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>}}
         | {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?)]
       [with* (bindings (listof binding?)) (body WAE?)])
```

2. Ejercicios propuestos

Puedes reutilizar los archivos ***grammars.rkt*** e ***interp.rkt*** del ejercicio anterior para realizar lo siguiente:

1. Completar el cuerpo de la función `(parse sexp)` dentro del archivo `parser.rkt`. La cual recibe una expresión simbólica[1], realiza el análisis sintáctico correspondiente, esto es, construye un Árbol de Sintaxis Abstracta[2] (ASA). Para el análisis sintáctico de las operaciones aritméticas se debe hacer un mapeo entre los símbolos de función en sintaxis concreta y las funciones de Racket.

```
;; parse: s-expression -> WAE
(define (parse sexp) ...)
```

   A continuación se muestran algunos ejemplos:

```
> (parse '2)
(num 2)
> (parse '{+ 2 3})
(op #<procedure:+> (list (num 2) (num 3)))
> (parse '{+ 2 x})
(op #<procedure:+> (list (num 2) (id 'x)))
> (parse '{with [(x 2) (y 3)] (+ x 3 y 5)})
(with (list (binding 'x (num 2)) (binding 'y (num 3)))
   (op #<procedure:+> (list (id 'x) (num 3) (id 'y) (num 5))))
> (parse '{with [(x 2) (y 3) (x 4)] (+ x 3 y 5)})
Error: Hay un identificador duplicado
> (parse '{with* [(x 2) (y 3) (x 4)] (+ x 3 y 5)})
(with* (list (binding 'x (num 2)) (binding 'y (num 3)) (binding 'x (num 4)))
   (op #<procedure:+> (list (id 'x) (num 3) (id 'y) (num 5))))
```

2. Completar el cuerpo de la función `(interp expr)` dentro del archivo `interp.rkt` el cual recibe una expresión y regresa su evaluación correspondiente. Utiliza la función `lookup` dek ejercicio anterior.

```
;; interp: WAE -> number
(define (interp expr) ...)
```

   Toma en consideración:

   - **Identificadores**

     Se debe lanzar un error indicando que se trata de una variable libre.

```
> (interp (parse 'foo) (mtSub))
Error lookup: Variable libre: 'foo
```

   - **Números**

     Al ser un valor atómico, los números se evalúan a sí mismos.

[1] Del inglés, *s-expression*. Puede ser un número, un símbolo o una lista de expresiones simbólicas.
[2] Del inglés, *Abstract Syntax Tree (AST)*.

```
> (interp (parse 1729) (mtSub))
(num 1729)
```

- **Operaciones aritméticas**

  Se debe aplicar el operador correspondiente a la lista de operandos indicada.

```
> (interp (parse '{add1 18})(mtSub))
(num 19)
> (interp (parse '{modulo 10 2})(mtSub))
(num 0)
> (interp (parse '{expt 2 3})(mtSub))
(num 8)
> (interp (parse '{+ 1 2 3})(mtSub))
(num 6)
> (interp (parse '{- 3 2 1})(mtSub))
(num 0)
> (interp (parse '{* 1 2 3})(mtSub))
(num 6)
> (interp (parse '{/ 8 2 2}) (mtSub))
(num 2)
```

- **Asignaciones locales simples (`with`)**

  Dada la lista de parejas de identificadores con valores de la forma `(binding id value)`, se deben de sustituir en el cuerpo (`body`) correspondiente cada uno de los identificadores (`id`) por su valor (*value*).

```
> (interp (parse '{with {{a 2} {b 3}} {+ a b}})(mtSub))
(num 5)
> (interp (parse '{with {{a 2} {b {+ a a}}} b})(mtSub))
Error interp.rkt:45:10: Variable libre
```

- **Asignaciones locales anidadas (`with*`)**

  Dada la lista de parejas de identificadores con valores de la forma `(binding id value)`, se deben de sustituir en el cuerpo (`body`) correspondiente cada uno de los identificadores (`id`) por su valor (*value*). En esta versión de `with`, se permite hacer referencia a identificadores definidos previamente (anidados).

```
> (interp (parse '{with* {{a 2} {b {+ a a}}} b}) (mtSub))
(num 4)
```