Grammer-2

COM S 319

Objectives

1. Learn formal and informal definitions of grammer.
2. Learn about types of grammer (Chomsky Hier..)
3. Learn about regular grammer
4. Learn about recognizer for regular grammer
5. Learn about LEX

REGULAR GRAMMER

Regular Grammer

- Production Rules have to have one of the forms
  1. $A \rightarrow a$
  2. $A \rightarrow aB$
  3. $A \rightarrow \varepsilon$

where A and B stand for arbitrary variables and a stands for an arbitrary terminal. Epsilon is the empty string.

Note: There is an equivalent form, where middle rule is $A \rightarrow Ba$

Example1

- This was shown before. This is an example of a regular grammer
- V = {S, A, B, C}, T = {a, b, c}
- S→ A
- A → aA
- A → B
- B → bC
- C → cC
- C → ε

Q1: What are strings in the language?

We have already learnt about regular expressions.

- Regular expressions express strings in regular language
- Regular grammer also expresses strings in regular language.
- Finite automaton is used to recognize regular expressions
- **RE, RG, FA are equivalent!**

REG EXP IMPLEMENTATION

Non-deterministic FA

- A NFA is a 5-tuple (Q, A, T, S, F)
  - Q is a FINITE set of states
  - A is the alphabet
  - T is the transition function
    - Q x A+ε → P(Q) (i.e. state & alphabet gives state
  - S is the start state
  - F is set of final states
- NFA (non-deterministic finite automaton) can
  1. transition to multiple states on the same input and
  2. can also transition on epsilon

easier to express in NFA vs DFA (note DFA and NFA are equivalent)

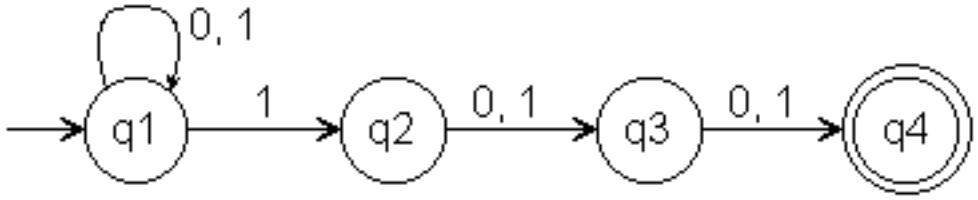
Accept a*bc*

- S0 is start state
- state S0 <epsilon, S1>,
- state S1 <a, S1>, <b, S2>,
- state S2 <c, S2>
- S2 is final state

Transition Table

- can use transition table to represent finite automaton.
  - rows represent states
  - cols represent input
  - element represents next state
- can use simple traversal over input and keeping set of future states. when an input string has been fully processed – is any of the next states a final state? If so accept – else reject.

Example Transition table



	0	1
q1	{q1}	{q1, q2}
q2	{q3}	{q3}
q3	{q4}	{q4}
q4	{}	{}

LEX (LEXER OR LEXICAL ANALYSER)

lex

- describe rules for language.
- lex automatically creates lexical analyzer.

Format of lex rules file:

{definitions}
%%
{rules}
%%
{user subroutines}

Example Lex file

```
%{
#include <stdio.h>
%}

%%
[a-zA-Z][a-zA-Z0-9]*    printf("WORD ");
[a-zA-Z0-9\/.-]+        printf("FILENAME ");
\"                      printf("QUOTE ");
\{                      printf("OBRACE ");
\}                      printf("EBRACE ");
;                       printf("SEMICOLON ");
\n                      printf("\n");
[ \t]+                  /* ignore whitespace */;
%%
```

ANTLR

http://www.antlr.org

ANTLR (ANother Tool for Language Recognition) is a powerful parser generator for reading, processing, executing, or translating structured text or binary files. It's widely used to build languages, tools, and frameworks. From a grammar, ANTLR generates a parser that can build and walk parse trees.

ANTLR

- lex and yacc – standard unix utilities to build lexer and parser (to build compilers).
  - c code
  - lexer and parser rules kept in separate files

- ANTLR
  - completely Java code.
  - Both lexer and parser rules are specified in one file.

- Commands

  antlr4 Expr.g4

  javac Expr*.java

  grun Expr prog –gui

  100+2*34 ^D

SUMMARY

- Basics definitions (terminal, alphabet, rules,..
- Chomsky Hierarchy (or Types of grammers)
- Regular Grammer
- Equivalence of RE, RG, FA
- Implementation NFA using transition table
- Implementation using lex & ANTLR