar la sustitución.

```
(λ) {with {a 3} {+ {* a 2} 4}}
(λ) {+ {* 3 2} 4}
(λ) {+ 6 4}
10
```

2. Sustitución textual

Para la interpretación de asignaciones locales with, es necesario contar con un algoritmo de sustitución textual. La notación `expr[sub-id:=val]` representa la sustitución textual de cada presencia de `sub-id` por `val` en `expr`. A continuación se presenta la definición recursiva del algoritmo de sustitución textual:

Sea `id` un identificador, `n` un número cualquiera, $\circledast$ un operador binario y `a,b,value,body` expresiones de LDP:

1. `id[id:=val] = valida`
2. `id[sub-id:=val] = id si id` $\neq$ `sub-id`
3. `n[sub-id:=val] = n`
4. `{`$\circledast$ `a b}[sub-id:=val] = {`$\circledast$ `a[sub-id:=val] b[sub-id:=val]}`
5. `{with {id value} body} =`
   `{with {id value[sub-id:=val]} body} si id = sub-id`
6. `{with {id value} body} =`
   `{with {id value[sub-id:=val]} body[sub-id:=val]} si id` $\neq$ `sub-id`

El Listado de código 1 muestra la implementación de esta función usando el tipo de dato `WAE` de la Nota de laboratorio 4.

```
1 ;; Algoritmo de sustitución textual.
2 ;; subst: WAE symbol WAE -> WAE
3 (define (subst expr sub-id val)
4    (match expr
5       [(id i) (if (symbol=? i sub-id)
6                   val
7                   expr)]
8       [(num n) expr]
```

```
9       [(binop f izq der)
10         (binop f (subst izq sub-id val) (subst der sub-id val))]
11      [(with id value body)
12         (if (symbol=? id sub-id)
13             (with id (subst value sub-id val) body)
14             (with id
15                   (subst value sub-id val)
16                   (subst body sub-id val))]))
```

Listado de código 1: Implementación del algoritmo de sustitución textual

3. Un intérprete sencillo

A continuación se muestra la implementación del analizador sintáctico para el lenguaje LDP mediante la función `interp`. Cada uno de los patrones reconocidos (líneas 5 a 8) corresponde con los casos definidos en la Sección 1.

```
1  ;; Interpretación de expresiones de WAE.
2  ;; interp: WAE -> number
3  (define (interp expr)
4     (match expr
5        [(id i) (error 'interp "Free identifier")]
6        [(num n) n]
7        [(binop f izq der) (f (interp izq) (interp der))]
8        [(with id value body)
9           (interp (subst body id value))]))
```

Listado de código 2: Función interp