Introduction

Computer science is no more about computers than astronomy is about telescopes.
Edsger Dijkstra

Computer science is the study of computation.

Theory of computation studies mathematical models of computation.

In mathematical terms computation Is accepting or rejecting strings of symbols.

Major topics in Theory of Computation

Computational Model	Does it have memory?	What problems does it solve?
Finite State Machine	No	Automatic door controller
Pushdown Automaton	Limited	Language parsing
Turing Machine	Unlimitted	Any algorithm

Computability

- Which problems are solvable (algorithms exist to solve them)?
- Are there problems that cannot be solved by any computer?
- Any algorithm is equivalent to a Turing Machine algorithm.
- If there is no Turing Machine that solves a problem, then the problem is unsolvable.
- Determine If any computer program terminates on any input (halting problem).

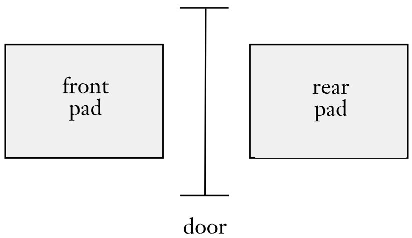
Complexity

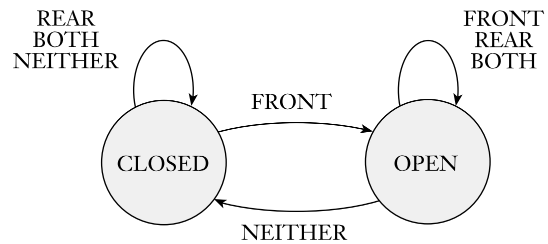
- How fast can a problem be solved?
- Polynomial (P)
- Nondeterministic polynomial (NP)
- P = NP?

Introduction

Finite State Machines (Finite Automata) and Regular Languages

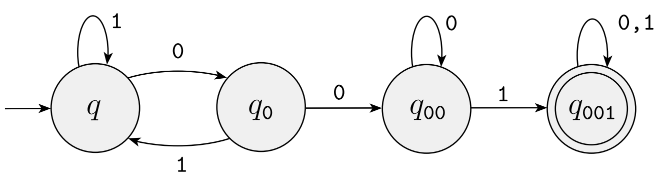
Automatic door controller

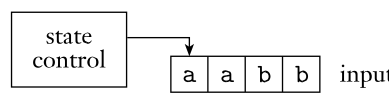


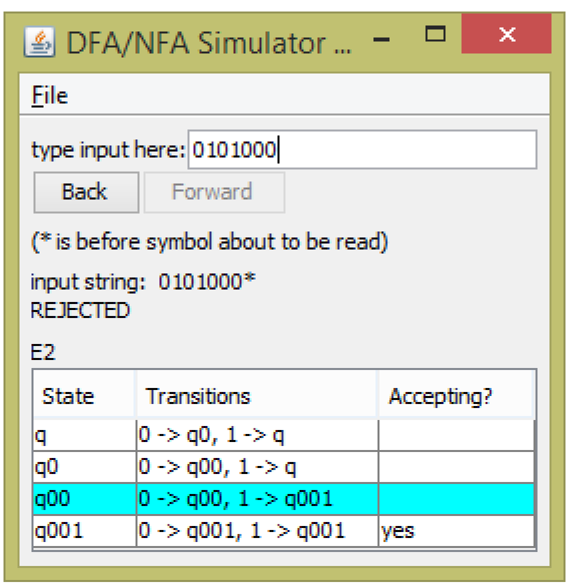


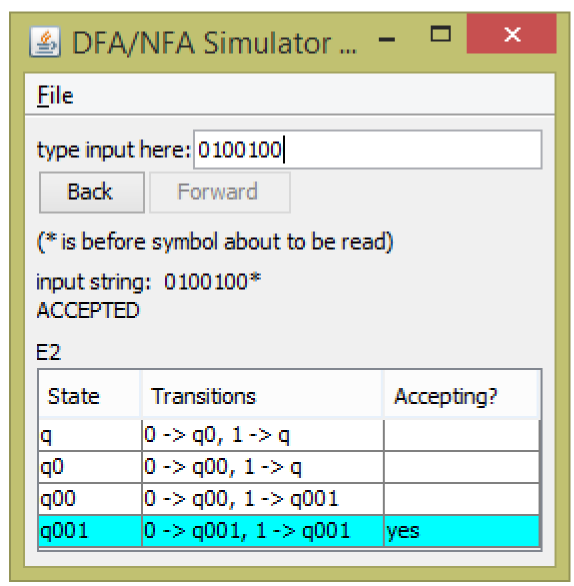
Input String	NEITHER	REAR	NEITHER	BOTH	FRONT	NEITHER	. . .
Output (sate)	CLOSED	OPEN	CLOSED	OPEN	OPEN	CLOSED	. . .

