Lenguajes de Programación

Práctica 5 - Evaluación perezosa

Semestre 2017-1

Fecha de inicio: 11 de octubre de 2016

Fecha de término: 19 de octubre de 2016

Instrucciones

- Resolver los siguientes ejercicios de manera clara y ordenada.
- Se deben modificar y entregar los archivos `streams.rkt, streams-ejm.rkt, parser.rkt` e `interp.rkt`.
- Todas las funciones deben ir acompañadas de al menos 3 pruebas unitarias en un archivo por separado `test.rkt`.
- La entrega es en equipos de máximo 3 integrantes. Seguir los lineamientos especificados en: `http://sites.ciencias.unam.mx/ldp171/formato`.
- Los puntos extra son individuales y sólo se puede escoger uno. El punto extra teórico debe de realizarse **a mano y entregarse a más tardar el 19 de octubre en la hora de laboratorio**. El punto extra de programación se envía por correo en un archivo `<no.cuenta>_P05.rkt` enviar con el asunto `[LDP-EXP5]`. Incluir el nombre completo del autor en el cuerpo del correo.

Ejercicios

Parte I: Sucesiones

En laboratorio de revisó el concepto de evaluación perezosa y cómo puede usarse para implementar estructuras de datos infinitas. En el archivo `streams.rkt` se encuentra la definición de una estructura de datos `Sucesion` que permite definir listas infinitas.

1. **Funciones de orden superior**

   Completar el cuerpo las siguientes funciones dentro del archivo `streams.rkt`:

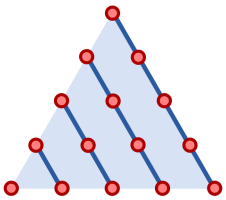
   a) Una función `(smap f s)` que toma una sucesión y regresa una nueva sucesión con la aplicación de la función `f` a cada elemento.

   b) Una función `(sfilter p s)` que toma una sucesión y regresa una nueva sucesión conteniendo únicamente los elementos que cumplen con el predicado `p`.

2. **Números triangulares**

   Un número triangular es aquel que puede descomponerse en la forma de un triángulo equilátero (por convención, el primer número triangular es el 1).

Números
triangulares



1, 3, 6, 10, 15, ...

Completar el cuerpo de las siguientes funciones en el archivo `streams-ejm.rkt`:

a) Función `(triangular n)` que toma un número `n` y regresa el `n`-ésimo número triangular. Cada número triangular está definido por la siguiente fórmula:

$$T_n = \frac{n\,(n+1)}{2}$$

b) Función `(genera-triangulares n)` que use la función anterior para definir una sucesión de números triangulares a partir del número `n`.

c) Dar una definición `triangulares` que use la función anterior para generar una sucesión de números triangulares a partir de 1 y después usarla en combinación de la función `stake` para generar los primeros 100 números triangulares.

Parte II: Un intérprete perezoso

Para la segunda parte se modificará el analizador sintáctico y el intérprete de la práctica anterior para que acepte condicionales y para que use semántica de evaluación perezosa en lugar de glotona.

La gramática de los nuevos lenguajes `CFWBAE/L` y `CFBAE/L` (la `C` es de *conditional* y la `L` es de *lazy*) es la siguiente:

```
<CFWBAE/L> ::= <num>
          | <bool>
          | <id>
          | {<op> <CFWBAE/L>+}
          | {if <CFWBAE/L> <CFWABAE/L> <CFWBAE/L>}
          | {with {{<id> <CFWBAE/L>}+} <CFWBAE/L>}
          | {fun {<id>*} <CFWBAE/L>}
          | {<CFWBAE/L> <CFWBAE/L>*}

<CFBAE/L> ::= <num>
          | <bool>
          | <id>
          | {<op> <CFBAE/L>+}
          | {if <CFBAE/L> <CFBAE/L> <CFBAE/L>}
```

```
          | {fun {<id>*} <CFBAE/L>}
          | {<CFBAE/L> <CFBAE/L>*}

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

<bool> ::= true | false

<id> ::= foo | bar | a | b | x | y | p | ...

<op> ::= +
       | -
       | *
       | /
       | %
       | min
       | max
       | pow
       | and
       | or
       | not
       | <
       | >
       | <=
       | >=
       | =
       | !=
```

1. **Análisis sintáctico / Azúcar sintáctica**

   Anexo a este PDF se encuentra un archivo `parser.rkt` que contiene la definición de dos funciones:

   - Una función `(parse expr)` que toma una expresión en sintaxis concreta y regresa su representación en sintaxis abstracta `CFWBAE/L`.
   - Una función `(desugar expr)` que toma una expresión en sintaxis abstracta `CFWBAE/L` y la transforma a su respectiva sintaxis en `CFBAE/L`.

   Completar el cuerpo de estas funciones para que procesen los tipos de datos definidos en el archivo anexo `grammars.rkt`. No olvidar usar la notación `quote` al ejecutar la función `parse`.

2. **Intérprete**

   Anexo a este PDF se encuentra un archivo `interp.rkt` que contiene la definición de una función `(interp exp env)` que toma un árbol de sintaxis abstracta y regresa su significado.

   Completar el cuerpo de esta función para que las expresiones del lenguaje `CFBAE/L` se evalúen usando semántica perezosa. Poner especial atención a las siguientes reglas y aplicar las cerraduras de expresiones como corresponda:

- Condición del `if`.
- Operaciones `n`-arias.
- Aplicación de funciones.

La evaluación perezosa debe hacerse en combinación de sustitución diferida mediante ambientes.

Puntos extra

- (2 pts.) Investigar qué es la memoización y escribir un ensayo de al menos dos cuartillas donde se explique de manera clara y órdenada en qué consiste, sus aplicaciones y resaltar cómo se usaría esta técnica para implementar la semántica perezosa del intérprete diseñado en esta práctica. Incluir además una implementación (con sintaxis de Racket) del procedimiento que calcula en n-ésimo número de fibonacci usando esta técnica. El ensayo debe contener título, introducción, desarrollo, conclusiones, bibliografía y debe estar debidamente citado.

- (2 pts.) Con base a la parte I de esta práctica, definir un procedimiento `(list->Stream l)` que toma una lista y la convierte a una sucesión. Por ejemplo:

```
> (define suc (list->Stream '(0 1 2 3 4 5)))
> suc
(scons 0 #<procedure>)
> (snext suc)
1
> (snext (stail suc))
2
```