Introduction to the Theory of Computation

Who needs Theory?

Quote from a certain software magazine:

> "Put the right kind of software into a computer,
> and it will do whatever you want it to.
> There may be limits on what you can do with the machines themselves,
> but **there are no limits on what you can do with software**"

This is **TOTALLY WRONG !!**

This part of the module is devoted to describing and explaining the facts that **REFUTE** this claim

Theory of computation

The module contains three basic parts:

1. Automata theory and Turing machines

– Deals with definitions and properties of precise mathematical models of computers and computations

2. Computability theory

– What problems computers can and cannot solve?

3. Complexity theory

– What makes some problems computationally hard and others easy?

£1000 problem

Consider the program **Collatz** (originally suggested by L. Collatz in 1937):

```
c = n;
while (c != 1) {
    if (c % 2 == 0) { c = c / 2; }
    else { c = 3*c + 1; }
}
```

set $c := n$, for $n > 1$
while $c \neq 1$ do the following
if c is even, set $c := c/2$
if c is odd, set $c := 3c + 1$

Nobody knows yet—after 80 years—whether this program always terminates!

The run of Collatz for $n = 57$: 172 86 43 130 65 196 98 49 148 74 37 112 56 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

Thwaites (1996) has offered a £1000 reward for a solution to this algorithmic problem

Paul Erdös also offered $500 for its solution

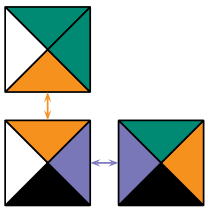
http://en.wikipedia.org/wiki/Collatz_conjecture

Tiling problem

Given a finite set T of tile types, say,

decide whether one can tile an arbitrarily large room,

using only the available tile types,

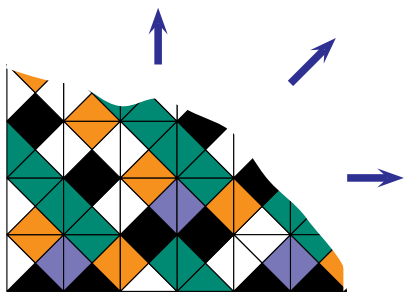


in such a way that the colours

on the touching edges of any two

adjacent tiles are identical

Can you write a **program** which can answer this question for an arbitrary input set T?



$1,000,000 problem

To celebrate mathematics in the new millennium, The Clay Mathematics Institute of Cambridge, Massachusetts, has named seven Prize Problems,

each of them worth of $\$1,000,000$ (one problem was solved by Perelman in 2002-3)

Here is one of them: You have to organise housing accommodations for a group of 400 students. Space is limited and only 100 of the students will receive places in the dormitory. Moreover, the Dean has provided you with a list of pairs of incompatible students, and requested that no pair from this list appear in your final choice.

An obvious algorithm: Choose 100 students out of the given 400 and check whether they satisfy the Dean's constraints. If they do, we are done. Otherwise, choose another set of 100 students, etc.

Trouble is that the total number of ways of choosing 100 students from the 400 applicants is greater than the number of atoms in the known universe! Thus no future civilisation could ever hope to build a supercomputer capable of solving the problem this way.

Problem: is there a better algorithm ?

What is a computation?

Starting with some data (the **input**) and following a step-by-step procedure, a **computation** will produce a result (the **output**)

- Exactly what kinds of steps are allowed in a computation?

A possible approach: a computation is a sequence of steps that can be performed by a **computer**

- Aren't we oversimplifying things? Computers are dealing with tasks far more complicated than merely reading a simple input, doing some work, then producing something and quitting. What about interactive, distributed, real-time embedded systems, multimedia, etc.?
  - **Yes:** modelling always means simplifying
  - **No:** this module will try to convince you about it

Which program is more efficient?

