Lenguajes de Programación

Ejercicios propuestos

Karla Ramírez Pulido Manuel Soto Romero Alejandro Hernández Mora
Silvia Díaz Gómez Pedro Ulises Cervantes González

Semestre 2021-2
Facultad de Ciencias, UNAM

1. Lenguaje de programación WAE

1.1. Gramática

Considera la siguiente gramática del lenguaje de expresiones aritméticas con variables **WAE** (With Arithmetic Expression Language)

```
<expr> ::= <id>
         | <num>
         | {<op> <expr>+}
         | {with {{<id> <expr>}+} <expr>}}
   <id>  ::= a | b | c | ...
   <num> ::= 1 | 2 | 3 | ...
   <op>  ::= + | - | * | / | modulo | expt | add1 | sub1
```

En Racket podemos definir la gramática anterior mediante el siguiente tipo abstracto de datos.

```
;; Definición del tipo Binding
   (define-type Binding
       [binding (id symbol?) (value WAE?)])

   ;; Definición del tipo WAE
   (define-type WAE
       [id   (i symbol?)]
       [num  (n number?)]
       [op   (f procedure?) (args (listof WAE?))]
       [with (bindings (listof binding?)) (body WAE?)])
```

Este tipo de datos está conformado por los siguientes constructores:

- `id` que representa una variable algebraica.
- `num` que representa un número.

- `op` que representa una operación aritmética. El primer parámetro es la función que se va a aplicar. Tiene ser una función que forme parte de la gramática del lenguaje. En este caso, tiene que ser alguna de las siguientes: $\{+, -, *, /, modulo, expt, add1, sub1\}$. El segundo parámetro la lista de argumentos que va a operar la función anterior.

  Por ejemplo, la instrucción: $\{op + `(1\ 2\ 3)\}$ representa la operación $(1 + (2 + 3))$.

- `with` que representa la definición de un conjunto de variables (`bindings`) conformados por la pareja (`id value`), dentro de un ambiente, en una expresión `body` de la siguiente forma:

```
{with {{id1 val1}{id2 val2}} {expr}}
```

  El valor de las variables `id1` y `id2` tendrá validez únicamente dentro de la expresión `expr`.

1.2. Caché de sustituciones

El caché de sustitución guarda símbolos de variables y sus valores, que son expresiones del lenguaje. La sintaxis abstracta del caché de sustituciones se modela con el siguiente tipo abstracto de datos en *Racket*:

```
;; Data-type que representa un caché de sustituciones
(define-type DefrdSub
    [mtSub]
    [aSub (name symbol?) (value WAE?) (ds DefrdSub?)])
```

`mtSub` representa un caché de sustituciones vacío. `aSub` es un caché de sustituciones con una asignación de un valor a una variable, seguido de otro caché de sustituciones. Notemos que esta definición es recursiva, donde el caso base es un caché vacío y el caso recursivo es una asignación y otro caché de sustituciones. La única forma de buscar en el caché de sustituciones, es a través de la primer asignación definida (en un `aSub`) y recursivamente con la cola del caché hasta que llegamos al caché vacío.

2. Ejercicios propuestos

1. Define el procedimiento `(lookup var ds)`, el cual busca el id de la variable `var` en el caché de sustituciones `ds`. El procedimiento devuelve el valor correspondiente a la variable o arrojando un error en caso de que no se encuentre.

2. Define el procedimiento `(interp expr ds)`. El cual realiza la evaluación de una expresión, considerando los valores de las variables dentro del ambiente `ds` dado. Es decir, que en la expresión `expr` todas las variables tomarán el valor que se asignó dentro del ambiente `ds`, posteriormente se evaluará la expresión.

```
;; interp: WAE AE DefrdSub -> WAE
(define (interp expr ds) ...)
```

3. ¿Cuál es la diferencia entre este tipo de sustitución (con ambientes o caché de sustitución diferida) y la sustitución textual (del ejercicio de la semana anterior)? Dada la misma expresión con las mismas sustituciones ¿Siempre es lo mismo evaluar con ambas formas?

4. ¿Cuál es la complejidad de la evaluación de una expresión con el algoritmo de sustitución? ¿Cuál es la complejidad de la evaluación de una expresión con ambientes?