Regular Expressions

with a brief intro to FSM

15-123

Systems Skills in C and Unix

Case for regular expressions

- Many web applications require pattern matching
  - look for <a href> tag for links
  - Token search
- A regular expression
  - A pattern that defines a class of strings
  - Special syntax used to represent the class
    - Eg; *.c - any pattern that ends with .c

Formal Languages

- Formal language consists of
  - An alphabet
  - Formal grammar

{a, b, c, ...}

- Formal grammar defines
  - Strings that belong to language

{abc, cab, ...

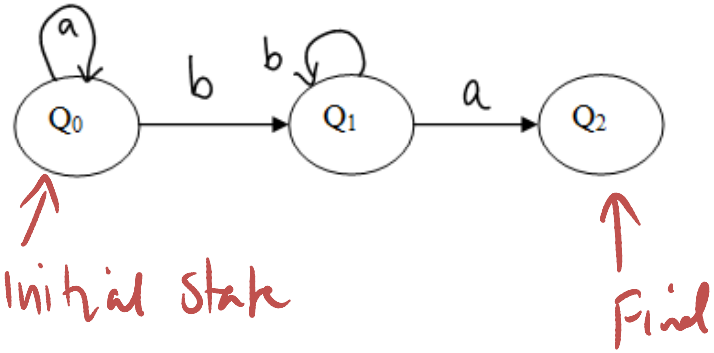
- Formal languages with **formal semantics** generates rules for semantic specifications of programming languages

Automaton

- An **automaton (**or **automata** in plural) is a machine that can recognize valid strings generated by a **formal language**.
- A **finite automata** is a mathematical model of a **finite state machine** (FSM), an abstract model under which all modern computers are built.

Automaton

- A FSM is a machine that consists of a set of finite states and a transition table.

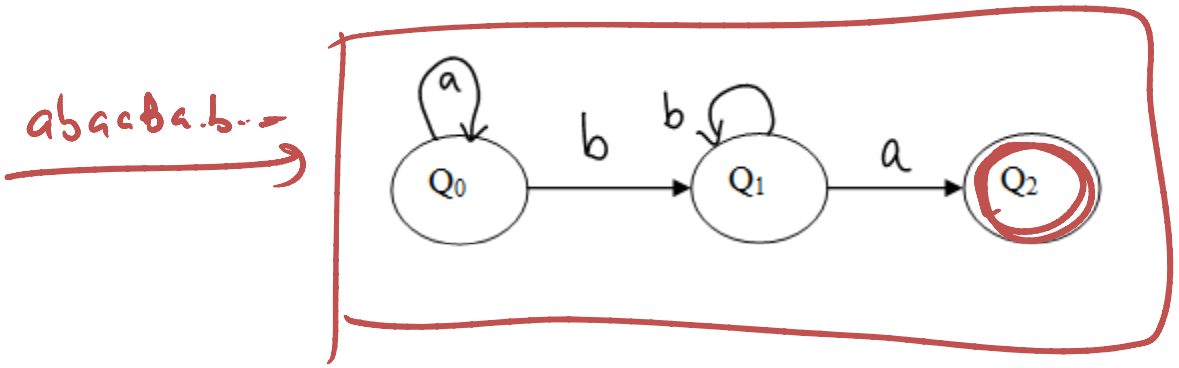


	Q_0	Q_1	Q_2
a	Q_0	Q_2	
b	Q_1	Q_1	

- The FSM can be in any one of the states and can transit from one state to another based on a series of rules given by a transition function.

Example

What does this machine represents? Describe the kind of strings it will accept.



