Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Escuela de Ciencias Exactas y Naturales
Departamento de Ciencias de la Computación
R-212 — Estructuras de Datos y Algoritmos I

Trabajo Práctico Final

Intérprete de Funciones de Listas

1 Motivación del problema

Las funciones de listas son un formalismo matemático inventado (¿o descubierto?) por Giuseppe Jacopini en la década del 70 que permite definir funciones sobre listas a partir de un conjunto de operaciones primitivas, composición y repetición. Su poder expresivo es comparable con las máquinas de Turing o el lambda cálculo. Su difusión ha estado a cargo de uno de los fundadores de la Licenciatura en Ciencias de la Computación en Rosario, Raúl Kantor y está vigente al día de hoy aunque poco difundido en comparación con sus hermanos los autómatas de pila o las trilladas máquinas de Turing presentes en todo plan de estudio de carreras afines a la computación.

2 Objetivos

El trabajo práctico final consiste en entender en profundidad el formalismo de funciones de lista, desarrollar un *parser* que permite transformar texto a una o varias estructuras de datos previamente definidas y diseñar un algoritmo de búsqueda sobre aquellas estructuras de datos. Además, se propone el desarrollo de un proyecto de envergadura lo suficientemente grande para ejercitar las buenas prácticas de programación como por ejemplo la modularización.

3 Detalles de la implementación

El proyecto consiste del desarrollo de un intérprete de funciones de listas que permite definir funciones de listas y listas, un evaluador de funciones de listas y un mecanismo que permite encontrar una secuencia de funciones a componer para transicionar de una lista a otra.

3.1 Entorno interactivo (shell)

Se desea contar con un entorno que permita definir nuevas funciones de listas, definir nuevas listas, aplicar funciones de listas sobre listas e invocar la operación de búsqueda.

Al correr el programa e iniciar el modo interactivo se estará a la espera de la siguiente instrucción. Para satisfacer el *prompt* se deberá ingresar de a una sentencia terminada en ';'. Las sentencias pueden ser de tipo `deff`, `defl`, `apply` o `search`. Las dos primeras actualizarán las estructuras de datos del programa mientras que las dos últimas arrojarán un resultado en pantalla.

3.2 Descripción del lenguaje

Una ejecución completa del intérprete de funciones de listas consiste de

- 0 más sentencias de definición (de funciones y/o listas)
- seguido de 0 o más expresiones de aplicación
- seguido de 0 o más expresiones de búsqueda

Las definiciones de funciones se construyen a partir de funciones primitivas, composición y repetición y las definiciones de listas se construyen a partir de sus elementos.

3.2.1 Funciones primitivas

Las funciones base o primitivas son: `Oi`, `Od`, `Si`, `Sd`, `Di` y `Dd`.

- `Oi`: agrega un 0 a la izquierda de la lista.
- `Od`: agrega un 0 a la derecha de la lista.
- `Si`: incrementa en 1 el valor del extremo izquierdo de la lista.
- `Sd`: incrementa en 1 el valor del extremo derecho de la lista.
- `Di`: elimina el valor del extremo izquierdo de la lista.
- `Dd`: elimina el valor del extremo derecho de la lista.

El dominio de las funciones de borrado y sucesor son listas no vacías. Por ejemplo:

- Aplicar `Oi` a la lista [1,2,3] da como resultado [0,1,2,3].
- Aplicar `Sd` a la lista [0,0,0] da como resultado [0,0,1].

3.2.2 Composición

Construimos nuevas funciones de listas a partir de funciones de listas ya definidas con la palabra reservada `deff`.

Ejemplo:

```
deff f = Od Od Sd Od Sd Sd;
```

Define una función `f` como la composición de izquierda a derecha de las funciones listadas. La función `f` aplicada a la lista vacía devuelve la lista [0, 1, 2].

3.2.3 Repetición

Dada una función de listas `f` podemos definir la función repetición de `f` notada $\langle f \rangle$ como:

$$\langle f \rangle [x, X, y] = \begin{cases} [x, X, y] & \text{si } x = y \\ \langle f \rangle (f[x, X, y]) & \text{si } x \neq y \end{cases}$$

Donde x e y son elementos de la lista y X es una lista (posiblemente vacía).

Ejemplo:

```
deff Mi = Oi ⟨Si⟩ Dd;
```

Define una función `Mi` que aplicada a cualquier lista con al menos un elemento mueve el extremo derecho al extremo izquierdo.

Ejemplo de evaluación:

```
Mi [X,n]
== deff Mi
(Oi ⟨Si⟩ Dd) [X,n]
== Oi
(⟨Si⟩ Dd) [0, X, n]
== ⟨⟩.2
(⟨Si⟩ Dd)(Si [0, X, n])
== Si
(⟨Si⟩ Dd) [1, X, n]
== ⟨⟩.2
(⟨Si⟩ Dd)(Si [1, X, n])
== Si
(⟨Si⟩ Dd) [2, X, n]
==
...
==
(⟨Si⟩ Dd) [n, X, n]
== ⟨⟩.1
Dd [n, X, n]
== Dd
[n, X]
```

Otros ejemplos de funciones que se pueden definir:

- `Md` análogamente a `Mi`.
- `deff DDi = Od ⟨Sd⟩ Mi` que duplica el extremo izquierdo.
- `deff S = Md Oi Mi Oi ⟨Si Md Md Si Mi Mi⟩ Dd Di` que intercambia extremos.

3.3 Definición de listas

Construimos listas a partir de sus elementos con la palabra reservada `defl`.

Ejemplo:

```
defl L1 = [0, 2, 4, 8];
```

3.4 Evaluación

Dada una función de listas y una lista es posible imprimir en pantalla el resultado de evaluar la función sobre la lista con la palabra clave `apply`.

Por ejemplo, la sentencia `apply` 0_i $[1, 2, 3]$; imprime en pantalla [0,1,2,3]. Y la sentencia `apply S` $[2, 5, 7]$; imprime $[7, 5, 2]$. Notar que no todas las evaluaciones necesariamente terminan.

3.5 Búsqueda

Una sentencia de búsqueda se escribe así:

```
search L11, L12; L21, L22; ...; Ln1, Ln2;
```

Para resolver la sentencia `search` se debe encontrar una función de listas capaz de transformar la lista $Li1$ en la lista $Li2$ para todos los pares listados.

Ejemplo de pares:

```
[0, 1, 2], [1, 2, 3]; [0, 0, 0], [1, 1, 1];
```

Soluciones posibles:

- `Si Md Si Md Si Md`
- `DDi Si Md Di DDi Si Md Di DDi Si Md Di`

Cualquiera de las soluciones encontradas es válida. Notar que puede haber conjuntos de pares de listas que no tengan solución.

4 Evaluación

Se evaluarán

a) Las estructuras de datos utilizadas.

b) Los algoritmos propuestos.

c) La eficiencia de dichos algoritmos.

d) La calidad del código entregado.

4.1 Informe

Se debe entregar una descripción general de la solución explicando las decisiones más importantes tomadas.

En dicho informe se tendrán que explicitar las designaciones que den cuenta de qué función en el proyecto (nombre de la misma) resuelve

- la evaluación de una función de listas sobre una lista y
- la búsqueda de una función de listas dado un conjunto de pares de listas.

Toda línea de código se considera de autoría propia y debe ser defendida en caso de ser solicitado.