Accepting strings of {0, 1} that contain 001 as a substring









Regular Expression: `(0|1)*001(0|1)* Matching 0100100 -> true, 0101000 -> false`

Introduction

Context-Free Languages (CFL) and Pushdown Automata (PDA)

Parsing an arithmetic expression with variables a, b

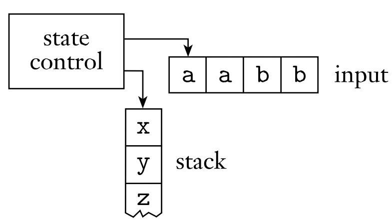
CF Grammar

S → S+S
S → S-S
S → S*S
S → S/S
S → (S)
S → N
N → a
N → b

Derivation of (a+b)/a

Rule	Result
Start	S
S → S/S	S/S
S → N	S/N
N → a	S/a
S → (S)	(S)/a
S → S+S	(S+S)/a
S → N	(S+N)/a
N → b	(S+b)/a
S → N	(N+b)/a
N → a	(a+b)/a

Pushdown Automaton (PDA)

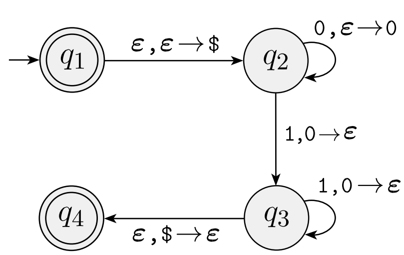


Recognize strings containing 0s followed by the same number of 1s: $\{0^n1^n \mid n \geq 0\}$