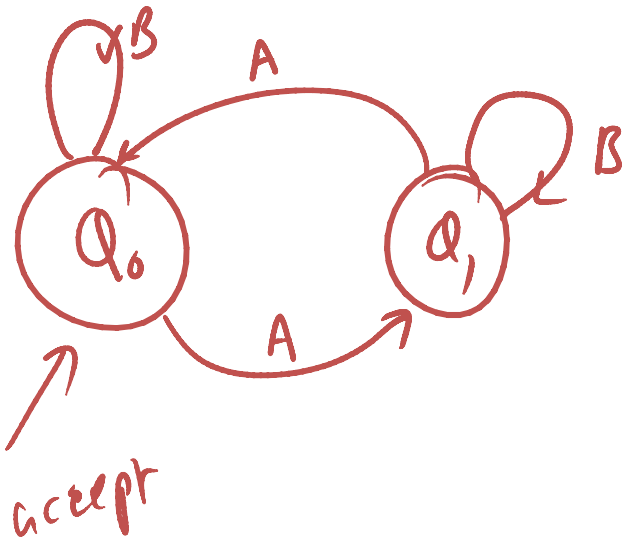
{ba, bba, aabbba, ...}

any string that has at least one b

ends with an a

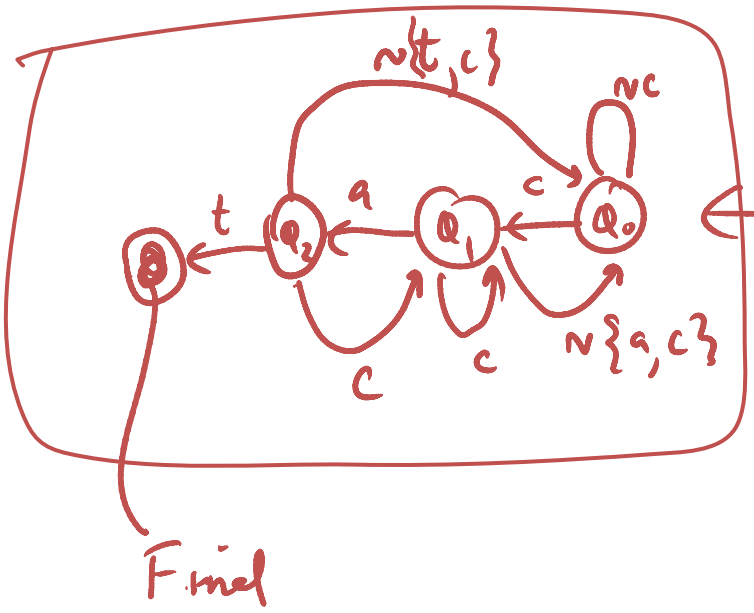
Exercise

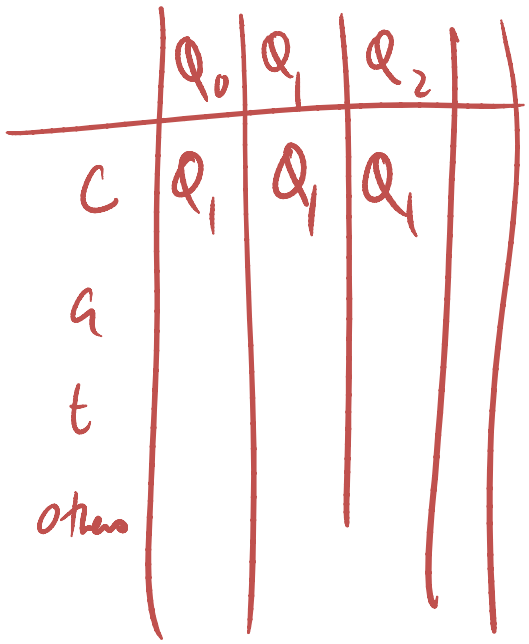
- Draw a FSM that accepts any string with even number of A's. Assume the alphabet is {A,B}



Build a FSM

- Stream: “I love cats”
- Pattern: “cat”





Regular Expressions

Regex versus FSM

- A regular expressions and FSM's are equivalent concepts.
- ***Regular expression is a pattern that can be recognized by a FSM.***
- *Discussion focus limited to Deterministic Finite Automata(DFA)*

Regular Expression

- regex defines a class of patterns
  - Patterns that ends with a "*"
- Regex utilities in unix
  - grep, awk, sed, perl

The grep command

grep

NAME

grep, egrep, fgrep - print lines matching a pattern

SYNOPSIS

grep [options] PATTERN [FILE...]
grep [options] [-e PATTERN | -f FILE] [FILE...]

DESCRIPTION

grep searches the named input FILEs (or standard input if no files are named, or the file name - is given) for lines containing a match to the given PATTERN. By default, grep prints the matching lines.

