7: Theory of Computation

Introduction to Theoretical CS

Two fundamental questions.

- What can a computer do?
- What can a computer do with limited resources?

General approach. Pentium IV running Linux kernel 2.4.22

- Don't talk about specific machines or problems.
- Consider minimal abstract machines.
- Consider general classes of problems.

Why Learn Theory

In theory . . .

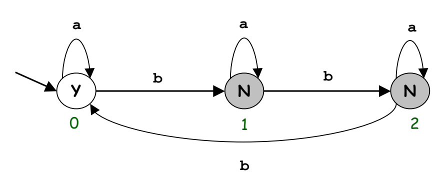
- Deeper understanding of what is a computer and computing.
- Foundation of all modern computers.
- Pure science.
- Philosophical implications.

In practice . . .

- Web search: theory of pattern matching.
- Sequential circuits: theory of finite state automata.
- Compilers: theory of context free grammars.
- Cryptography: theory of computational complexity.
- Data compression: theory of information.

7.1: Regular Expressions and DFAs

`a* | (a*ba*ba*ba*)*`



Pattern Matching Applications

Test if a string matches some pattern.

- Scan for virus signatures.
- Process natural language.
- Search for information using Google.
- Search for markers in human genome.
- Access information in digital libraries.
- Retrieve information from Lexis/Nexis.
- Search-and-replace in a word processors.
- Filter text (spam, NetNanny, Carnivore, malware).
- Validate data-entry fields (dates, email, URL, credit card).

Parse text files.

- Compile a Java program.
- Crawl and index the Web.
- Read in data stored in TOY input file format.
- Automatically create Java documentation from Javadoc comments.

Regular Expressions: Basic Operations

Regular expression.

- Notation to specify a set of strings.

Operation	Regular Expression	Yes	No
Concatenation	aabaab	aabaab	every other string
Logical Or	aa | baab	aa baab	every other string
Replication	ab*a	aa aba abbba	ε ab ababa
Grouping	a(a|b)aab	aaaab abaab	every other string
	(ab)*a	a aba ababa	ε aa abbba

Regular Expressions: Examples

Regular expression examples.

- Notation is surprisingly expressive.

Regular Expression	Yes	No
a* | (a*ba*ba*ba*) multiple of three b's	ε bbb aaa abbbaaa bbbaababbaa	b bb abbaaaa baabbbaa
a | a(a|b)*a begins and ends with a	a aba aa abbaabba	ε ab ba
(a|b)* abba (a|b)* contains the substring abba	abba bbabbabb abbaabba	ε abb bbaaba

Using Regular Expressions

Regular expressions are a standard programmer's tool.

- Built in to Java, Perl, Unix, Python, ...
- Additional operations typically added for convenience.
- Ex: `[a-e]+` is shorthand for `(a|b|c|d|e)(a|b|c|d|e)*`.

Operation	Regular Expression	Yes	No
Any single character	..oo..oo.	bloodroot spoonfood	cookbook choochoo
One or more	a(bc)+de	abcde abcbcde	ade bcde
Character classes	[a-e]+	decade accede	Upper45

Regular Expressions in Java

Ex: pattern match.
- Is `text` in the set described by the `pattern`?

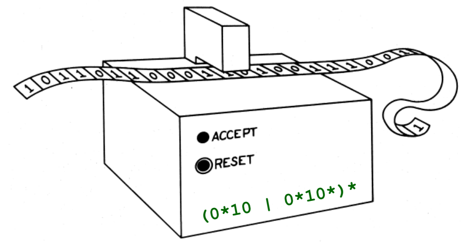
```
public class RE {
   public static void main(String[] args) {
      String pattern = args[0];
      String text    = args[1];
      System.out.println(text.matches(pattern));
   }
}
```

```
                 need help solving crosswords?
% java RE "..oo.oo." bloodroot
true
                       legal Java identifier
% java RE "[$_A-Za-z][$_A-Za-z0-9]*" ident123
true
                      legal email address (simplified)
% java RE "[a-z]+@([a-z]+\.)+(edu|com)" rs@cs.princeton.edu
true
                need quotes to "escape" the shell
```


Solving the Pattern Match Problem

Regular expressions are a concise way to describe patterns.
- How would we implement `String.matches` ?
- Hardware: build a deterministic finite state automaton (DFA).
- Software: simulate a DFA.

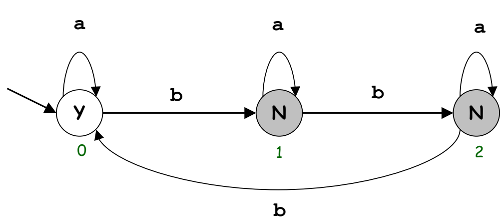
DFA: simple machine that solves the pattern match problem.
- Different machine for each pattern.
- Accepts or rejects string specified on input tape.
- Focus on `true` or `false` questions for simplicity.



