Taller 2: Abstracción de datos y sintáxis abstracta

Fundamentos de Interpretación y Compilación de LP / Grupo 01 / Prof. Robinson Duque / Monitor Mauricio Muñoz Gutierrez / 2024-II

Los indicadores de logro que se abordarán en este taller están determinados por:

- Implementa Tipos Abstractos de Datos utilizando dos componentes: una interfaz y una implementacón
- Utiliza estrategias con listas y datatypes para representar tipos de dato definidos por una gramática
- Construye un árbol de sintaxis abstracta a partir de una representación concreta en una gramática BNF
- Implementa procedimientos Parse y Unparse para derivar entre representación concreta y abstracta de una gramática BNF
- Utiliza herramientas para especificar analizadores léxicos y sintacticos de una gramática BNF

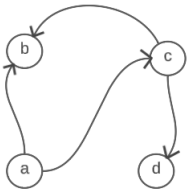
1 Grafos dirigidos

Los grafos son estructuras que sirven para representar relaciones entre objetos. Se define formalmente como $G = (V, E)$, donde V representa el conjunto de nodos o vértices y E representa el conjunto de aristas o conexiones entre nodos, cada arista tiene la forma (v, u) tales que $v, u \in V$

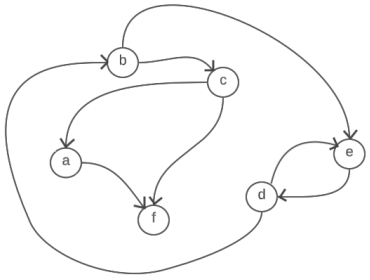
Los grafos son estructuras muy utilizadas pues tienen multiples aplicaciones como en redes sociales (conexiones entre las personas y relaciones de amistad), geografía (rutas y logísticas), infraestructura (dispositivos tecnológicos conectados en una empresa), entre otros.

Un grafo dirigido es un tipo de grafo en el que sus conexiones tienen una dirección especifica, a diferencia del grafo no dirigido, en el cual las aristas son relaciones simétricas y no apuntan en ningún sentido. De manera formal se entiende que la arista (v, u) NO es la misma arista (u, v) (situación que no es verdadera en un grafo no dirigido). Un ejemplo de un grafo dirigido es:

Sea el grafo $G_1 = (V_1, E_1)$ donde $V_1 = \{a, b, c, d\}$ y $E_1 = \{(a, b), (c, d), (c, b), (a, c)\}$, gráficamente corresponde al siguiente grafo:



Otro ejemplo, sea el grafo $G_2 = (V_2, E_2)$ donde $V_2 = \{a, b, c, d, e, f\}$ y $E_2 = \{(a, f), (c, f), (c, a), (d, b), (e, d), (d, e), (b, e), (b, c)\}$, gráficamente se ve de la siguiente manera:



En la actualidad se han definido muchos algoritmos y técnicas para analizar y trabajar con éstas estructuras, en nuestro caso vamos a implementar algunas funciones que podrían ser de interés sobre grafos dirigidos.

2 Ejercicios

2.1 (30pts) Gramática BNF

- Implemente una gramática que permita definir grafos. En particular vamos a asumir que tal gramática corresponde a una sobre grafos dirigidos. Existen muchas formas para expresar un grafo, un ejemplo teniendo en cuenta G_1 es el siguiente:

```
'(graph
    (vertices (a,b,c,d))
    (edges ((a,b),(c,d),(c,b),(a,c))))
```

  Donde graph indica que la expresión corresponde a un grafo y le siguen dos elementos correspondientes a los vértices y a las aristas.

  - (15 Pts) Proponga una implementación de la gramática basada en listas: Esta implementación deberá contener los respectivos constructores (graph, vertices, aristas o edges) y extractores ($graph->vertices, graph->edges, vertices->nodelist$), o los que considere en su gramática. Incluya ejemplos donde se evidencie su utilización y la creación de por lo menos 3 grafos dirigidos.
  - (15 Pts) Proponga una implementación basada en datatypes. Incluya ejemplos donde se evidencie su utilización y la creación de por lo menos 3 grafos dirigidos. Revise los ejemplos de sintáxis abstracta de la sección 2.2 y 2.3 para que lo implemente en su gramática.

