Universidad Simón Bolívar
Departamento de Computación y T.I.
CI–3725 Traductores e Interpretadores
Abril–Julio 2010

Proyecto II – 1ra entrega versión 1.1

Lenguaje para graficar funciones y datos

El segundo proyecto consiste en hacer un traductor que permita graficar funciones y datos. La primera entrega consiste en un analizador léxico y sintáctico. A continuación describimos toda la sintaxis del lenguaje y algo de su semántica. Los detalles completos de la semántica serán relevantes para la segunda entrega.

1. Descripción del lenguaje

Se quiere implementar un lenguaje de graficación basado en Gnuplot (`http://www.gnuplot.info`) que a su vez producirá gráficos usando Gnuplot. A continuación describimos incrementalmente los elementos del lenguaje.

Las expresiones matemáticas (*em*) son

- Números, enteros o de punto flotante. Por ej: 8, -5, 1.2, 5.2e-3.
- Las constantes `pi` y `e`
- Variables: cualquier cadena alfabética que no corresponda a otros elementos del lenguaje. Por ej: x, y, ax
- `( em )`, una expresión matemática parentizada
- El resultado de las operaciones binarias sobre dos *em*. Los operadores binarios son `+, -, *, /, ^`
- El resultado de la operación unaria - sobre una *em*. Dicha operación permite negar la expresión. Por ej: -(1+x)
- `f ( em )`, el resultado de aplicar una función *f* sobre una expresión matemática *em*. Se cuenta además con las siguientes funciones predefinidas: `sin, cos, tan, exp, log, ciel y floor`. Por ej: `sin(1)`
- Los arreglos de *em*'s. Por ej: `[1,2,x^2]`
- `range(e,f)` si `e` y `f` son *em*
- Los arreglos por comprensión

Los arreglos son secuencias de expresiones matemáticas. Por ejemplo: `[1,8,sin(8)]` es un arreglo. Al aplicar una función o una operación unaria sobre un arreglo, el resultado es el arreglo con la función u operación aplicada a cada uno de sus elementos. Al aplicar una operación binaria sobre dos

arreglos, el resultado será el arreglo que resulte de aplicar la operación, uno a uno, a los elementos de ambos arreglos.

Si bien se definen las construcciones sintácticas del lenguaje, es posible también que se incurra en un error *semántico*. Estos incluyen variables sin declarar, o tipos de datos conflictivos. En particular, algunas expresiones con arreglos podrían resultar en un error semántico. Por ejemplo, `[1,2] + [sin(1),1^cos(3)]` es una expresión matemática que no resultaría en un error semántico. En cambio la expresión `[1,2] + [1,2,3]` es sintácticamente correcta, pero resultaría en un error semántico. Los errores semánticos serán detectados y reportados en la segunda entrega del proyecto, cuando se darán más detalles.

La expresión `range(e,f)` retorna un arreglo `[e, e+1, ..., f]` si `e` y `f` son evaluables a números enteros. Sin embargo, tal verificación se realizará a tiempo de ejecución, lo cual corresponde a la 2da entrega. Los arreglos de comprensión son expresiones de la forma: `[exp for x in arr]`. Donde `exp` es una expresión matemática, `x` es una variable y *arr* es una expresión matemática que evalúa a un arreglo. La función especial `if(cond,exp1,exp2)` que retorna la expresión matemática `exp1` si `cond` evalúa a un número distinto de 0, y `exp2` en caso contrario. Las condiciones `cond` son como las expresiones matemáticas, pero también pueden contener los operadores binarios `AND y OR`, el operador unario `NOT` y los operadores relacionales `>, <, <=, >= y ==`.

Una expresión graficable es una expresión matemática o una expresión `'file'`, con las comillas, donde `file` es el nombre de un archivo desde donde se cargará una expresión. El formato de éste archivo se aclarará para la próxima entrega.

Los instrucciones del lenguaje son las siguientes:

- *f(x) = exp*, donde `f` es una cadena alfabética que representa el nombre de una función, `x` es una variable, y `exp` es una expresión matemática. La expresión sólo puede hacer referencia a la variable `x`, a constantes, y a funciones previamente definidas, pero no a `f`. Esto último se verificará en la 2da entrega, donde también se darán otros detalles semánticos.

- `v = exp`, donde `v` es una variable, y `exp` es una expresión matemática. La variable `v` quedará asignada a la expresión matemática.