Problem 1. *Design a computer program which, for any input word over the Latin alphabet, outputs* **1** *if the word is of length* $3n$ $(n = 0, 1, 2, \ldots)$*, and outputs* **0***, otherwise.*

Problem 2. *Design a computer program that sorts (in ascending order) and outputs the result for any input sequence* $a_1, a_2, \ldots, a_n$ *of numbers, where* n *is any natural number.*

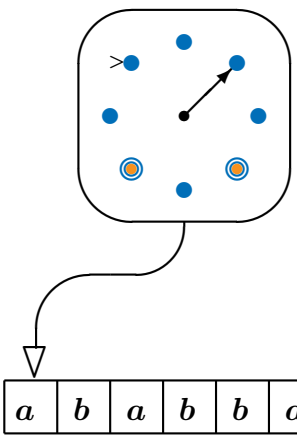
A possible measure: the amount of required memory

- Program 1 requires **constant** memory, regardless of the input length
- Program 2 must memorise the entire input list, and that can be of any **arbitrary** length

Finite automata or finite-state machines

(S.S. Epp. 'Discrete Mathematics with Applications,' Chapter 12)

A **theoretical model** for programs using a constant amount of memory regardless of the input form



Reading head

Finite control device: at any moment can be in one of its **states**. It is hard-wired how it changes from one state to another

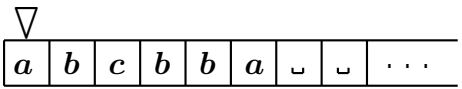
There are some special states:

- one **initial state** (marked with >)
- some **accepting states** (denoted ◉)

Input tape: it is divided into cells; has a leftmost cell, and is as long as we want to the right. Each cell may contain one character of the **input alphabet**

Finite automaton: how it starts

- The finite control device is in its unique **initial state**
- The tape contains a finite word of the input alphabet, the **input**, at its left end. The remainder of the tape contains only **blank** cells
- The reading head is positioned on the leftmost cell of the input tape, containing the first character of the input word.



Finite automaton: how it works

- At regular time intervals, the automaton

- reads one character from the input tape,
- moves the reading head one cell to the right, and
- changes the state of its control device.

- The control device is hard-wired so that the next state depends on
  (1) the previous state, and
  (2) the character read from the tape.

- As the input is finite, at some moment the reading head reaches the end of the input word (that is, the first blank cell).

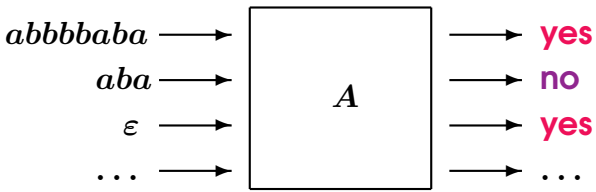
  If at this moment the control device is in one of the **accepting states**, then the input word is **accepted** by the automaton.

  Otherwise, the input word is not accepted (or **rejected**)

Finite automata: what they are used for

- Each finite automaton is a kind of '**recognition**' or '**decision**' device over all possible words of its input alphabet

- Each automaton can be 'tried' on infinitely many input words, and gives a **yes**/**no** answer each time

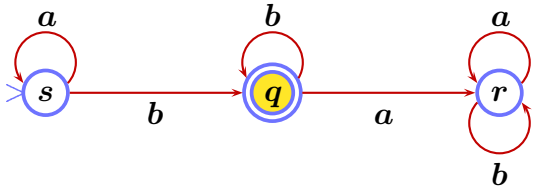


State transition diagrams

A convenient way to represent finite automata is a **state transition diagram**, a directed graph, where nodes represent states and arrows are labelled with symbols from the input alphabet. If in state q and reading input a the automaton changes its state to r, the arrow from q to r is labelled by a.

Accepting states are circled, and the initial state is indicated by $>$

Automaton A_1:



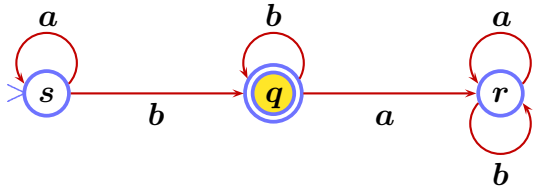
How automaton A_1 works:

- **Input:** $abba$
- **Computation:** $(s, abba),\ (s, bba),\ (q, ba),\ (q, a),\ (r, \varepsilon)$

$\rightsquigarrow$ word $abba$ is **not accepted**

More on how automaton A_1 works

Automaton A_1:



- **Input 2:** $bbaaa$

  Computation: $(s, bbaaa),\ (q, baaa),\ (q, aaa),\ (r, aa),\ (r, a),\ (r, \varepsilon)$

