CMPT 379
Compilers

Anoop Sarkar

http://www.cs.sfu.ca/~anoop

Implementing Regular Expressions with Finite-state Automata

Regular Expressions

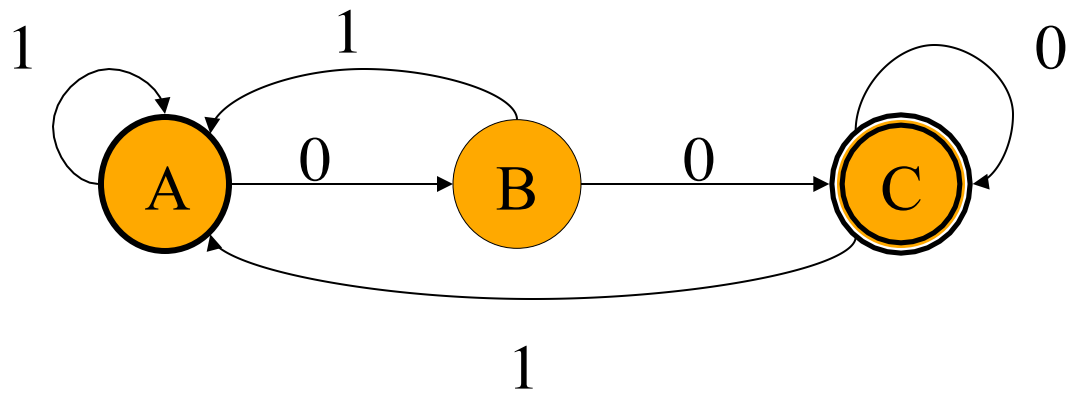
- To describe all lexemes that form a token as a *pattern*
  - (0|1|2|3|4|5|6|7|8|9)+
- Need decision procedure: to which token does a given sequence of characters belong (if any)?
  - Finite State Automata
  - Can be deterministic (DFA) or non-deterministic (NFA)

Deterministic Finite State Automata: DFA

- A set of states S
  - One start state q_0, zero or more final states F
- An alphabet $\sum$ of input symbols
- A transition function:
  - δ: S x $\Sigma \Rightarrow$ S
- Example: $\delta(1, a) = 2$

DFA: Example

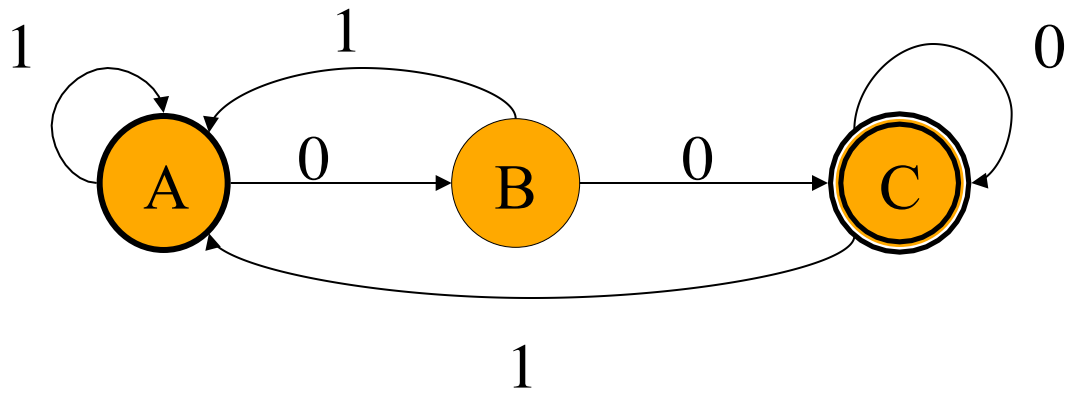
- What regular expression does this automaton accept?



A: start state
C: final state

Answer: (0|1)*00

DFA simulation



Input string: 00100

DFA simulation takes at most *n* steps for input of length *n* to return accept or reject

- Start state: A
  1. δ(A,0) = B
  2. δ(B,0) = C
  3. δ(C,1) = A
  4. δ(A,0) = B
  5. δ(B,0) = C
- no more input and C is final state: **accept**

Building a Lexical Analyzer

- Token ⇒ Pattern
- Pattern ⇒ Regular Expression
- Regular Expression ⇒ NFA
- NFA ⇒ DFA
- DFAs or NFAs for all the tokens ⇒ **Lexical Analyzer**
- Two basic rules to deal with multiple matching: **greedy match** + **regexp ordering**

Note that **greedy** means *longest leftmost match*

Lexical Analysis using Lex

```
%{
#include <stdio.h>
#define NUMBER     256
#define IDENTIFIER 257
%}

/* regexp definitions */
num [0-9]+

%%

{num}           { return NUMBER; }
[a-zA-Z0-9]+    { return IDENTIFIER; }

%%

int
main () {
  int token;
  while ((token = yylex())) {
    switch (token) {
      case NUMBER: printf("NUMBER: %s, LENGTH:%d\n", yytext, yyleng); break;
      case IDENTIFIER: printf("IDENTIFIER: %s, LENGTH:%d\n", yytext, yyleng); break;
      default: printf("Error: %s not recognized\n", yytext);
    }
  }
}
```

simpletok.lex