2.2 (40 pts) Funciones Parse y Unparse

- (20pts) Para la representación basada en listas, construya una función PARSEBNF donde dada una lista con la representación concreta de un grafo dirigido, construya el árbol de sintáxis abstracta basado en datatypes. Un ejemplo es el siguiente:

```
(PARSEBNF
    '(graph
        (vertices (a b c d))
        (edges ((a b) (c d) (c b) (a c)))))

> #(struct:graph-exp
    #(struct:vertices-exp (a b c d))
    #(struct:edges-exp
        (#(struct:edge-exp a b)
        #(struct:edge-exp c d)
        #(struct:edge-exp c b)
        #(struct:edge-exp a c))))
```

- (20pts) Para la representación basada en datatypes, construya una función UNPARSEBNF donde dado un árbol de sintáxis abstracta de un grafo dirigido, entregue la representación concreta basada en listas. Por ejemplo, si realiza UNPARSEBNF sobre el resultado del PARSEBNF, deberia obtener la lista original.

2.3 (30 pts) Funciones sobre grafos dirigidos

- (14 pts) Implemente una función llamada `add-edge` que reciba un grafo dirigido (basada en listas o datatypes) y una arista o conexión y de como respuesta el mismo grafo adicionando la nueva arista. Por ejemplo:

```
(add-edge #(struct:graph-exp
    #(struct:vertices-exp (a b c d))
    #(struct:edges-exp
        (#(struct:edge-exp a b)
        #(struct:edge-exp c d)
        #(struct:edge-exp c b)
        #(struct:edge-exp a c)))) '(a,d))

> #(struct:graph-exp
  #(struct:vertices-exp (a b c d))
  #(struct:edges-exp
```

```
(#(struct:edge-exp a b)
#(struct:edge-exp c d)
#(struct:edge-exp c b)
#(struct:edge-exp a c)
#(struct:edge-exp a d))))
```

Tenga en cuenta que, al tratarse E de un conjunto de aristas, no pueden haber elementos repetidos, es decir, no debe incluir aristas que ya existan en el grafo; y que (v, u) NO es igual a (u, v) por la definición de grafos dirigidos.

- (8 pts) Implemente una función llamada `vecinos-salientes` que reciba un grafo dirigido (basada en datatypes) y un vértice o nodo n y da como respuesta una lista de nodos que se pueden alcanzar directamente desde el nodo dado (adyacentes salientes). Por ejemplo:

```
(vecinos-salientes #(struct:graph-exp
    #(struct:vertices-exp (a b c d))
    #(struct:edges-exp
        (#(struct:edge-exp a b)
        #(struct:edge-exp c d)
        #(struct:edge-exp b c)
        #(struct:edge-exp a d)
        #(struct:edge-exp c a)))) 'a)

> (b,d)
```

- (8 pts) Implemente una función llamada `vecinos-entrantes` que reciba un grafo dirigido (basada en datatypes) y un vértice o nodo n y da como respuesta una lista de nodos desde los cuales se puede llegar al nodo dado (adyacentes entrantes). Por ejemplo:

```
(vecinos-entrantes #(struct:graph-exp
    #(struct:vertices-exp (a b c d))
    #(struct:edges-exp
        (#(struct:edge-exp a b)
        #(struct:edge-exp c d)
        #(struct:edge-exp b c)
```

```
        #(struct:edge-exp a d)
        #(struct:edge-exp c a))))  'd)

> (c,a)
```

Aclaraciones

1. El taller es en grupos de dos (2) estudiantes.

2. Se debe subir al campus virtual en el enlace correspondiente a este taller un archivo comprimido **.zip** que siga la convención *CódigodeEstudiante1-CódigodeEstudiante2-Taller2FLP.zip*. El archivo con los ejercicios deberá nombrarse **ejercicio1.rkt** y deberá contener el desarrollo de los puntos definidos anteriormente.

3. Para todos los tipos de datos se debe **incluir la gramática que utilizó**.

4. En las primeras líneas de cada archivo deben estar comentados los nombres y los códigos de los estudiantes participantes.

5. Se debe incluir para cada procedimiento un comentario que explique lo que realiza cada función y para qué es empleada. También deben documentar los procedimientos que hayan implementado como solución a los problemas y de igual manera las funciones auxiliares que se utilicen, con ejemplos de prueba (mínimo 2 pruebas).

Entregas Tardías o por Otros Medios

1. Este taller **sólo se recibirá a través del campus virtual**. Adicionalmente, sólo se evaluarán los documentos solicitados en el punto 2 de la sección anterior. Cualquier otro tipo de correo o nota aclaratoria será descartado. Las fechas de entrega se definiran en el mismo campus virtual.

2. Las entregas tarde serán penalizadas así: (-1pt de la nota final obtenida) por cada hora de retraso o fracción. Por ejemplo, si usted realiza su entrega y el campus registra las 24:00 (i.e., 1min después de la hora de entrega), usted está incurriendo en la primer hora de retraso. Asegurese con mínimo dos horas de anticipación que el link de carga funciona correctamente toda vez que es posible incurrir en una entrega tardía debido a los tiempos de respuesta.