Finite Automata

These allow us to recognise whether a given string belongs to a given language

1 Nondeterministic Finite Automata

A nondeterministic finite automaton (NFA) consists of

- A finite set S of states
- The input alphabet Σ (the set of input symbols)
- A start state $s_0 \in S$ (or initial state)
- A set F of final states (or accepting states)

Often we represent an NFA by a transition graph

- Nodes are possible states
- Edges are directed and labelled by a symbol from $\Sigma \cup \{\epsilon\}$
- The same symbol can label edges from a state s to many different other states

Note that if a symbol is not defined at a state and you read it, then it rejects

You can move straight along a node represented by ϵ

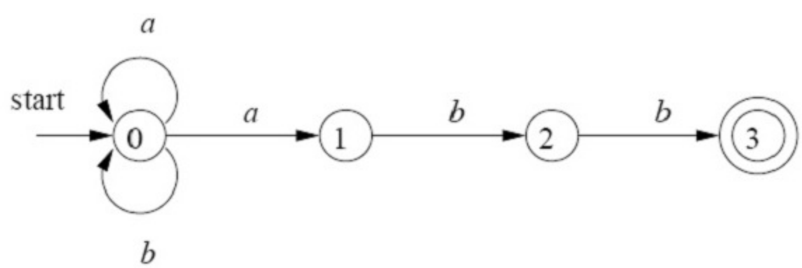
1.1 Representation

The accepting states are represented by double circles
For it ot be accepting there needs to be a given route to the accepting state, this is why there is two options for a coming out of 0.

$$\Sigma = \{a, b\}$$

$$s_0 = 0$$

$$F = \{3\}$$



Alternative representation is a transition table

- Rows → states
- Columns → symbols in $\Sigma \cup \{\epsilon\}$
- Entries → Transitions between states

STATE	a	b	ϵ
0	$\{0,1\}$	$\{0\}$	$\emptyset$
1	$\emptyset$	$\{2\}$	$\emptyset$
2	$\emptyset$	$\{3\}$	$\emptyset$
3	$\emptyset$	$\emptyset$	$\emptyset$

Advantage of transition table: more visible transitions
Disadvantage of transition table: needs more space than the transition graph
This accepts the language:

$$(a|b)^*abb$$

1.2 Acceptance of NFA

An NFA accepts an input string x if there exists a path that:

- Starts at the start state s_0
- Ends at one of the accepting states in F
- Concatenation of the symbols on its edges gives exactly x

A language accepted (or defined) by an NFA:

- The set of strings that this NFA accepts

2 Deterministic Finite Automata

A deterministic finite automaton (DFA) is a special case of a NFA, where:

- No edge is labelled by the empty string ϵ
- For each state s and each input symbol a, there is exactly one edge out of s labelled with a. If in a state with a certain letter, there is exactly one choice, so not 0.

A direct algorithm to decide whether a given string x is accepted by a DFA:

- Start at the start state s_0
- Iteratively follow the edges labelled by the characters of x
- Check whether you reach a final state when x ends:
  - If yes, then the DFA accepts x
  - Otherwise not
- All of this meaning, follow the path guided by the arrows, see if you are in an accepting state at the end

You can label a state with $\varnothing$ to represent a rejecting state

3 NFA vs DFA

Theorem 1 *NFAs accept exactly the regular languages (i.e. the regular expressions)*

Therefore, simulation of an NFA can be used in the lexical analyser to recognise strings, identifiers etc

However the simulation of NFAs is not straightforward

- Many alternative outgoing edges from a state
- Transitions labelled with ϵ are possible

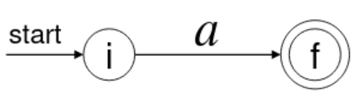
Theorem 2 *NFAs accept exactly the same languages as DFAs*

i.e. for every NFA, we can construct an equivalent DFA

4 From regular expressions to NFA

Our aim: given a regular expression r, construct a NFA that accepts r

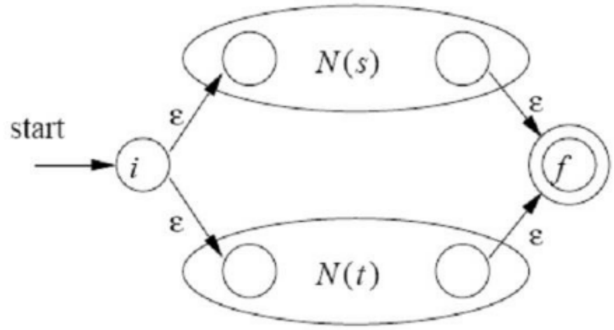
Recursive construction



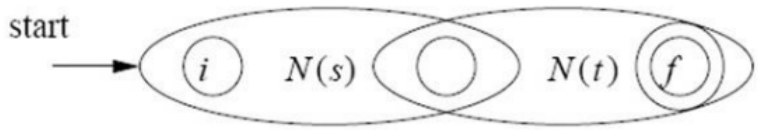
For any symbol $a \in \Sigma \cup \{\epsilon\}$

For any two regular expressions s and t with NFAs N(s) and N(t).

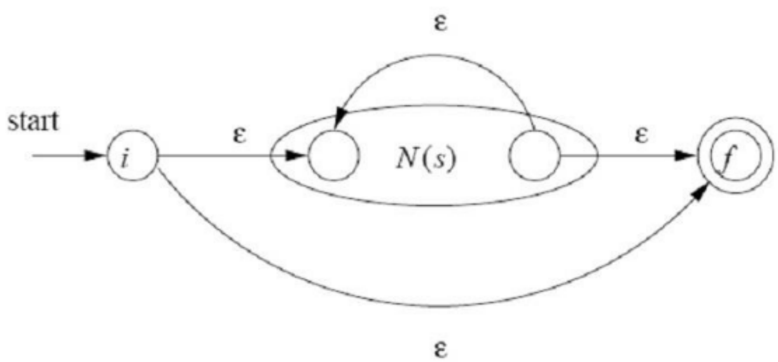
If $r = s|t$



if $r = st$, then:



If $r = s^*$, then



5 From NFA to DFA

Our aim: Given an NFA, construct a DFA that accepts the same regular language. A DFA can be used directly as an automatic string/identifier recogniser.

The main idea is that each state of the constructed DFA corresponds to a rest of states in the NFA

Recursive construction of the DFA, after reading (any) input $a_1a_2 \dots a_k$ the DFA is in the state that corresponds to the set of states that the NFA reaches when reading the same input.

6 Extensions of DFA

A context free language can be recognised by a push-down automaton (PDA). This is exactly the same as an NFA, with the addition of a stack

7 Push-Down Automata

A push-down automaton (PDA) is a tuple $(Q, \Sigma, \Gamma, \delta, p, Z, F)$, where:

- Q is a finite set of states
- Σ is the input alphabet
- Γ is the push-down alphabet
- δ is a set of transitions
- p is the initial state
- Z is a push-down symbol, initially in the stack
- F is the same set of finial states

In general a PDA is non-deterministic

A move in a PDA consists of:

- Reading a symbol of $\Sigma \cup \{\epsilon\}$
- Changing state
- Replacing the top symbol of the stack by a (possibly empty) string

Writing a symbol on the stack ”pushes” all the other
A PDA accepts an input string x if it reaches:

- Either a final state in F
- Or an empty stack (ϵ)

After reading the input x

Theorem 3 *PDAs accept exactly the context free languages*

Don’t worry about proving theorem 3

Theorem 4 *Deterministic PDAs accept strictly fewer languages that nondeterministic ones*