- `plot rango exp {with estilo}`, donde rango es una expresión matemática que evalúe a un arreglo (principalmente provenientes de construccion `range`, como se aqui explicó anteriormente), `exp` es una expresión graficable, la parte entre llaves – `{}` – es opcional, y `estilo` es una de las siguientes palabras: `lines, points, linespoints`. El `estilo` también puede ser un arreglo de estilos, para el caso en que se esté graficando una expresión que sea un arreglo de expresiones. Por ejemplo: `[lines, lines, points]`.

- En la siguiente expresión, `v` es una variable, `rango`, es una expresión matemática que evalúe a un arreglo (generalmente un rango) `step` es el paso con el cual se moverá la variable en el rango en cada iteración (nótese que es opcional, el valor por defecto es 1) y cada $\texttt{instr}_i$ es una instrucción.

  ```
  for v in rango {step paso}
        instr_1;
        instr_2;
        ...
        instr_n
  endfor
  ```

- `push_back(v,exp)`, donde `v` es una variable y `exp` es una expresión matemática. `push_back` añade al final del arreglo almacenado en `v` la expresión `exp`.

Cada expresión termina en ;, excepto por el for, cuya sintaxis ya fue explicada. Por ejemplo, juntando las expresiones anteriores, se puede realizar lo siguiente:

```
f=[];
for i in (1, 4)
   push_back(f,x^i+10);
endfor
plot (-10,20) f with [lines, points, lines, points];
```

que crea un arreglo con 4 expresiones.

2. Primera entrega

En la primera entrega deberá crear un analizador léxico y un analizador sintáctico. El analizador léxico le permitirá usar expresiones regulares para detectar los elementos básicos del lenguaje, los tokens. Este debe ser la salida del analizador, la cual le servirá como entrada al analizador sintáctico. Éste último le permitirá, a través de una gramática libre de contexto, reconocer las expresiones válidas del lenguaje, para luego poder interpretarlo adecuadamente en la segunda entrega. La salida del análisis sintáctico debe ser una *arbol sintáctico abstacto (AST)* correspondiente a las expresiones reconocidas.

Puede elegir el lenguaje de programación que guste de entre los siguientes: Haskell, Java, OCaML, Perl, Python o Ruby, pero deberá seguirlo usando para la segunda entrega. Deberá usar una herramienta de análisis sintáctico, y una herramienta de análisis léxico recomendada por los creadores de la herramienta de análisis sintáctico.

Como se mencionó anteriormente, el analizador sintáctico debe recibir como entrada la lista de tokens producida por el analizador léxico. Muchas herramientas de análisis sintáctico ofrecen grandes facilidades para integrarse con algún analizador léxico. El analizador sintáctico debe producir un árbol sintáctico abstracto, que será analizado estáticamente en busca de los errores semánticos, y usado por el traductor de la segunda entrega. Si el analizador léxico encuentra una cadena que no corresponde a un token válido, debe reportarse un error léxico, indicando la posición en el archivo (línea y columna) en las cuales ha sido encontrado. Del mismo modo debe proceder el analizador sintáctico al reportar un error sintáctico.

Actividades:

- Diseñar un analizador léxico que identifique los tokens del lenguaje.
    - Los espacios en blanco, tabuladores y saltos de linea deben ser ignorados. Si se encuentra el caracter #, tanto él, como todos los caracteres hasta el final de la linea deben ser ignorados. Cualquier otro caracter encontrado, que no corresponda a la definición del lenguaje, debe ser reportado como un error.
- Diseñar una gramática libre de contexto para el lenguaje presentado y escribirla usando una herramienta de análisis sintáctico. La gramática debe ser ambigua, resolviendo los conflictos utilizando las herramientas de resolución de precendencias y asociatividades que incluyen las herramientas de análisis sintáctico. Además debe utilizar recursión por izquierda la cual es más eficiente en analizadores LR.
- Diseñar suficientes TADs recursivos para representar los elementos sintácticos del lenguaje, que serán usados por el traductor a ser elaborado en la 2da entrega.

- En esta entrega su ejecutable deberá llamarse `mygnuplot`. Para usarlo se ejecutará

  `./mygnuplot entrada.txt`

  que leerá las expresiones en el archivo `entrada.txt` e imprimirá el árbol sintáctico correspondiente a cada expresión.

Requerimientos adicionales:

- El proyecto deberá funcionar en Linux, en las máquinas del LDC.
- Entregar un `Makefile` e instrucciones para compilarlo.
- El código debe estar suficientemente comentado para ser comprendido.