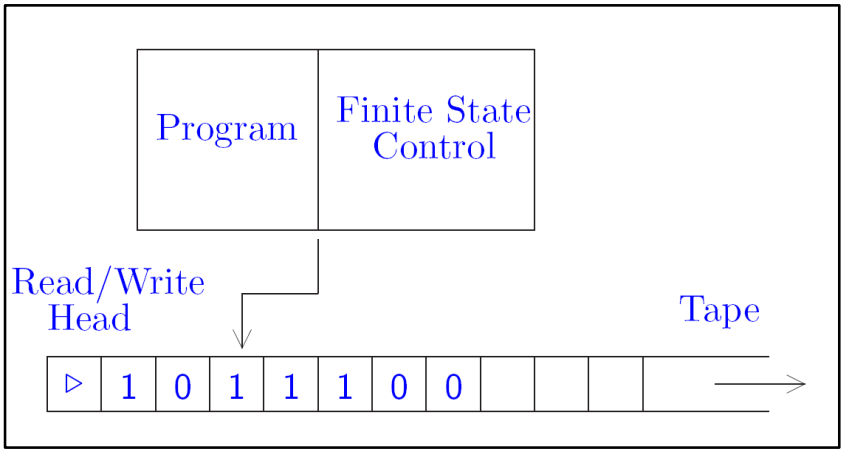
Lectures 5-6

Introduction to computer science

MODELS FOR COMPUTATION: TURING MACHINES



Finite state control: consists of finite set of internal states q_1, …, q_m and special states q_s and q_h which are the starting state and halting state, respectively.

Tape: one-dimensional object which stretches to infinity in one direction and consists of the tape squares. The tape squares each contain one symbol drawn from some alphabet .

Read/Write Head: identifies a square on the tape which is being accessed by a machine.

Program: finite ordered list of program lines in the form <q, x, q', x', s>.

Turing machine: How does it work?

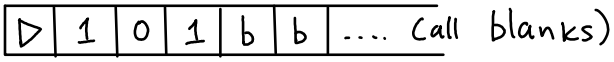
1) The Turing machine looks through the lines of the program searching for a line <q, x ,…>, where q is the current state of the machine and x is the symbol being read on a tape.

2) If it can find such a line it changes the state of the machine to q', overwrites the symbol on the tape square to x' and moves the read/write head by **s** tape squares.

3) If it can not find such a line the internal state of the machine is changed to q_h, machine halts operation and whatever is one the tape is an output.

Turing machine: Example

- Internal states: q_1, q_2, q_3, q_h, q_s.
- Alphabet: ▷ (marks left hand edge), 0, 1 and blank spaces (designated in program as b).
- Tape initially contains binary number x followed by all blanks.

  Example: 

- Program:

$\langle q_s, \triangleright, q_1, \triangleright, +1 \rangle$

$\langle q_1, 0, q_1, b, +1 \rangle$

$\langle q_1, 1, q_1, b, +1 \rangle$

$\langle q_1, b, q_2, b, -1 \rangle$

$\langle q_2, b, q_2, b, -1 \rangle$

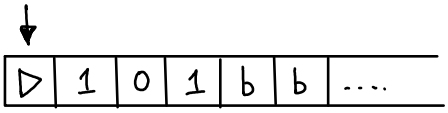
$\langle q_2, \triangleright, q_3, \triangleright, +1 \rangle$

$\langle q_3, b, q_h, 1, 0 \rangle$

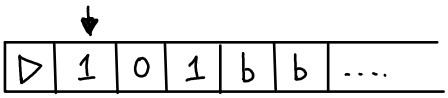
Class exercise: What does this this program do (use example above)? $f(x) = ?$

1. $\langle q_s, \triangleright, q_1, \triangleright, +1 \rangle$ move head by +1 square

initial state — q_s; symbol on tape — ▷; change state to q_1 — q_1; keep tape square the same — ▷