Deterministic Finite State Automaton (DFA)

Simple machine with N states.
- Begin in *start state*.
- Read first input symbol.
- Move to new state, depending on current state and input symbol.
- Repeat until last input symbol read.
- *Accept* or *reject* string depending on label of last state.

DFA



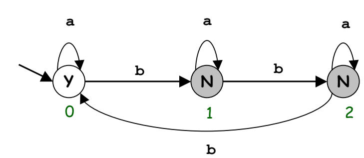
Input

b	b	a	a	b	b	a	b	b

Theory of DFAs and REs

RE. Concise way to *describe* a set of strings.
DFA. Machine to *recognize* whether a given string is in a given set.

Duality: for any DFA, there exists a regular expression to describe the same set of strings; for any regular expression, there exists a DFA that recognizes the same set.



multiple of 3 b's

`(a*ba*ba*ba*)* a*`

multiple of 3 b's

Practical consequence of duality proof: to match regular expression patterns, (i) build DFA and (ii) simulate DFA on input string.

Implementing a Pattern Matcher

Problem: given a regular expression, create program that tests whether given input is in set of strings described.

Step 1: build the DFA.
- A compiler!
- See COS 226 or COS 320.

Step 2: simulate it with given input.
- Easy.

```
State state = start;
while (!CharStdIn.isEmpty()) {
   char c = CharStdIn.readChar()
   state = state.next(c);
}
System.out.println(state.accept());
```


Application: Email Harvester

Harvest email addresses from web for spam campaign.
- User enters name of file and program prints email addresses.

```
import java.util.regex.Pattern;
import java.util.regex.Matcher;
public class EmailHarvester {
   public static void main(String[] args) {
      In in = new In(args[0]);
      String input = in.readAll();
      String regexp = "[a-z]+@([a-z]+\\.)+(edu|com|net|tv)";
      Pattern pattern = Pattern.compile(regexp);
      Matcher matcher = pattern.matcher(input);
      while (matcher.find())
         System.out.println(matcher.group());
   }
}
```

simple email validator (annotation pointing to the regexp)

```
% java EmailHarvester http://www.cs.princeton.edu/courses/cs126/precepts.html
pcalamia@cs.princeton.edu
dgabai@cs.princeton.edu
sgaw@cs.princeton.edu
wayne@cs.princeton.edu
```

OK to enter URL instead of file!

Application: Parsing a Data File

Parsing input files: TOY, Internet movie database, NCBI genome file.

```
LOCUS AC146846 128142 bp DNA linear HTG 13-NOV-2003
DEFINITION Ornithorhynchus anatinus clone CLM1-393H9,
ACCESSION AC146846
VERSION AC146846.2 GI:38304214
KEYWORDS HTG; HTGS_PHASE2; HTGS_DRAFT.
SOURCE Ornithorhynchus anatinus (platypus)
ORIGIN
     1 tgtatttcat ttgaccgtgc tgttttttcc cggtttttca gtacggtgtt agggagccac
    61 gtgattctgt ttgttttatg ctgccgaata gctgctcgat gaatctctgc atagacagct  // a comment
   121 gccgcaggga gaaatgacca gtttgtgatg acaaaatgta ggaaagctgt ttcttcataa
   ...
128101 ggaaatgcga cccccacgct aatgtacagc ttctttagat tg
//
```

```
String  regexp  = "[ ]*[0-9]+([actg ]*).*";
Pattern pattern = Pattern.compile(regexp);
In in = new In(filename);
String line;
while ((line = in.readLine()) != null) {
   Matcher matcher = pattern.matcher(line);
   if (matcher.find()) {
      String s = matcher.group(1).replaceAll(" ", "");
      // do something with s
   }
}
```

just the RE part in parentheses

replace this RE   with this string

Fundamental Questions

Which languages CANNOT be described by any RE?
- Set of all bit strings with equal number of 0s and 1s.
- Set of all decimal strings that represent prime numbers.
- Many more. . . .

How can we extend REs to describe richer sets of strings?
- Context free grammar. ← see COS 320
- Ex: Java language. ← http://java.sun.com/docs/books/jls/second_edition/html/syntax.doc.html

How can we make simple machines more powerful?

Are there any limits on what kinds of problems machines can solve?

Summary

Programmer.
- Regular expressions are a powerful pattern matching tool.
- Implement regular expressions with finite state machines.

Theoretician.
- Regular expression is a compact description of a set of strings.
- DFA is an abstract machine that solves pattern match problem for regular expressions.
- DFAs and regular expressions have limitations.

You. Practical application of core CS principles.

7.2: Turing Machines

Challenge: Design simplest machine that is "as powerful" as conventional computers.

