Documentación Analizador Sintáctico

Objetivo: Construir en un mismo programa, los analizadores Léxico, Sintáctico y Semántico.

Procesos Semánticos

- Adecuar la gramática definida en clase que incluya símbolos de acción y atributos para que realice la actualización del campo *tipo* en la tabla de símbolos.
- Revisar que al usar cada identificador esté declarado.
- Completar el analizador léxico-sintáctico para que incluya el análisis semántico con las funciones mencionadas en los puntos anteriores.

Descripción del problema.

Para este proyecto/programa opcional de la materia de compiladores, tuvimos que elaborar un analizador semántico. Éste debería poder corroborar que los identificadores del programa (tanto de variables como de funciones) estuvieran declarados. Mediante el agregar diferentes atributos a la gramática se debía ir actualizando la tabla de símbolos conforme se declararán los identificadores. Esta actualización consistía en alterar la tercera columna de la tabla, la cual representaba el tipo de variable del identificador y fue inicializada en -1.

Además, los requerimientos nos establecían que el análisis semántico debía checar que cuando se declara un identificador, este no estuviera ya previamente declarado (que su tipo no estuviera definido en la tabla). Y que cuando se utilizara un identificador, éste ahora sí ya hubiese sido declarado con anterioridad (con el tipo con un valor diferente a la inicialización de -1).

Las especificaciones de entrada definidas para el programa se siguen manteniendo. El programa deberá recibir un archivo a través de la línea de comandos al ser ejecutado. Por otro lado, durante el análisis deberá crear un total de 6 archivos de salida de la ejecución:

- CadenaAtomos.txt
- Tokens.txt
- TablaSimbolos.txt (actualizada)
- TablaLiteralesCadenas.txt
- TablaLiteralesNumeros.txt
- Errores.txt (léxicos, sintácticos y semánticos).

A continuación, se muestra la gramática, los renglones en naranja son los que habrá que modificar. Estos renglones están relacionados con la declaración o uso de identificadores, tanto de variables como de funciones.

(Los renglones morados son producciones repetidas).

1:	<Programa> → <ListaD>[<SerieF>]
2:	<ListaD> → D<ListaD>
3:	<ListaD> → ξ
4:	<SerieF> → <Func><otraF>
5:	<otraF> → <Func><otraF>
6:	<otraF> → ξ
7:	D → <Tipo>L:
8:	<Tipo> → g
9:	<Tipo> → n
10:	<Tipo> → d
11:	<Tipo> → h
12:	L → a<Valor> I’
13:	I’ → ,a<Valor> I’
14:	I’ → ξ
15:	<Valor> → = V
16:	<Valor> → ξ
17:	V → c
18:	V → s
19:	V → z
20:	V → r
21:	A → a A’:
22:	A’ → = E
23:	A’ → m E
24:	A’ → k E
25:	A’ → p E
26:	A’ → t E
27:	A’ → u E
28:	A → a A’:
29:	I → i [R] F’
30:	F’ → (S)B
31:	B → e(S)
32:	B→ ξ
33:	I → i [R] F’
34:	W → w[R](S)
35:	<For> → f [E](S)

36:	<Return> → bZ
37:	Z → [E]:
38:	Z → :
39:	E → TE’
40:	E’ → +TE’
41:	E’ → -TE’
42:	E’ → ξ
43:	T → FT’
44:	T’ → *FT’
45:	T’ → /FT’
46:	T’ → $FT’
47:	T’ → ξ
48:	F → (E)
49:	F → a
50:	F → z
51:	F → r
52:	F → [a]
53:	R → ER’
54:	R’ → >E
55:	R’ → <E
56:	R’ → ?E
57:	R’ → yE
58:	R’ → lE
59:	R’ → ¿E
60:	S → S’<otraS>
61:	S’ → A
62:	S’ → I
63:	S’ → W
64:	S’ → <For>
65:	S’ → <Return>
66:	S’ → [a]:
67:	<otraS> → S’<otraS>
68:	<otraS> → ξ
69:	<Func> → <TipoFun>a(<ListaD>S)
70:	<TipoFun> → <Tipo>
71:	<TipoFun> → v

Creación de la gramática con analizador semántico

Declaración de variables con atributos y acciones.