  $\rightsquigarrow$ word $bbaaa$ is **not accepted**

- **Input 3:** $aabb$

  Computation: $(s, aabb),\ (s, abb),\ (s, bb),\ (q, b),\ (q, \varepsilon)$

  $\rightsquigarrow$ word $aabb$ is **accepted**

- **Input 4:** bbb

  Computation: $(s, bbb),\ (q, bb),\ (q, b),\ (q, \varepsilon)$ $\rightsquigarrow$ word bbb is **accepted**

- **Input 5:** ε

  Computation: (s, ε) $\rightsquigarrow$ word ε is **not accepted**

Deterministic Finite Automaton (DFA)

Formal definition:

A **deterministic finite automaton (DFA)** is a quintuple $A = (Q, \Sigma, \delta, s, F)$ where

- Q is a finite set of **states**
- Σ is a finite set, the **input alphabet**
- $s \in Q$ is the **initial state**
- $F \subseteq Q$ is the set of **favourable** (or **accepting**) **states**
- $\delta : Q \times \Sigma \to Q$ is a function, the **transition function**

  If A, being in state $q \in Q$, has read the symbol $a \in \Sigma$, it enters state $q' = \delta(q, a)$

Transition table

A convenient way of defining the transition function of a DFA:

Automaton A_1: $Q = \{s, q, r\}$, $\Sigma = \{a, b\}$, initial state: s, $F = \{q\}$

δ	a	b
s	s	q
q	r	q
r	r	r

Observe that, for every 'cell' in the transition table,

there is a **unique** state to put in.

In other words, the pair

(current state, symbol read)

uniquely determines the next state

That is why DFAs are called **deterministic**

Configurations of DFAs

Let $A = (Q, \Sigma, \delta, s, F)$ be a DFA.

– A **configuration** of A is any pair of the form

(state in Q, word over Σ)

For example: $(q, baaa)$, $(s, aabb)$, (r, ε)

– Configuration (q, w) **yields** configuration (q', w') **in one step**

if $w = \sigma w'$ and $\delta(q, \sigma) = q'$

E.g., (s, bba) yields in one step (q, ba), (q, aaa) yields in one step (r, aa)

Observe that every configuration **uniquely determines** the configuration it yields in one step

Computations of DFAs

– The **computation of A on input word w** is the **unique** sequence of configurations

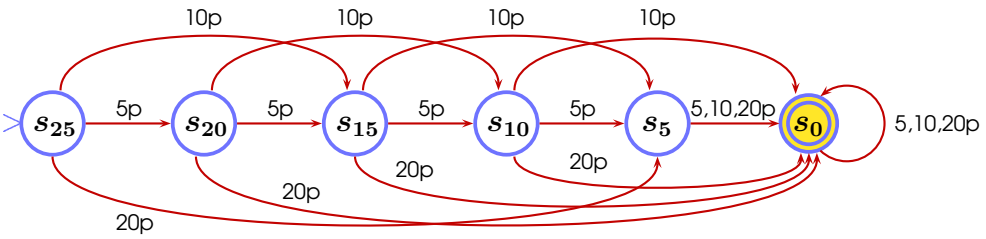
$$(q_1, w_1), (q_2, w_2), \ldots, (q_k, w_k)$$

such that

- $(q_1, w_1) = (s, w)$ where s is the initial state of A
- w_k is the empty word ε, and
- every (q_i, w_i) yields (q_{i+1}, w_{i+1}) in one step

– A word w is **accepted** by DFA A if the computation of A on input w ends up with a configuration (q, ε) for some favourable state q.

Example A_2: vending machine



$Q = \{s_{25}, s_{20}, s_{15}, s_{10}, s_5, s_0\}$, initial state $= s_{25}$,

$\Sigma = \{5p, 10p, 20p\}$, $F = \{s_0\}$

Word: any sequence of $5p$, $10p$ and $20p$ coins

Accepts: all words such that the sum of the coins is $\geq 25p$

Languages and DFAs

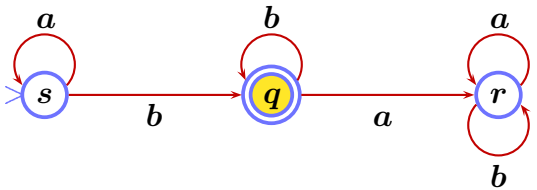
DFAs accept certain words, while may reject others.