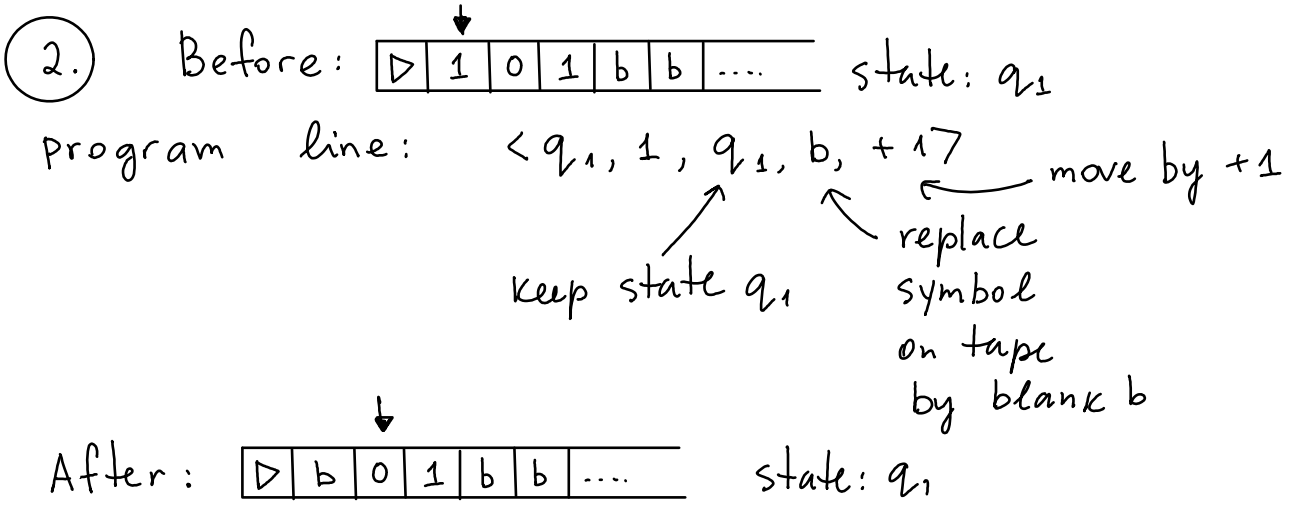
Before: 

State: q_s

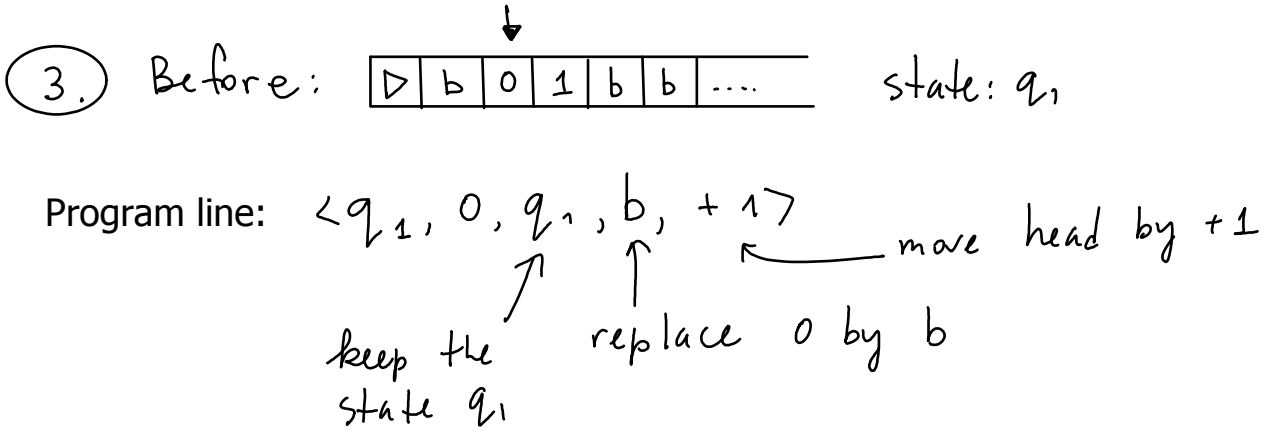
After: 

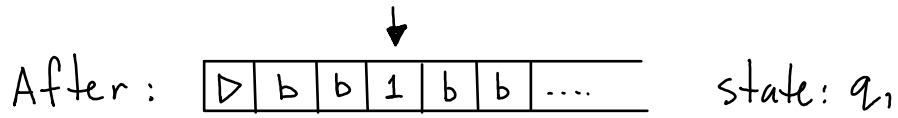
state: q_1

Look for line in program starting with $\langle q_1, 1, \ldots \rangle$



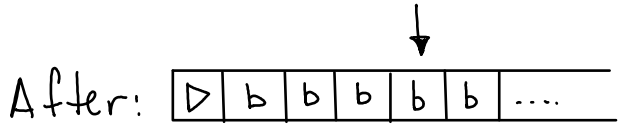
Look for program line starting with $\langle q_1, 0, \ldots \rangle$



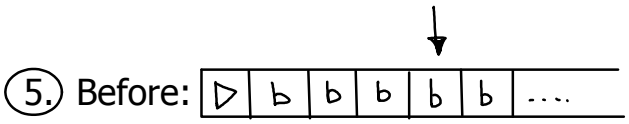


4. Now use the same program line as in step 2

$\langle q_1, 1, q_1, b, +1 \rangle$



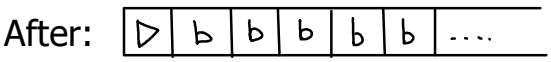
Look for program line: $\langle q_1, b, \ldots$



Program line: $\langle q_1, b, q_2, b, -1 \rangle$

move back one square

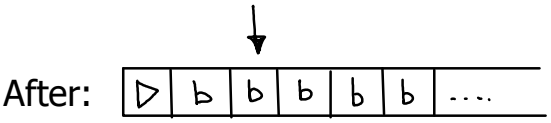
change state to q_2



Look for program line starting from $\langle q_2, b, \ldots \rangle$.

