7. Theory of Computation

> "In theory there is no difference between theory and practice. In practice there is."
>
> *-Yogi Berra*

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 . . .

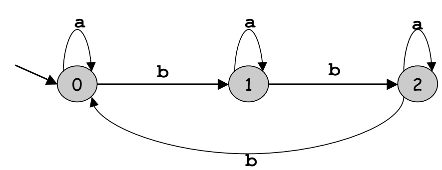
- Deeper understanding of what is a computer and computing.
- Foundation of all modern computers.
- Pure science.
- Philosophical implications.

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.

Regular Expressions and DFAs

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



Pattern Matching Applications

Test if a string matches some pattern.

- Process natural language.
- Scan for virus signatures.
- Search for information using Google.
- 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).
- Search for markers in human genome using PROSITE patterns.

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
Wildcard	`.u.u.u.`	`cumulus` `jugulum`	`succubus` `tumultuous`
Union	`aa | baab`	`aa` `baab`	every other string
Closure	`ab*a`	`aa` `abbba`	`ab` `ababa`
Parentheses	`a(a|b)aab`	`aaaab` `abaab`	every other string
	`(ab)*a`	`a` `ababababa`	ε `abbbaa`

Regular Expressions: Examples

Regular expression. Notation is surprisingly expressive.

Regular Expression	Yes	No
`.* spb .*` contains the trigraph `spb`	`raspberry` `crispbread`	`subspace` `subspecies`

`a* | (a*ba*ba*ba*)*`
multiple of three `b`'s
	`bbb`
	`aaa`
	`bbbaababbaa`
		`b`
		`bb`
		`baabbbaa`

`.*0....` fifth to last digit is `0`	`100011` `98701234`	`111111111` `403982772`

`gcg (cgg|agg)* ctg`
fragile X syndrome indicator
	`gcgctg`
	`gcgcggctg`
	`gcgcggaggctg`
		`gcgcgg`
		`cggcggcggctg`
		`gcgcaggctg`

Generalized 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
One or more	`a(bc)+de`	`abcde` `abcbcde`	`ade` `bcde`
Character classes	`[A-Za-z][a-z]*`	`capitalized` `Word`	`camelCase` `4illegal`
Exactly k	`[0-9]{5}-[0-9]{4}`	`08540-1321` `19072-5541`	`111111111` `166-54-111`
Negations	`[^aeiou]{6}`	`rhythm`	`decade`

Regular Expressions in Java

Validity checking. Is `input` in the set described by the `re`?

```
public class Validate {
   public static void main(String[] args) {
      String re    = args[0];
      String input = args[1];
      System.out.println(input.matches(re));
   }
}
```

powerful string library method

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

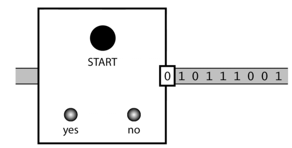

Solving the Pattern Match Problem

Regular expressions are a concise way to describe patterns.

- How would you implement `String.matches` ?
- Hardware: build a deterministic finite state automaton (DFA).
- Software: simulate a DFA.

DFA: simple machine that solves the pattern match problem.

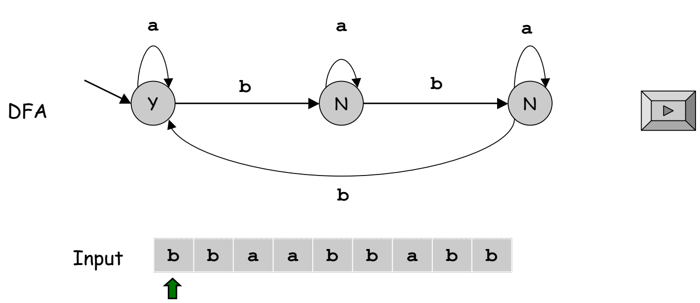
- Different machine for each pattern.
- Accepts or rejects string specified on input tape.
- Focus on `true` or `false` questions for simplicity.



Deterministic Finite State Automaton (DFA)

Simple machine with N states.

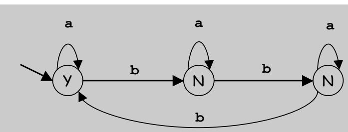
- Begin in *start state*.
- Read first input symbol.
- Move to new state, depending on current state and input symbol.
- Repeat until last input symbol read.
- *Accept* or *reject* string depending on last state.



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.



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

multiple of 3 b's　　　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 RE, 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.

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


Application: Harvester

Harvest information from input stream.

- Harvest patterns from DNA.

```
% java Harvester "gcg(cgg|agg)*ctg" chromosomeX.txt
gcgcggcggcggcggcggctg
gcgctg
gcgctg
gcgcggcggcggaggcggaggcggctg
```

- Harvest email addresses from web for spam campaign.

```
% java Harvester "[a-z]+@([a-z]+\\.)+(edu|com|net|tv)"
      http://www.princeton.edu/~cos126
doug@cs.princeton.edu
dgabai@cs.princeton.edu
mona@cs.princeton.edu
```

email validator (simplified)

Application: Harvester

Harvest information from input stream.

- Use `Pattern` data type to compile regular expression to NFA.
- Use `Matcher` data type to simulate NFA.
- (NFA is fancier but equivalent variety of DFA)

```
import java.util.regex.Pattern;
import java.util.regex.Matcher;
public class Harvester {
   public static void main(String[] args) {
      String re = args[0];
      In in = new In(args[1]);
      String input = in.readAll();
      Pattern pattern = Pattern.compile(re);
      Matcher matcher = pattern.matcher(input);
      while (matcher.find()) {
         System.out.println(matcher.group());
      }
   }
}
```


Limitations of DFA

No DFA can recognize the language of all bit strings with an equal number of 0's and 1's.

