



DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico

2 de Julio de 2014

Teoría de lenguajes

| Integrante | LU | Correo electrónico |
|-----------------------|--------|---------------------------|
| Axel Iglesias | 79/10 | axeligl@gmail.com |
| Agustin Martínez Suñé | 630/11 | agusmartinez.92@gmail.com |
| Nahuel Lascano | 476/11 | laski.nahuel@gmail.com |



Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2160 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (54 11) 4576-3359

<http://www.fcen.uba.ar>

1. Introducción

2. Decisiones

3. Resultados

4. Gramática

El esqueleto para hacer la gramática¹ fue tomado de la página citada en la presentación². Sobre este esqueleto armamos nuestra gramática, que tiene varios puntos de diferencia con la que muestran de ejemplo en ese sitio.

Nuestra gramática parte de un símbolo inicial *programa* que a su vez se divide en una lista de *funciones* y una instrucción de *ploteo*.

Una función consiste en la palabra 'function' (representada por el token terminal TFUNC), un *nombre* (que no es más que un terminal que matchea con strings no empezados por números), una lista de *argumentos* (lista de *nombres* separadas por comas) entre paréntesis y un *bloque* de código.

Un *bloque* es una lista (quizás vacía) de sentencias encerradas entre llaves o una única sentencia.

Una *sentencia* puede ser una *asignacion* (un *nombre*, un símbolo de = y una *expresión*), una sentencia *ifthenelse*, un *while* (que a su vez tiene una *condición* y otro *bloque*) o un *return* (que lleva a su lado una *expresión*).

El caso del *ifthenelse* requiere un análisis específico. Un *ifthenelse* puede tener o no tener 'else', lo cual nos trae una ambigüedad a la hora de parsear cadenas del estilo (intencionalmente sin indentar):

```
if cond1 then
if cond2 then
codigo1
else
codigo2
```

que se refleja en el parser en un conflicto shift/reduce (cuando termina el codigo1 y viene un else puede tanto consumir el else y shiftear como reducir lo que ya tiene). Para resolverlo, nos basamos en el comportamiento de C en este caso (hacer shift, es decir, que el código se lea como

```
if cond1 then
  if cond2 then
    codigo1
  else
    codigo2
```

) y como Bison por defecto elije hacer siempre shift lo dejamos así.

Una *expresión* puede ser un *numero* (entero, double o π), una *llamada a función* (el nombre de una función con más *expresiones* entre paréntesis como argumentos) u operaciones aritméticas entre expresiones. Para salvar las ambigüedades propias de esta parte de la gramática le indicamos a Bison la precedencia de los operadores³.

Una *condición* puede ser una comparación entre dos *expresiones* u otras condiciones con operadores lógicos.

¹implementada en parser.y

²<http://gnu.org/2009/09/18/writing-your-own-toy-compiler/>

³basándonos en el código de ejemplo de <http://www-h.eng.cam.ac.uk/help/tpl/languages/flexbison/>

Por último, la instrucción *ploteo* consiste en dos *llamadafuncion* con un *nombre* de variable y tres *expresiones* que reflejan el inicio, el paso y el final.

La gramática se complementa con un archivo⁴ con directivas para el tokenizador, que le permite convertir símbolos terminales en los tokens que usamos, ignora saltos de línea y maneja números enteros, floats y cadenas arbitrarias (como los nombres de variables).

Entre los problemas que encontramos hubo uno que no pudimos resolver: no logramos armar la expresión regular para que el tokenizador ignore los comentarios multilínea. Es por esto que para pasar los tests los archivos con comentarios debe ser en primer lugar pasados por un script hecho en bash que los borra.

5. Ejemplos de funcionamiento

6. Respuestas

7. Manual

Modo de uso:

```
make clean && make  
cat archivodetest.cod | ./borracomentarios.sed | ./parser | ./graficador.sh
```

⁴tokens.l