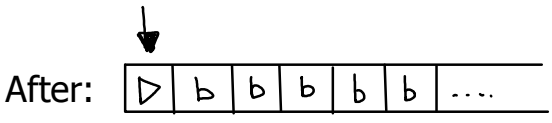
6. Program line $\langle q_2, b, q_2, b, -1 \rangle$

just move back one square, change nothing else



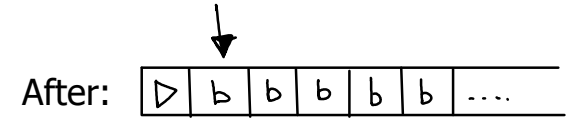
7 and 8

Same program line as in step 6: keep moving back until the end of the tape is reached.



Look for line starting with $\langle q_2, \triangleright, \ldots \rangle$

9. Program line: $\langle q_2, \triangleright, q_3, \triangleright, +1 \rangle$



Look for line starting with $\langle q_3, b, \ldots \rangle$

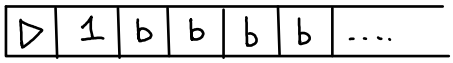
(10) Program line: $\langle q_3, b, q_h, 1, 0 \rangle$

do not move head

overwrite b, put one

change state to halt state

After:



state: q_h

Since the present state of the machine is a halt state, the calculation ends.

Result: $f(x) = 1$.

Your can see from the program that this it true for any x since the program will just keep erasing all 0 and 1, then comes back to the beginning of the tape and writes "1".

Church-Turing thesis:

The class of functions computable by a Turing machine corresponds exactly to the class of functions which we would naturally regard as being computable by an algorithm.

The thesis asserts equivalence between a rigorous mathematical concept, i.e. function computable by the Turing machine and the intuitive concept what it means for a function to be computable by an algorithm. No evidence to the contrary has been found.

- Quantum computers also obey Turing thesis, the difference is in efficiency.
- There are different versions of the Turing machine: multi-tape machines, introduction of the randomness in the model.
- There exists a Universal Turing machine which can simulate any other Turing machine.

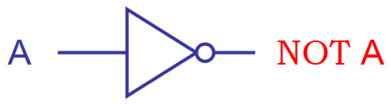
Models for computation: Circuit model

Circuits are made of **wires** which carry information and **gates** which perform simple computational tasks.

Assume that **no loops are allowed**: acyclic model of computation.

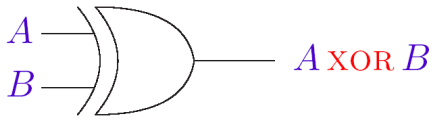
Logic gate: a function $f: \{0, 1\}^k \rightarrow \{0, 1\}^l$ from k input bits to l output bits.

Example: classical not gate, which is the only non-trivial single bit classical gate.

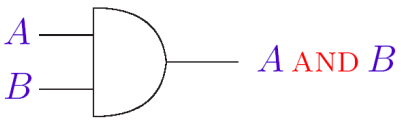


A	NOT A
0	1
1	0

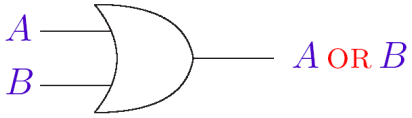
More classical logic gates:



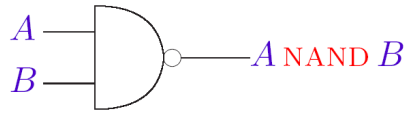
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0



A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

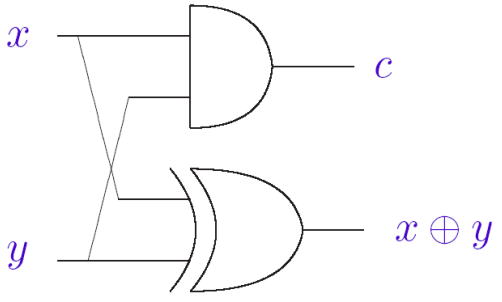


A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1



A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

Class exercise: What does this gate do?



x	y	$x \oplus y$	c
0	0		
0	1	?	
1	0		
1	1		

x	y	$x \oplus y$	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Elements of universal circuit construction

- Wires: preserve the states of the bits
- Ancilla bits prepared in standard state
- The FANOUT operation, which takes a bit and make a copy of it.
- The crossover operation, which interchanges the value of two bits
- The AND, XOR, and NOT gates

