Lenguajes de Programación
Práctica 5

Karla Ramírez Pulido Manuel Soto Romero Alejandro Hernández Mora
Alma Rocío Sánchez Salgado Silvia Díaz Gómez

Semestre 2021-1

Facultad de Ciencias, UNAM
Fecha de inicio: 11 de enero de 2021
Fecha de entrega: 22 de enero de 2021

1. Objetivos

Implementar el intérprete de evaluación estática para el lenguaje ***CFWBAE*** con ambientes. Para llevar a cabo esta tarea, se debe completar el cuerpo de las funciones faltantes dentro de los archivos ***grammars.rkt***, ***parser.rkt***, ***desugar.rkt*** e ***interp.rkt***[1].

La gramática del lenguaje `CFWBAE` se presenta a continuación:

```
<expr> ::= <id>
         | <num>
         | <bool>
         | {<op> <expr>+}
         | {if <expr> <expr> <expr>}
         | {cond {<expr> <expr>+} {else <expr>}}
         | {with {{<id> <expr>}+} <expr>}
         | {with* {{<id> <expr>}+} <expr>}
         | {fun {<id>*} <expr>}
         | {<expr>}{<expr>*}
<id>   ::= a | b | c | ...
<num>  ::= 1 | 2 | 3 | ...
<bool> ::= true | false
<op>   ::= + | - | * | / | modulo | expt | add1 | sub1
         | < | <= | = | > | >= | not | and | or | zero?
```

Nota que agregamos tipos booleanos, por lo tanto la expresión `if0` será reemplazado por la instrucción `if`, cuya condición es una expresión que al ser interpretada da como resultado una expresión booleana. La gramática anterior se define mediante los siguientes tipos en *Racket*.

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

[1]Puedes reutilizar los archivos que entregaste en la práctica anterior, agregando los nuevos tipos incluidos en esta práctica

```
;;Definición del tipo condition para la definición de cond.
(define-type Condition
  [condition (test-expr SCFWBAE?) (then-expr SCFWBAE?)]
  [else-cond (else-expr SCFWBAE?)])

;; Definición del tipo SCFWBAE
(define-type SCFWBAE
  [idS   (i symbol?)]
  [numS  (n number?)]
  [boolS (b boolean?)]
  [iFS   (condicion SCFWBAE?) (then SCFWBAE?) (else SCFWBAE?)]
  [opS   (f procedure?) (args (listof SCFWBAE?))]
  [condS (cases (listof Condition?))]
  [withS (bindings (listof binding?)) (body SCFWBAE?)]
  [withS* (bindings (listof binding?)) (body SCFWBAE?)]
  [funS  (params (listof symbol?)) (body SCFWBAE?)]
  [appS  (fun SCFWBAE?) (args (listof SCFWBAE?))])
```

2. Ejercicios

1. (2.0 pts.) Completar el cuerpo de la función `(parse sexp)` dentro del archivo `parser.rkt`. Agrega las nuevas reglas sintácticas que aparecen con respecto a la práctica anterior.

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

   La función `parse` deberá tomar un símbolo o una lista de elementos y transformarlo a una expresión del lenguaje ***SCFWBAE***, que será representado mediante la sintaxis abstracta intermedia, definida anteriormente. Esta sintaxis abstracta considera expresiones con azúcar sintáctica.

2. (4.0 pts) Completar el cuerpo de la función `(desugar sexp)` dentro del archivo `desugar.rkt`.

   ```
   ;; desugar SCFWBAE-> CFWBAE
   (define (desugar sexpr)...)
   ```

   Dicha función deberá tomar una expresión del tipo ***SCFWBAE***, que es la sintaxis abstracta intermedia del lenguaje con azúcar sintáctica [2] y transformarla en una expresión del tipo ***CFWBAE*** que es la sintaxis abstracta final del lenguaje, sin azúcar sintáctica.

   La sintaxis abstracta final deja de considerar las expresiones de tipo ***with, with**** y ***cond***. Dicha sintáxis se define mediante los siguientes tipos en *Racket*.

[2] La *S* viene del inglés *sugar*, que significa azúcar.

```
;; Definición del tipo CFWBAE
(define-type CFWBAE
  [id   (i symbol?)]
  [num  (n number?)]
  [bool (b boolean?)]
  [iF   (condicion CFWBAE?) (then CFWBAE?) (else CFWBAE?)]
  [op   (f procedure?) (args (listof CFWBAE?))]
  [fun  (params (listof symbol?)) (body CFWBAE?)]
  [app  (fun CFWBAE?) (args (listof CFWBAE?))])
```

La función ***desugar*** deberá considerar el manejo de las siguientes expresiones como se indica a continuación.

