Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Teoría de Lenguajes

Segundo Cuatrimestre del 2011

Trabajo Práctico

Integrante	LU	Correo electrónico
Ezequiel Castellano	161/08	ezequielcastellano@gmail.com
Mariano Semelman	143/08	marianosemelman@gmail.com

Índice

1. Gramática

Optamos por describir la gramática de manera tal que no tenga conflictos de ningún tipo (reduce/reduce o shift/reduce). A su vez no perdimos ni agregamos poder de expresividad, ya que de esta manera podemos representar las mismas cadenas que antes.

La gramática utilizada es la siguiente:

G = < {Expression},{caracter, |, *, +, ?, ., (,)},P,E >

P:

```
Expression --> Term
    | Term Expression
    | Term | Expression


Term --> Term +
    | Term ?
    | Term *
    | Operand

Operand --> Parenthesis
    | Character

Character --> caracter
    | .

Parenthesis --> (Expression)
```

2. Implementación

3. Información y requerimientos de software

En esta sección se indicarán versiones, herramientas, compiladores y todo lo necesario para la realización del trabajo práctico.

3.1. Requerimientos

//TODO: Completar que librerias usamos. Se que son las de C++ y esas cosas, pero no se ni como se dice y no quiero hacer lio, jeje.

Es necesario tener instaladas las librerias para compilar en c++ (GNU GCC)y Bison. El trabajo práctico fue implementado sobre Ubuntu 11.05.

3.2. Compilar

Utilizar el MakeFile provisto con el código, el cual se encuentra detallado en el apéndice.

4. Casos de Prueba

//TODO: Estos fueron los que usamos al comienzo para ver que la gramática sea válida, falta completar en que casos la debería aceptar y en cuales no.

Las casos de prueba utilizados para validar la gramática fueron los siguientes:

```
casa
(casa)
a?
(a)?
(a?)
casa|perro
(casa)|(perro)
(casa)|perro
casa|(perro)
casa?|perro?
(casa?)|(perro)?
(casa)?|perro?
casa?|(perro)?
a+?
(asd?)?
asd|asd
```

5. Resultados

//TODO: Completar cuales fueron los resultados obtenidos.

6. Conclusiones

//TODO: Completar las conclusiones.

Nos fue muy útil para chequear que la gramática no tenga conflictos el tener una herramienta como Bison.

7. Apéndice

7.1. MakeFile

```
CXXFLAGS = -g -Wall -Wextra -std=c++0x
LDFLAGS = $(CXXFLAGS)
ABSOBJ = automata grep-line.tab main matcher graph graph_utils tests/test-automata tests/te
SOURCES= $(addsuffix .cpp, $(ABSOBJ))
OBJ = $(addsuffix .o, $(ABSOBJ))
BISON = grep-line.ypp
MAIN_OBJ = automata.o grep-line.tab.o main.o matcher.o graph.o graph_utils.o
GRAPH_TEST_OBJ = graph.o graph_utils.o tests/test-graph.o
AUTOMATA_TEST_OBJ = automata.o graph.o graph_utils.o tests/test-automata.o


grep-line: bison-build $(MAIN_OBJ)
	$(CXX) $(CXXFLAGS) $(MAIN_OBJ) -o $@

clean:
	rm -f *.o tests/*.o
	rm -f graph/*.png graph/dot/*.dot
	rm deps
	rm grep-line test-automata test-graph

bison-build: $(BISON)
	bison -d $(BISON)

bison:$(BISON)
	bison -d -ggraph/dots/grammar.dot $(BISON)
	dot graph/dots/grammar.dot -Tpng -o graph/grammar.png


deps:
	$(CXX) $(CXXFLAGS) -MM -MP $(SOURCES) > $@

-include deps



test-graph: $(GRAPH_TEST_OBJ)
	$(CXX) $(CXXFLAGS) $^ -o $@

test-automata: $(AUTOMATA_TEST_OBJ)
	$(CXX) $(CXXFLAGS) $^ -o $@

build-test: grep-line test-automata test-graph

test: build-test
```

```
./tests/run-test-regexp
./test-automata
./test-graph
```