Proyecto 3

Analizador Semántico

Lenguaje C-

Descripción

Hacer un programa en Python, llamado **`semantica.py`** que contenga dos funciones:

- **`tabla(AST tree, imprime = True)`**
  - La cual recibe el Árbol Sintáctico Abstracto (AST, por sus siglas en inglés) creado por el **`parser`** (proyecto 2) y una variable **`imprime`** que por default es verdadera (si es verdadera imprime la tabla o tablas de símbolos generadas).
  - Como salida genera la tabla o tablas de símbolos, una por cada bloque.
- **`semantica(AST tree, imprime = True)`**
  - La cual recibe el Árbol Sintáctico Abstracto (AST, por sus siglas en inglés) creado por el **`parser`** (proyecto 2) y una variable **`imprime`** que por default es verdadera. Esta variable la pasa cuando llame a la función **`tabla`**.
  - Llama a la función **`tabla`** y utiliza la tabla o tablas de símbolos que genera.
  - Utiliza **reglas lógicas de inferencia** para implementar la semántica de C-. Ver descripción de la semántica de C- en el documento en Bb).

Posteriormente, continuará con la definición de las dos funciones principales y todas las funciones auxiliares que requiera.

Detección y recuperación de errores

Si el analizador semántico encuentra un error (recuerde que el analizador semántico sólo encuentra errores en las declaraciones de los identificadores y los tipos de las expresiones), debe marcar la línea y el lugar (token) donde encontró el error. Por ejemplo, si la línea 22 del archivo es:

```
if (contador + fact(n)) then
```

Al detectar el tipo de la función **`fact`** se da cuenta que no es **`int`** sino **`void`** debe marcar un error de semántica porque la expresión no se puede evaluar ya que la función **`fact`** no es **`int`**.

Marcará un error, mandando un mensaje como por ejemplo:

```
Línea 22: Error en el tipo de la expresión:
if (contador + fact(n)) then
            ^
```

Donde se indica el número de línea y el acento circunflejo (^) indicará la posición del token lo más cerca posible de donde encontró el error (podría marcarlo sobre **`fact`**, lo cual dependerá de la forma en la que se detecta).

Después de esto, deberá tener un mecanismo para tratar de **recuperarse del error**, por ejemplo, suponiendo que el tipo **`void`** es correcto y continuando desde ahí. Desde luego que, nada garantiza la exactitud de lo que se detecte después, lo cual dependerá del mecanismo utilizado y, sobre todo, de la intención que tenga el programador al escribir el código, la cual desconocemos por completo y sólo la podemos suponer.

Prueba del programa

Todos los archivos necesarios para que corra (todos los que se entregan en Bb, incluyendo **lexer.py**, **parser.py** y el que contiene el archivo de prueba) se colocarán en la misma carpeta para evitar el uso de paths adicionales.

El analizador semántico será probado con un script que iniciará importando tanto el archivo con los tipos globales como el que contiene su analizador semántico. Posteriormente seguirá invocando el **`parser`** y asignando lo que regresa a una variable AST, luego invocará la función para revisar la semántica, pasándolo el AST y la variable de impresión.

La función **`parser(imprime)`**, como en el proyecto 2, deberá manejar las siguientes variables globales:

`posicion`: contiene la posición del siguiente carácter del string que se debe analizar. Deberá poder modificarla en su funcionamiento.

`progLong`: contiene la logitud del programa. Sólo la leerá cuando la requiera.

`programa`: contiene el string del programa completo. Sólo la leerá cuando lo requiera.

Por la forma en la que funciona Python al definir las variables y al hacer los **`import`**, la única forma que tenemos de pasarles estas variables globales será por medio de una función que las reciba y le pase el valor recibido a las variables globales que utiliza su programa.

La función para el paso de valores globales la tienen que agregar a su programa y tendrá la siguiente forma:

```
def globales(prog, pos, long):
    global programa
    global posicion
    global progLong
    programa = prog
    posicion = pos
    progLong = long
```

El script con el que se prueba será el siguiente:

```
from globalTypes import *
from parser import *
```

```
from semantica import *

f = open('sample.c-', 'r')
programa = f.read()          # lee todo el archivo a compilar
progLong = len(programa)     # longitud original del programa
programa = programa + '$'    # agregar un caracter $ que represente EOF
posicion = 0                 # posición del caracter actual del string

# función para pasar los valores iniciales de las variables globales
globales(programa, posicion, progLong)

AST = parser(true)
semantica(true)
```

Para la entrega

- Un documento con:
  - Las reglas lógicas de inferencia de tipos que se usaron para programar el analizador semántico.
  - La explicación de la estructura de la tabla de símbolos (y su stack en caso de ser utilizado)
- Todos los archivo Python (comentados) y TXT necesarios para que corra.

El manual del usuario no será necesario porque ya se dio la definición del lenguaje C- y la forma en la que se probará el programa.

Si este proyecto se fuera a entregar a un tercero, es indispensable entregarla toda la definición del lenguaje y la forma en la que el usuario deberá usar el analizador de léxico, paso a paso.

Nota FINAL:

Aunque existen algunas herramientas, desafortunadamente, ninguna de ellas se ha convertido en estándar para la creación automática de analizadores semánticos (como Lex o Yacc para las etapas anteriores). Por esta razón, en este tercer proyecto NO SE PUEDEN utilizar este tipo de herramientas.