7:	D → <Tipo>$_{t}$ L $_{t1}$:
8:	<Tipo>$_{t1}$ → g $_{t}$
9:	<Tipo>$_{t1}$ → n $_{t}$
10:	<Tipo>$_{t1}$ → d $_{t}$
11:	<Tipo>$_{t1}$ → h $_{t}$
12:	L$_{t}$ → a $_{p}$ {RD}$_{p1}$ {AT}$_{p2,t1}$ <Valor> I'$_{t2}$
13:	I'$_{t}$ → ,a $_{p}$ {RD}$_{p1}$ {AT}$_{p2,t1}$ <Valor> I'$_{t2}$
14:	I'$_{t}$ → ξ

Declaración de funciones con atributos y acciones.

69:	<Func> → <TipoFun>$_{t}$ a $_{p}$ {RD}$_{p1}$ {AT}$_{p2,t1}$ (<ListaD>S)
70:	<TipoFun>$_{t1}$ → <Tipo>$_{t}$
71:	<TipoFun>$_{t1}$ → v $_{t}$

Utilización de identificadores (variables – funciones) con atributos y acciones.

21:	A → a $_{p}$ {RD}$_{p1}$ A':
49:	F → a $_{p}$ {RD}$_{p1}$
52:	F → [a $_{p}$ {RD}$_{p1}$]
66:	S' → [a $_{p}$ {RD}$_{p1}$]:

Donde:
{AT} es la acción de asignar un tipo a algún identificador de la tabla de símbolos.
{RD} es la acción de revisar si para algún identificador de la tabla de símbolos su tipo ya fue asignado. (es decir si su columna de Tipo ya no vale -1).
Para los atributos:
t, t1 – definen el tipo de un identificador.
p, p1, p2 – definen la posición en la tabla de símbolos de un identificador.

¿Cómo correr el programa?

Para correr el programa lo que debemos hacer es los siguiente:

1. Abrir la consola o terminal de su sistema Linux de preferencia. (Debe contar con flex/lex yacc instalado, al igual que gcc)

2. Moverse al directorio donde tenga el código fuente (cd “directorio”) y preferentemente el archivo que se analizará.

3. Ejecutar el siguiente comando de compilación flex:

```
flex semantico.l
```

4. Luego ejecutar el siguiente comando de compilación de C para el archivo previamente compilado por flex que se llamará lex.yy.c:

```
gcc -lfl lex.yy.c
```

5. Finalmente ejecutar el comando de ejecución del archivo con el código objeto generado y pasando como argumento el nombre del archivo a analizar:

```
./a.out nombreArchivoParaAnalizar.txt
```

6. Con esto habrá logrado ejecutar exitosamente el programa. Se habrán mostrado los resultados en pantalla y se le habrán generado un total de 8 archivos nuevos a su carpeta, 6 de los cuales son los resultados:
    - lex.yy.c
    - a.out
    - CadenaAtomos.txt
    - Tokens.txt
    - TablaSimbolos.txt
    - TablaLiteralesCadenas.txt
    - TablaLiteralesNumeros.txt
    - Errores.txt

Conclusiones

Este último programa de la materia nos ayudó a entender como se integraban gran parte de los temas vistos. Si bien faltaron aspectos del análisis semántico que incluir (como concordancia de los tipos en operaciones), es una buena primera aproximación a lo que podríamos encontrar o desarrollar en el mundo real. Además, en el mismo programa se le puede seguir agregando características para robustecerlo y completarlo. Una de estas sería la de agregarle la parte de traducción a código objeto o algún otro lenguaje. Que es parte de lo que vimos en el curso, pero no pudimos alcanzar a desarrollar por los límites de tiempos.

Creo que el desarrollo del analizador semántico sobre el programa que ya teníamos fue relativamente sencillo. Aunque sin duda alguna la parte más complicada fue el trabajar con archivos. Tanto para leer los tokens y poder obtener los valores de una manera que nos permitiera trabajar con ellos, como para modificar el archivo de la tabla de símbolos , tuve que dar una gran repasada al uso de archivos en lenguaje C. Me costó trabajo lidiar con el formato en el que se guardó la información, e inclusive desarrollé un programa auxiliar para hacer las pruebas de lectura y escritura con él. Pero finalmente pude lograr con éxito que funcionara el manejo de los archivos de manera satisfactoria.

En la implementación se cumplieron los objetivos del proyecto, se realizan los aspectos de análisis semántico establecidos y se va modificando la tabla de símbolos. Además, al final se le dice al usuario sus errores y si el programa es correcto o no. En definitiva, creo que el analizador semántico es el más complicado de entender debido a la gramática de traducción (los atributos y símbolos de acción como algo nuevo), pero también es la más fácil de implementar. Y al menos yo la entendí mejor al codificarlo.