CF Grammar

S → 0S1
S → ε

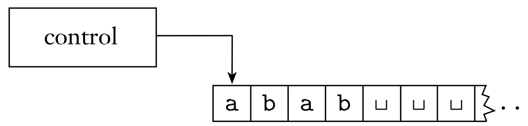
PDA



Introduction

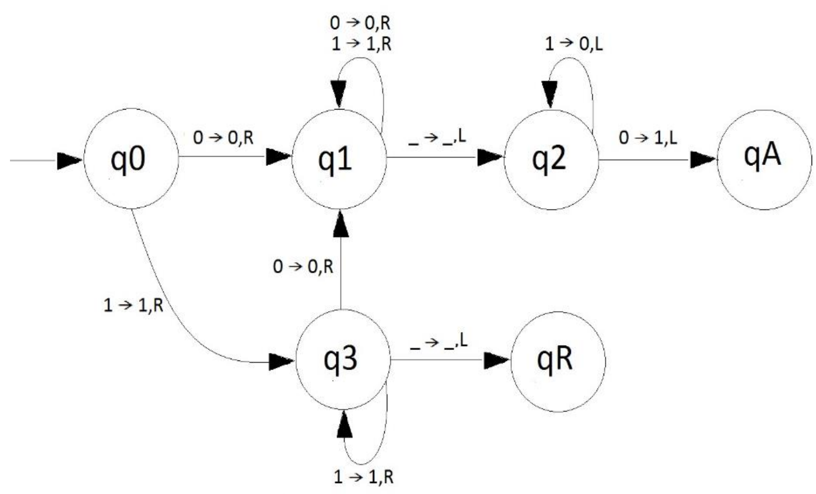
Turing Machines

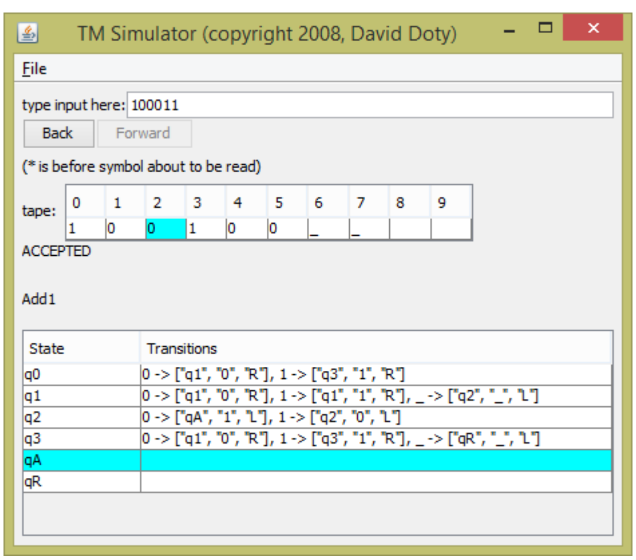
Can implement any algorithm => If an algorithm exists then there is a Turing machine that implements it.

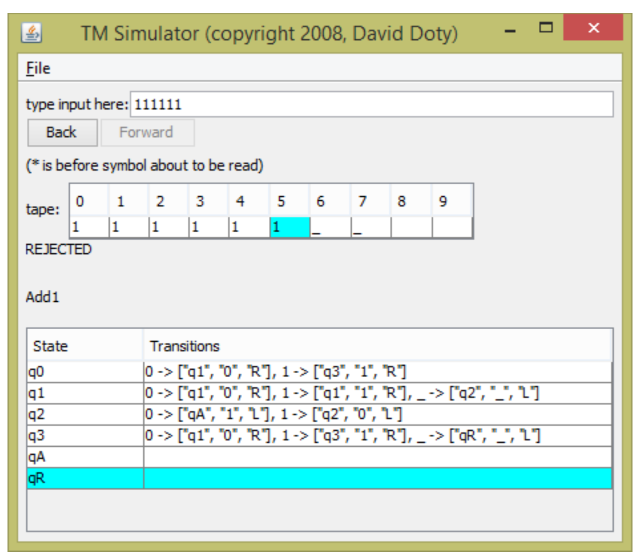


Add 1 to a binary number, reject input 11...1 (overflow)

```
states:
q0,q1,q2,q3,qA,qR
input alphabet: 0,1
tape alphabet:  0,1,_
start state:    q0
accept state:   qA
reject state:   qR
delta:          q0,0 -> q1,0,R
                q0,1 -> q3,1,R
                q1,0 -> q1,0,R
                q1,1 -> q1,1,R
                q1,_ -> q2,_,L
                q2,1 -> q2,0,L
                q2,0 -> qA,1,L
                q3,0 -> q1,0,R
                q3,1 -> q3,1,R
                q3,_ -> qR,_,L
```







Introduction

Computability

Hilbert's tenth problem: Devise an algorithm that tests whether a polynomial has an integral root (unsolvable, no algorithm exists).

Church-Turing Thesis: Intuitive notion of algorithms equals Turing machine algorithms.

Halting Problem: Determine whether a Turing machine halts (by accepting or rejecting) on a given input.

Theorem (Turing, 1936). The halting problem is undecidable.
Idea: Self reference ("This statement is false" – true or false?)
Proof:

- Assume the existence of a function halt(f, x) that solves the problem.
- Create a function test(f) that goes into an infinite loop if f(f) halts and halts otherwise.
- Call test() with *itself* as argument.
- If test(test) halts, then test(test) goes into an infinite loop.
- If test(test) does not halt, then test(test) halts.
- *Reductio ad absurdum.*

⇨ halt(f,x) cannot exist.

Example: *test(test) halts or does not halt?*

```
public boolean halt(String f, String x)
{
if ( /* f(x) halts */ ) return true;
else return false;
}

public void test(String f)
{
if (halt(f, f))
while (true) { } // infinite loop
}
```

Introduction

Complexity

Problems solved in Polynomial time (P)

- Is there a path between two nodes in a directed graph?
- Are two integers relatively prime?

NP problems

- Clique
- Subset Sum

P versus NP

- P = the class of languages for which membership can be *decided* quickly.
- NP = the class of languages for which membership can be *verified* quickly.

NP-complete problems

- SAT problem

P=NP?