If we collect all words accepted by a DFA A, we obtain the language of A

The **language $L(A)$ of a DFA** $A = (Q, \Sigma, \delta, s, F)$ is

$$L(A) = \text{the set of all the words over } \Sigma \text{ that } A \text{ accepts}$$
$$= \{w \in \Sigma^* \mid w \text{ is accepted by } A\}$$

DFA A_1 revisited



We already know:

- accepted: $aabb$, bbb
- rejected: $abba$, $bbaaa$, ε

$L(A_1)$ = all the words starting with a (possibly empty) string of a followed by a nonempty string of b

$= \{a^n b^m \mid n = 0, 1, 2, \ldots, \quad m = 1, 2, \ldots\}$

How to find the language of a finite automaton

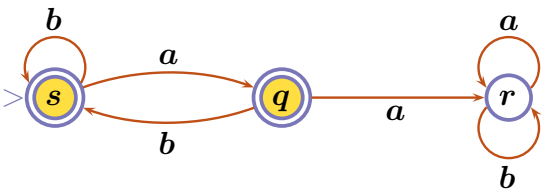
(1) experiment with words

(2) come up with a language

(3) test it

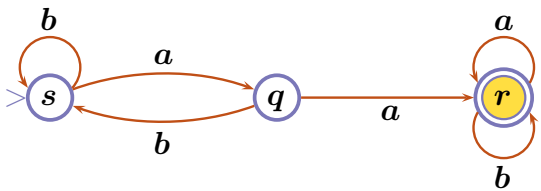
(4) revise the suggested language if necessary, then go to step (3)

Example: DFA A_3



$L(A_3)$ = all the words that do not contain two consecutive a

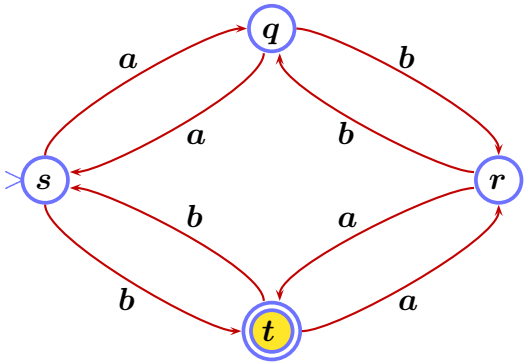
Example: DFA A_4



$L(A_4)$ = all the words that contain two consecutive a

Observe that $L(A_4) = \Sigma^* - L(A_3)$

Example: DFA A_5



$L(A_5)$ = all the words that contain an even number of a and an odd number of b

DFAs vs. languages

(1) So far we have discussed how to determine the language of a DFA

$$A \rightsquigarrow L(A)$$

$$\text{program} \rightsquigarrow \text{specification}$$

(2) The 'reverse' task:

> given a language L,
> design a DFA A such that the language $L(A)$ of A is L

$$L \rightsquigarrow A$$

$$\text{specification} \rightsquigarrow \text{program}$$

Designing finite automata

Problem: *Design a DFA A such that $L(A)$ consists of all strings of 0's and 1's ending with 11*

We need to determine 'what to remember' while reading the input string

- It is *not* appropriate to identify all strings that do not end in 11. E.g. suppose our DFA does not distinguish between two strings ending in 01 and 00 and the next input symbol is 1. Then the DFA would not be able to distinguish between the two new strings, which end in 11 and 01, respectively.
- We do not need to distinguish between two strings ending in 00 and 10, or the string 1 and any string ending in 01. In both cases no matter what the next symbol is, the two resulting strings will have the same last two symbols.
- The two strings 0 and ε can be identified with all other strings ending in 0: we need at least two more input symbols to end with 11

Designing finite automata (cont.)

So it is sufficient to remember three 'states' of the current string:

- The string does not end in $\mathbf{1}$. (Either it is ε or it ends in $\mathbf{0}$.)
  $\rightsquigarrow$ initial state s
- The string is $\mathbf{1}$ or ends in $\mathbf{01}$. $\rightsquigarrow$ state q
- The string ends in $\mathbf{11}$. $\rightsquigarrow$ favourable state p