The analysis of computational problems

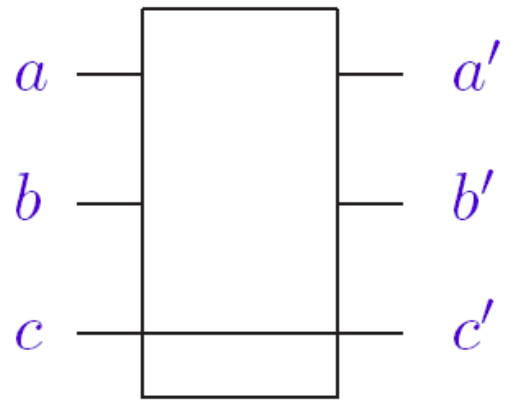
- What is a computational problem?
- How may we design algorithms to solve a computational problem?
- What are the minimal resources required to solve a given computational problem?
- Resources: time, space, and energy.
- Can we classify the problems according to the resource requirements needed to solve them?

Computational complexity

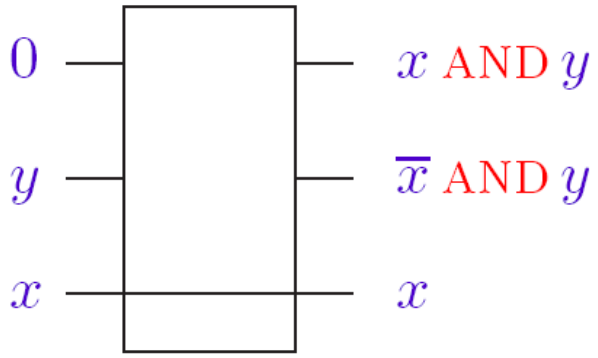
- Computational complexity is the study of the time and space resources required to solve computational problems.
- Task: prove lower bounds on the resources required by the best possible algorithm for solving a problem.
- Suppose that the problem is specified by giving n bits as an input. Chief distinction: problems which can be solved using the resources which grow polynomial in n and problems which grow faster than any polynomial in n.
- The problem is regarded as **easy, tractable or feasible** if an algorithm for solving the problem using polynomial resources exists, and as **hard, intractable or infeasible** if the best possible algorithm requires exponential resources.

Reversible computation: Fredkin gate

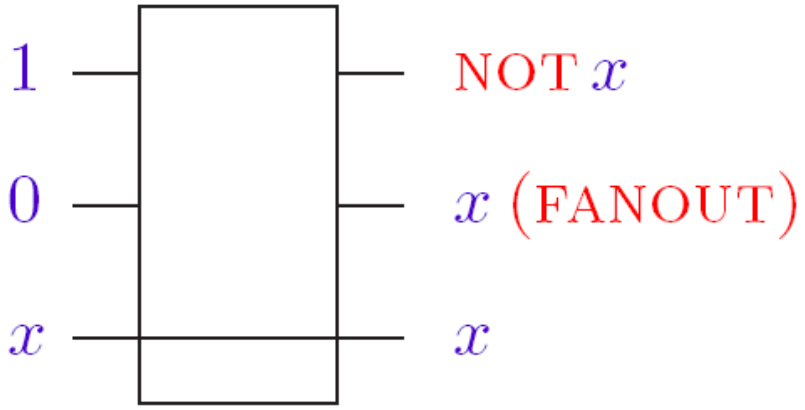
Bits a and b are swapped if control bit c is 1.



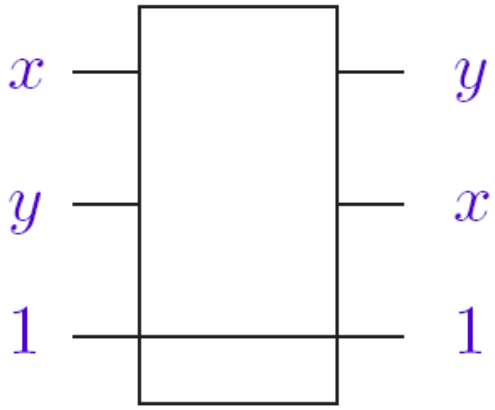
a	b	c	a'	b'	c'
0	0	0	0	0	0
0	1	0	0	1	0
1	0	0	1	0	0
1	1	0	1	1	0
0	0	1	0	0	1
0	1	1	1	0	1
1	0	1	0	1	1
1	1	1	1	1	1



a	b	c	a'	b'	c'
	y	x	xy	$\overline{x}y$	x
0	0	0	0	0	0
0	1	0	0	1	0
0	0	1	0	0	1
0	1	1	1	0	1



a	b	c	a'	b'	c'
		x	$\overline{x}$	x	x
1	0	0	1	0	0
1	0	1	0	1	1



a	b	c	a'	b'	c'
x	y		y	x	
0	0	1	0	0	1
0	1	1	1	0	1
1	0	1	0	1	1
1	1	1	1	1	1