- Suppose an N-state DFA *can* recognize this language.
- Consider following input: $\underbrace{000000000}_{N+1\ 0's}\underbrace{111111111}_{N+1\ 1's}$
- DFA must accept this string.
- Some state x is revisited during first N+1 0's since only N states.

0000000011111111
 x   x

- Machine would accept same string without intervening 0's.

000011111111

- This string doesn't have an equal number of 0's and 1's.

Fundamental Questions

Which languages CANNOT be described by any RE?

- Bit strings with equal number of 0s and 1s.
- Decimal strings that represent prime numbers.
- Genomic strings that are Watson-Crick complemented palindromes.
- Many more. . . .

How can we extend REs to describe richer sets of strings?

- Context free grammar (e.g., Java).

Reference: http://java.sun.com/docs/books/jls/second_edition/html/syntax.doc.html

Q. How can we make simple machines more powerful?

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

Application: Parsing a Data File

Ex: parsing an NCBI genome data 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 re = "[ ]*[0-9]+([actg ]*).*";
Pattern pattern = Pattern.compile(re);
In in = new In(filename);
while (!in.isEmpty()) {
   String line = in.readLine();
   Matcher matcher = pattern.matcher(line);
   if (matcher.find()) {                         extract the RE part in parentheses
      String s = matcher.group(1).replaceAll(" ", "");
      // do something with s
   }                                             replace this RE    with this string
}
```


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.

Variations

- Yes (accept) and No (reject) states sometimes drawn differently
- Terminology: Deterministic Finite State Automaton (DFA), Finite State Machine (FSM), Finite State Automaton (FSA) are the same
- DFA's can have output, specified on the arcs or in the states
  - These may not have explicit Yes and No states

Turing Machines

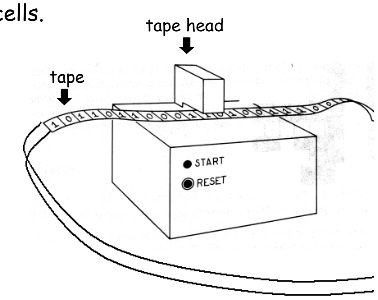
Challenge: Design simplest machine that is "as powerful" as conventional computers.

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.

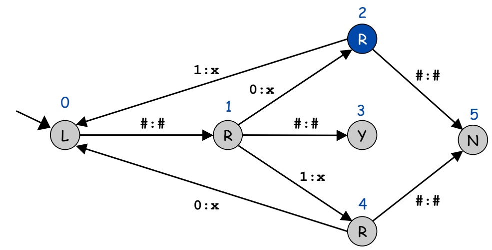


Tape head.
- Points to one cell of tape.
- Reads a symbol from active cell.
- Writes a symbol to active cell.
- Moves left or right one cell at a time.

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.



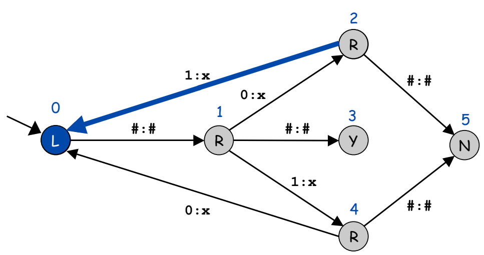
Before

...	#	#	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

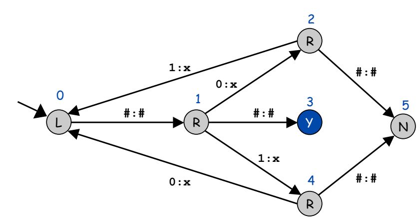
...	#	#	x	x	x	x	1	0	#	#	...

Turing Machine: Initialization and Termination

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

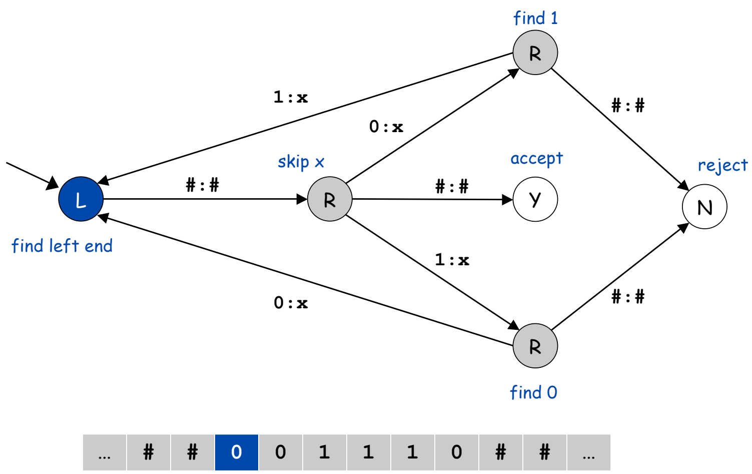
...	#	#	0	0	1	1	1	0	#	#	...

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



Turing Machine Summary

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

Surprising Fact 1. Such machines are very simple: TM is enough!
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!

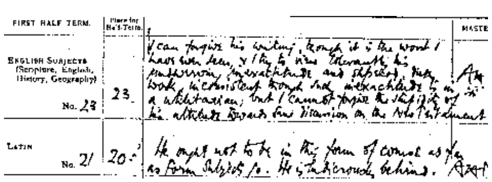
Variations

- Instead of just recognizing strings, TM's can produce output: the contents of the tape
- Instead of Y and N states, TM's can have a plain Halt state

Alan Turing

Alan Turing (1912-1954).

- Father of computer science.
- Computer Science's "Nobel Prize" is called the Turing Award.



Alan's report card at 14.

Alan Turing and his elder brother.