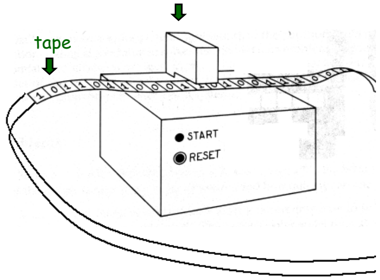
Alan Turing (1912-1954)

Turing Machine: Components

Alan Turing sought the most primitive model of a computing device.

Tape.
- Stores input, output, and intermediate results.
- One arbitrarily long strip, divided into cells.
- Finite alphabet of symbols.

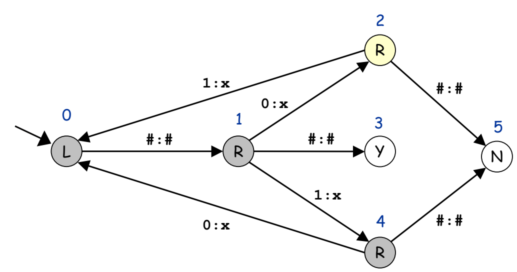
Tape head.
- Points to one cell of tape.
- Reads a symbol from active cell.
- Writes a symbol to active cell.
- Moves left or right one cell at a time.



Turing Machine: Fetch, Execute

States.
- Finite number of possible machine configurations.
- Determines what machine does and which way tape head moves.

State transition diagram.
- Ex. if in state 2 and input symbol is `1` then: overwrite the `1` with `x`, move to state 0, move tape head to left.



Before

...	#	#	x	x	x	1	1	0	#	#	...

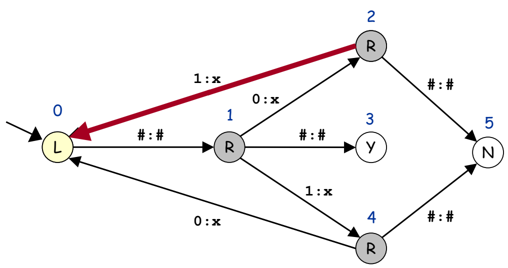
Turing Machine: Fetch, Execute

States.

- Finite number of possible machine configurations.
- Determines what machine does and which way tape head moves.

State transition diagram.

- Ex. if in state 2 and input symbol is `1` then: overwrite the `1` with `x`, move to state 0, move tape head to left.



After

...	#	#	x	x	x	x	1	0	#	#	...

Turing Machine: Initialization and Termination

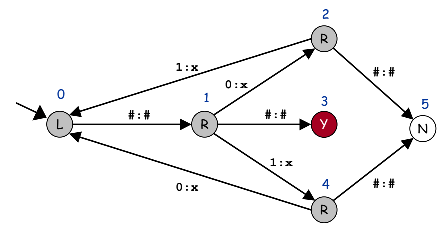
Initialization.

- Set input on some portion of tape.
- Set tape head.
- Set initial state.

...	#	#	0	0	1	1	1	0	#	#	...

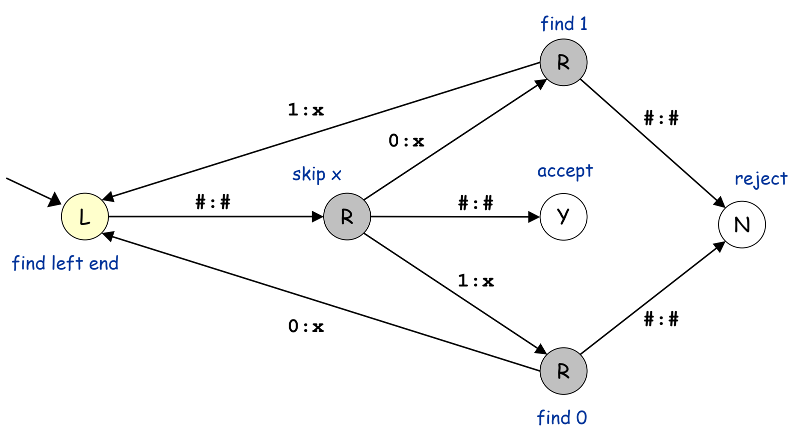
Termination.

- Stop if enter `yes`, `no`, or `halt` state.
- Infinite loop possible.



...	#	#	x	x	x	x	x	x	#	#	...

Example: Equal Number of 0's and 1's



...	#	#	0	0	1	1	1	0	#	#	...

Turing Machine Summary

Goal: simplest machine that is "as powerful" as conventional computers.

Surprising Fact 1. Such machines are very simple.
Surprising Fact 2. Some problems cannot be solved by ANY computer.

↑ next lecture

Consequences.

- Precursor to general purpose programmable machines.
- Exposes fundamental limitations of all computers.
- Enables us to study the physics and universality of computation.
- No need to seek more powerful machines!