Practica extraordinaria Lenguajes de Programación

Daniel de Vicente Garrote

Gramatica inicial:

$$F \rightarrow graphid\ \{\ L\ \}$$
$$L \rightarrow L\ ;\ C\ |\ C$$
$$C \rightarrow N\ |\ N\ E$$
$$N \rightarrow id\ |\ longId$$
$$E \rightarrow edge\ N\ E\ |\ edge\ N$$

Cambios en gramática:

$$L \rightarrow L\ ;\ C\ |\ C \quad \Longrightarrow \quad \begin{array}{c} L \rightarrow C\ B \\ B \rightarrow\ ;\ C\ B\ |\ \varepsilon \end{array}$$

$$C \rightarrow N\ |\ N\ E \quad \Longrightarrow \quad \begin{array}{c} C \rightarrow N\ D \\ D \rightarrow E\ |\ \varepsilon \end{array}$$

$$E \rightarrow edge\ N\ E\ |\ edge\ N \quad \Longrightarrow \quad \begin{array}{c} E \rightarrow edge\ N\ D \\ D \rightarrow E\ |\ \varepsilon \end{array}$$

Gramática equivalente

$$F \rightarrow graph\ id\ \{\ L\ \}$$
$$L \rightarrow C\ B$$
$$B \rightarrow\ ;\ C\ B\ |\ \varepsilon$$
$$C \rightarrow N\ D$$
$$D \rightarrow E\ |\ epsilon$$
$$N \rightarrow id\ |\ longId$$
$$E \rightarrow edge\ N\ D$$

Con estos cambios en la gramática, nos quedaría un equivlaente pero del tipo LL(1).

First y follow:

FIRST	F	L	B	C	D	N	E
	graph	id	;	id	edge	id	edge
		longId	ε	longId	ε	longid	

FOLLOW	F	L	B	C	D	N	E
	$	}	}	;	}	edge	}
				}	;	;	;
						}	

Tabla de analisis sintáctico predictivo:

TASP	graph	edge	{	}	;	id	longId	$
F	graph id { L }							
L						CB	CB	
B				ε	;CB			
C						ND	ND	
D		E		ε	ε			
N						id	longId	
E		edgeND						

Funcionamiento del programa: El programa se compone de:

- Un analizador léxico hecho en LEX
- Un programa en C, que toma un fichero de entrada y genera dos de salida

Analizador léxico: El analizador se encarga de leer los elementos del fichero y de determinar de que tipo son. Para cada tipo detecta:

- GRAPH: El lexema graph, con el que comienza el fichero de entrada
- EDGE: El lexema - - , que indica un arco entre dos nodos
- ID: Nombre de un nodo, que se identifica con un primer carácter del alfabeto inglés o con una barra baja, pudiendo tener luego cero o varios caracteres alfanuméricos o barras bajas.
- LONGID: Nombre de un nodo, que se identifica con una combinacion de uno o varios caracteres alfanuméricos entre comillas. Puede tener dentro de este comillas como caracteres literales(\").
- [\t\n]: Ignora los espacios, las tabulaciones y los saltos de linea a la hora de detectar elementos.
- En el resto de casos, salta error de componente inesperado.

Analizador sintáctico descendiente recursivo: El programa en C se encarga de leer un fichero que se le meta como primer argumento de entrada, y de ahí va moviendose por las diferentes funciones del programa, que representan partes de la gramática, o funciones que crean las estructuras de datos necesarias y registran en ellas la información sobre los nodos, su número de hijos, su nodo padre, o si son nodos simples o entre comillas.

Durante el recorrido del fichero, si el programa detecta un valor de tipo ID o LONGID, además de parearlo, comprueba que no se haya guardado previamente en la lista de nodos, y si no esta se le añade(funcion creaNodo(text)). Así mismo se guarda en un array de strings los nodos recorridos y las relaciones del tipo - - .

Una vez recorrido todo el grafo del fichero, se procede a determinar las descendencias de los nodos a partir de una cadena que contiene los nodos recogidos por el analizador, asi como el lexema - - para determinar las relaciones entre

estos(funcion calculaHijos()). Luego se crea una matriz bidimensional de enteros para indicar las relaciones entre nodos con unos y ceros, en base al número de nodos diferentes. Si un nodo se relaciona consigo mismo, solo se tiene en cuenta como un hijo, y no se añade como su propio padre.

Una vez creada la matriz y rellenada en base a la lista de nodos anterior. Se procede a crear dos ficheros (si no estan creados), y se abren en modo escritura. El primero guarda la lista de nodos, entrecomillandolos si no lo están, su número de hijos, su nodo padre y el número asociado al orden en el que se añadió(nodos.txt). El segundo fichero contiene la matriz bidimensional de las relaciones(arcos.txt).

Instrucciones de ejecucion:

- Extraer p3b.l, p3b.c y tipo.h en una carpeta junto a los codigos fuente de tipo DOT.
- Ejecutar: lex p3b.l
- Ejecutar: cc -o p3b p3b.c -lfl (salen varios warnings sobre la declaracion de los metodos, se pueden omitir con -w pero no es necesario).
- Ejecutar el programa compilado: ./p3b <nombre-del-fichero.gv>

Ejemplo del enunciado:

graph miGrafo { NodoAislado ; Nodo_Raiz -- Nodo -- "Nodo \"final\""; Nodo -- Otro }

Nodos.txt

0 "NodoAislado" 0

1 "Nodo_Raiz" 1

2 "Nodo" 2 "Nodo_Raiz"

3 "Nodo \"final\"" 0 "Nodo"

4 "Otro" 0 "Nodo"

Arcos.txt

0 0 0 0 0

0 0 1 0 0

0 0 0 1 1

0 0 0 0 0

0 0 0 0 0

Ejemplos propios:

```
graph m{
    a -- b;
    b -- c;
    a -- c;
    d -- c;
    e -- c;
    e -- a
}
```

Nodos.txt

```
0    "a"  2    "e"
1    "b"  1    "a"
2    "c"  0    "e"
3    "d"  1
4    "e"  2
```

Arcos.txt

```
0 1 1 0 0
0 0 1 0 0
0 0 0 0 0
0 0 1 0 0
1 0 1 0 0
```

```
graph ciclo{
    a           --          b;
    b
    --
    a
}
```

Nodos.txt

```
0    "a"  1    "b"
1    "b"  1    "a"
```

Arcos.txt

```
0 1
1 0
```

```
graph auorelacion{
    a -- b;
    a -- a
}
```

Nodos.txt

```
0    "a"  2
1    "b"  0    "a"
```

Arcos.txt

```
1 1
0 0
```

```
graph cadenas{
	"a"
	-- "Nodo \"sucesor\"";
	"No nodo";
	"a" --			"a"
}
```

Nodos.txt

```
0	"a"	2
1	"Nodo \"sucesor\""	0	"a"
2	"No nodo"	0
```

Arcos.txt

```
1 1 0
0 0 0
0 0 0
```

```
graph autoCadena{
	"a" -- "a";
	"a" -- "b"
}
```

Nodos.txt

```
0	"a"	2
1	"b"	0	"a"
```

Arcos.txt

```
1 1
0 0
```