Universidad Nacional Autónoma de México
Facultad de Ciencias
Lenguajes de Programación

Práctica 7

Karla Ramírez Pulido
karla@ciencias.unam.mx

J. Ricardo Rodríguez Abreu
ricardo_rodab@ciencias.unam.mx

Manuel Soto Romero
manu@ciencias.unam.mx

Fecha de inicio: 28 de abril de 2018
Fecha de término: 4 de mayo de 2018
Semestre 2018-2

Objetivo

Extendnder el lenguaje de programación implementado en la Práctica 6 para que maneje excepciones mediante el uso de continuaciones.

Antecedentes

En clase y en las sesiones previas de laboratorio se revisó el concepto de continuación y se implementó un pequeño intérprete para una versión simplificada del lenguaje de programación `ERCFWAE` que implementa manejo de excepciones. Se recomienda revisar dicha actividad junto con el libro de texto del curso [3].

Repositorio

El material necesario para completar esta práctica se encuentra en el repositorio de *GitHub Classroom* del curso: `https://classroom.github.com/g/Cg2nrv6h`.

Desarrollo de la práctica

La gramática en EBNF para las expresiones del lenguaje `ERCFWBAEL/L` (*Exceptions*, *Recursive*, *Conditionals*, *Functions*, *With*, *Booleans*, *Arithmetic Expressions Lists and Lazy)* que se implementa en esta práctica es la siguiente:

```
<expr> ::= <id>
         | <num>
         | <bool>
         | <list>
         | {<op> <expr>+}
         | {if <expr> <expr> <expr>}
         | {cond {<expr> <expr>}+ {else <expr>}}
         | {with {{<id> <expr>+}
         | {with* {{<id> <expr>}+} <expr>}
         | {rec {{<id> <expr>}+} <expr>}
         | {fun {<id>*} <expr>}
         | {<expr> <expr>*}
         | {thorws <id>}
         | {try/catch {{<id> <expr>}+} <expr>}

 <id> ::= a | ... | z | A | Z | aa | ab | ab | ... | aaa | ...
        (Cualquier combinación de caracteres alfanuméricos
         con al menos uno alfabético)

 <num> ::= ... | -1 | 0 | 1 | 2 | ...

<bool> ::= true | false

 <list> ::= empty
          | {list <expr>+}

 <op> ::= + | - | * | / | % | min | max | pow | sqrt | inc | dec
        | < | <= | = | /= | > | >= | zero?
        | not | and | or
        | head | tail | append | empty?
```

En equipos de **tres integrantes** se deben de completar cada una de las funciones faltantes de los archivos correspondientes hasta que logren pasar todas las pruebas unitarias que se incluyen en el archivo `pruebas_practica7.rkt` y se ejecute correctamente el archivo `pratica7.rkt`[1].

Ejercicio 7.1 (1 pt.) Completar el cuerpo de la función `(parse sexp)` del archivo `parser.rkt` que realiza el análisis sintáctico correspondiente, es decir, construye expresiones del TDA `ERCFWBAEL/L` incluido en el archivo `grammars.rkt`.

```
;; parse: s-expression -> ERCFWBAEL/L
(define (parse sexp) ...)
```

```
> (parse '{+ 1 2})
(opS + (list (numS 1) (numS 2))
```

[1]Para tener derecho a calificación, los archivos deben ejecutarse sin errores.

Ejercicio 7.2 (1 pt.) Completar el cuerpo de la función `(desugar expr)` del archivo `desugar.rkt` el cual elimina el azúcar sintáctica[2] de las expresiones de ERCFWBAEL/L, es decir, las convierte en expresiones del TDA ERCFBAEL/L incluido en el archivo `grammars.rkt`[3].

```
;; desugar: ERCFWBAEL/L -> ERCFBAEL/L
(define (desugar expr) ...)
```

```
> (desugar (parse '{with {{a 3}} {+ a 4}}))
(app (fun '(a) (op + (list (id 'a) (num 4)))) (list (num 3)))
```

Ejercicio 7.3 (8 pts.) Completar el cuerpo de la función `(interp expr env)` del archivo `interp.rkt` que realiza el análisis semántico correspondiente, es decir, evalúa expresiones de ERCFBAEL/L.

Además de las especificaciones que se tomaron en cuenta para resolver la Práctica 6, se deben considerar ahora, los siguientes puntos para implementar la función `interp`:

- Al lanzar excepciones se debe obtener un valor `exceptionV` que guarde el identificador de la excepción que está siendo lanzada y la pila de ejecución actual para conocer en qué punto del programa ocurrió el error, para ello se debe obtener la continuación actual usando `call/cc` o `let/cc`.

  ```
  > (interp (desugar (parse '{throws DivisionEntreCero})) (mtSub))
  (exceptionV 'DivisionEntreCero #<continuation>)
  ```

- Para atrapar excepciones se debe evaluar el cuerpo del bloque try/catch, si éste genera un error, se debe obtener el tipo de excepción generado y compararlo con la lista de posibles excepciones para saber qué valor regresar en caso de que ocurra un error de ese tipo. Para recuperarse del error, se debe aplicar la continuación capturada por el tipo `exceptionV` con el valor en caso de falla.

```
> (define expr
   '{try/catch {{DivisionPorCero 2} {MiExcepcion 4}}
      {+ 1 {throws MiExcepcion}}})
> (interp (desugar (parse expr)) (mtSub))
(numV 5)
```

```
;; interp: ERCFBAEL/L -> ERCFBAEL/L-Value
 (define (interp expr env) ...)
```

```
> (interp (op + (list (num 1) (num 2))))
3
```

[2]Tipo de sintaxis que hace que un programa sea más "dulce" o fácil de escribir.

[3]Para esta práctica no se agregan nuevas expresiones endulzadas.

Referencias

Algunas referencias de consulta:

[1] Karla Ramírez, Manuel Soto, *Notas de laboratorio del curso de Lenguajes de Programación*, Semestre 2018-2, Facultad de Ciencias, UNAM. Disponibles en: [`http://lenguajesfc.com/notas.html`].

[2] Rodrigo Ruiz Murgía, *Manual de prácticas para la asignatura de Lenguajes de Programación*, Reporte de actividad docente, Facultad de Ciencias, 2016.

[3] Shriram Krishnamurthi, *Programming Languages: Application and Interpretation*, Primera edición, Brown University, 2007.