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

  Las expresiones del tipo `with` serán transformadas en una aplicación de función. Por ejemplo:

  ```
  '{with {{a 2}} {sub1 a}}
  ```

  Se transformará en:

  ```
  (app
      (fun '(a) (op #<procedure:sub1> (list (id 'a))))
      (list (num 2)))
  ```

- **Asignaciones locales anidadas (*with**):**
  Como ya mencionamos antes, ***with**** también dejará de formar parte de la sintáxis abstracta, por lo que la función desugar se encargará de transformar las instancias de ***with**** en instancias anidadas de ***with***, que posteriormente también se van a eliminar como se especificó antes. Por ejemplo la expresión

  ```
  '(with* ((x 1) (y 2)) (+ y x))
  ```

  se transformará en[3]:

  ```
  '(with ((x 1))
      (with ((y 2))
          (+ y x)))
  ```

  y a su vez esto se transforma en:

  ```
  (app
      (fun '(x)
          (app
               (fun '(y)
                    (op #<procedure:+> (list (id 'y) (id 'x))))
               (list (num 2))))
      (list (num 1)))
  ```

[3]Esto es opcional, no es necesario que esperemos hasta el desugar, podemos hacer el cambio directo desde el parser, aunque puede ser más truculento.

- **Condicionales (*cond*):**
  En este intérprete ya tenemos tipos booleanos, por lo que es conveniente cambiar el tipo de condicional a una booleana. Recordemos que la expresión condicional es azúcar sintáctica para evitar escribir expresiones *iF* anidadas, por lo que en esta función deberemos eliminar el azúcar sintáctica. Por ejemplo:

```
>(desugar (parse '{cond {#t 1} {#f 2} {else 3}}))
(iF (bool #t) (num 1) (iF (bool #f) (num 2) (num 3)))
>(interp (desugar (parse '{cond {#t 1} {#f 2} {else 3}})) (mtSub))
(numV 1)
>(interp (desugar (parse '{cond {#f 1} {#f 2} {else 3}})) (mtSub))
(numV 3)
> (interp (desugar (parse '{cond {#f 1} {#t 2} {else 3}})) (mtSub))
(numV 2)
```

3. (1 pts.) Completar el cuerpo de la función `(lookup name ds)` dentro del archivo `interp.rkt` la cual busca el nombre de la variable `name` dentro del caché de sustitución `ds`, regresando el valor correspondiente o arrojando un error en caso de que no lo encuentre.

```
;; lookup: symbol DefrdSub -> CFWBAE
(define (lookup name ds) ...)
```

   El caché de sustitución guarda símbolos de variables y sus valores, que son expresiones del lenguaje. La sintáxis abstracta del caché de sustituciones se modela mediante los siguientes tipos en *Racket*:

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

4. (3 pts.) Completar el cuerpo de la función `(interp expr ds)` dentro del archivo `interp.rkt` el cual recibe una expresión, un caché de sustituciones y regresa la evaluación correspondiente de acuerdo con las variables definidas en el caché. Deberás considerar las modificaciones hechas a la gramática.

```
;; interp: CFWBAE DefrdSub-> CFWBAE-Value
(define (interp expr ds)
```

   Algunos ejemplos de los resultados esperados de la interpretación para algunas expresiones son:

```
> (interp (desugar( parse '3)) (mtSub))
(numV 3)
> (interp (desugar( parse #t)) (mtSub))
(boolV #t)
> (interp (desugar( parse 'x)) (mtSub))
Error lookup: Variable libre: 'x
```

```
> (interp (desugar (parse '{cond {#t 1} {#f 2} {else 3}})) (mtSub))
(numV 1)
> (interp (desugar (parse '{cond {#f 1} {#t 2} {else 3}})) (mtSub))
(numV 2)
> (interp (desugar (parse '{cond {#t 1} {#t 2} {else 3}})) (mtSub))
(numV 1)
>(interp (desugar (parse '{cond {#f 1} {#f 2} {else 3}})) (mtSub))
(numV 3)

> (interp (desugar (parse '(+ 1 1 1 (- 3 4 1) (sub1 1)))) (mtSub))
(numV 1)
```

Se deberán respetar todas las definiciones de las funciones, al igual que los tipos del caché de sustituciones y de los valores del intérprete *CFWBAE-Value*; es decir que está prohibido cambiar el tipo que tengan, ya sea el de algún parámetro o el del resultado.

3. Requerimientos

Se deberá subir los archivos requeridos a la plataforma google classroom antes de las 23:59 hrs del día de la fecha de entrega, conforme lo como lo indican los lineamientos de entrega. No olvides incluir el archivo `ReadMe.txt` con los datos de tus compañeros. Sólo es necesario que una persona suba la práctica, los demás compañeros deberán subir como tarea el contenido del archivo `ReadMe.txt`, de manera que se pueda ver el equipo que integran desde la plataforma google classroom.

El orden en el que aparezcan las funciones en los archivos solicitados, debe ser el orden especificado en este archivo PDF, de lo contrario podrán penalizarse algunos puntos.

La idea es que reutilices funciones de los ejercicios que ya se programaron anteriormente.

¡Que tengas éxito en tu práctica!.