Source: unix manual

Simple grep examples

- grep “<a href” guna.html > output.txt
- ls | grep “guna”

Regular Expression Grammar

- Regex grammar defines a set of rules for finding patterns. Grammar categories
  - Alternation
  - Grouping
  - quantification

Regular Expression Grammar

- **Alternation**
- The vertical bar is used to describe alternating choices among two or more choices.
  - the notation **a | b | c** indicates that we can choose a or b or c as part of the string.
  - Another example is that **"(c|s)at"** describes the expressions "cat" or "sat". **n**

Regular Expression Grammar

Grouping

Parenthesis can be used to describe the scope and precedence of operators.

In the example above **(c|s)** indicates that we can either begin with c or s but must immediately follow by “at”

c|sat

Regular Expression Grammar

- **Quantification**
- Quantification is the notation used to define the number of symbols that could appear in the string.
- The most common quantifiers are
  - **?, * and +**
  - The **? mark indicates that there is zero or one** of the previous expression.
  - The **"*" indicates that zero or more** of the previous expression can be accepted.
  - The **"+" indicates that one or more** of the previous expression can be accepted.

(ab)? → { ∅ , ab }

(ab)* → { ∅, ab, abab, ababab, ... }

(ab)+ → { ab, abab, ... }

Examples of *, ? , +

a(ab)?b ⟶ ab , aabb

(ab)+b?c ⟶ abc , abbc , ababbc

Other facts

- . matches a single character
- .* matches any string
- [a-zA-Z]* matches any string of alphabetic characters
- [ag].* matches any string that starts with a or g
- [a-d].* matches any string that starts with a,b,c or d
- ^(ab) matches any string that begins with ab. In general, to match all lines that begins with any string use ^string

Finding non-matches

- To exclude a pattern
  - [^class]
  - Eg: [^0-9]

[0-9] – inclusive
[^0-9] – exclusive
[^0-9] — class of thing that does not include 0-9

Group Matches

- grep "<h\([1-4]\)>.*h\([1-3]\)>" filename
  - What patterns match?
- grep "h\([1-4]\).*h\1" filename
  - Back-reference

Character Classes

- \d digit [0-9]
- \D non-digit [^0-9]
- \w word character [0-9a-z_A-Z]
- \W non-word character [^0-9a-z_A-Z]
- \s a whitespace character [\t\n\r\f]
- \S a non-whitespace character [^ \t\n\r\f]

More regex notation

- *{n,m} at least n but not more than m times*

a{2,4} ⟶ aa, aaa, aaaa

- *{n,} – match at least n times*

- {n} – match exactly n times

a{2}(ab)+

More examples of regex

- Find all files that begins with “guna”

grep "^guna" *

- Find all files that does not begins with “guna”

- Find all directories in current folder. Write them to an external file.

ls -l | grep "^d" > out.txt

Summarized Facts about regex

- Two regular expressions may be concatenated; the resulting regular expression matches any string formed by concatenating two substrings that respectively match the concatenated sub expressions.
- Two regular expressions may be joined by the infix operator **| the resulting** regular expression matches any string matching either sub expression

Summarized Facts about regex

- Repetition takes precedence over concatenation, which in turn takes precedence over alternation. A whole sub expression may be enclosed in parentheses to override these precedence rules
- The backreference \n, where n is a single digit, matches the substring previously matched by the nth parenthesized sub expression of the regular expression.
- In basic regular expressions the metacharacters ?, +, {, |, (, and) lose their special meaning; instead use the backslashed versions \?, \+, \{, \|, \(, and \).

grep "a\+"

Text Processing Languages

- awk
  - Text processing language
  - awk '/pattern/' somefile
  - awk '{if ($3 < 1980) print $3, " ",$5,$6,$7,$8}' somefile
- sed
  - A stream editor
  - sed s/moon/sun/ < moon.txt >sun.txt

- Perl
  - A powerful scripting language → Any, Hash, Ref
  - We will discuss this next

exercises

- Download an index.html file from your favorite website
  - Use wget
- Change all URL's www.cnn.com to www.foxnews.com
  - Use sed

Coding Examples