COMP2022: Formal Languages and Logic

2018, Semester 2, Week 5

Joseph Godbehere

30th August, 2018

THE UNIVERSITY OF SYDNEY

Outline

- **Non-Deterministic Finite Automata (NFA)**
  - Non-determinism
  - ε-transitions
- Equivalence of NFA and DFA
- Minimal DFA
- Regular Languages and Closure properties
- Regular Expressions (introduction)

Non deterministic Finite Automata (NFA)

DFA

- has exactly one transition per input from each state
- no choice in the computation $\Longrightarrow$ *deterministic*

NFA

- can have any number of transitions per input from each state
- so some steps of the computation might be *non-deterministic*
- can also have ε-transitions
  - i.e. transitions which the automaton can follow without scanning any input

Non-determinism

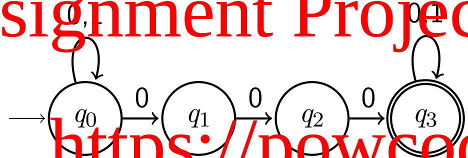
- As we scan through a string, the NFA can be in many states at the same time
- Transitions from a state given an input symbol is to a *set* of states (0, 1, 2 or more states)

- The string is accepted if at least one sequence of states leads to a final state
  - i.e. if we are in at least one accept state, after reading all the input

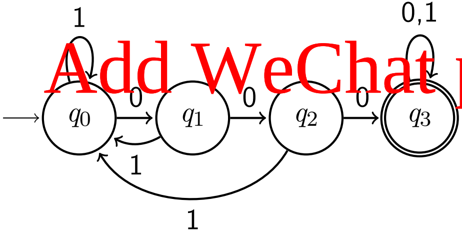
- Parallel computation

Example

NFA accepting strings over $\{0, 1\}$ containing substring "000"



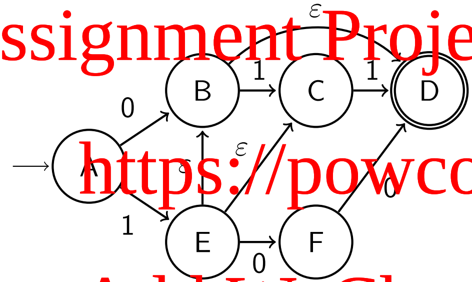
The corresponding DFA would be



NFA with ε-Transitions

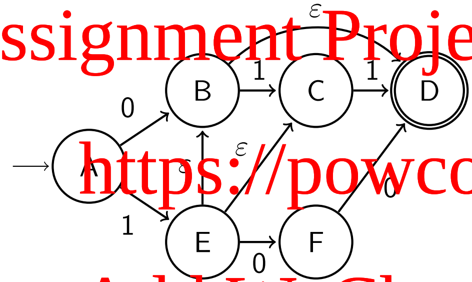
- ε-transitions do not consume any input
- Remember that ε is the empty string

Example



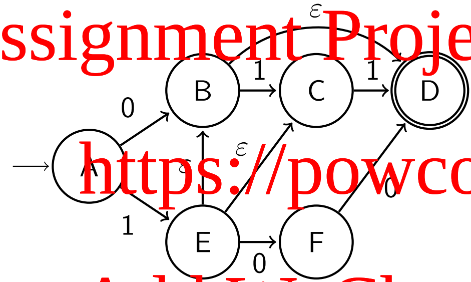
- From A, with a 0:
- From A, with a 1:
- From B, with ε:
- From E, with ε:

Example



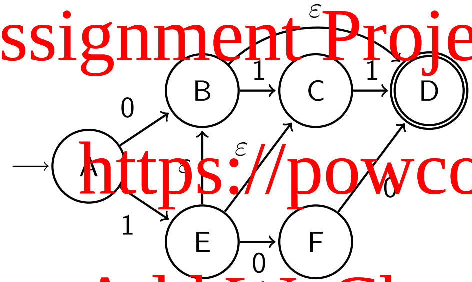
- From A, with a 0: B
- From A, with a 1:
- From B, with ε:
- From E, with ε:

Example



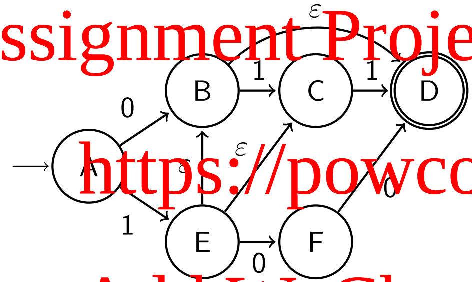
- From A, with a 0: B
- From A, with a 1: E
- From B, with ε:
- From E, with ε:

Example



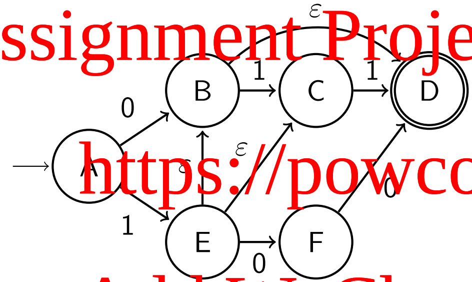
- From A, with a 0: B
- From A, with a 1: E
- From B, with ε: D
- From E, with ε:

Example



- From A, with a 0: B
- From A, with a 1: E
- From B, with ε: D
- From E, with ε: B and C

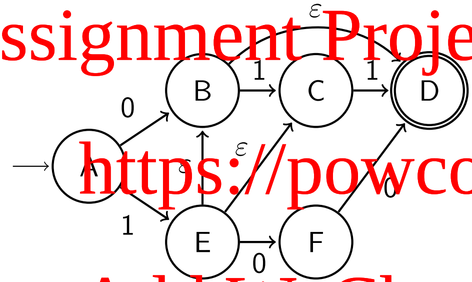
Example



- From A, with a 0: B
- From A, with a 1: E
- From B, with ε: D
- From E, with ε: B and C

So from A with 0, we can potentially *also* reach D without needing to scan more input.
(Epsilon closure (*later in the lecture*))

Example



- From A, with a 0: B
- From A, with a 1: E
- From B, with ε: D
- From E, with ε: B and C

	0	1	ε
A	$\{E\}$	$\{B\}$	$\emptyset$
B	$\emptyset$	$\{C\}$	$\{D\}$
C	$\emptyset$	$\{D\}$	$\emptyset$
D	$\emptyset$	$\emptyset$	$\emptyset$
E	$\{F\}$	$\emptyset$	$\{B, C\}$
F	$\{D\}$	$\emptyset$	$\emptyset$

So from A with 0, we can potentially *also* reach D without needing to scan more input.
(Epsilon closure (*later in the lecture*))

Formal definition of NFA

Definition: A non-deterministic finite automaton is a 5-tuple

- $(Q, \Sigma, \delta, q_0, F)$ where:
- Q is a finite set called the states,
- Σ is a finite set called the alphabet,
- $\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$ is the transition function*,
- q_0 is the start state, and
- $F \subseteq Q$ is the set of accept states.

*Recall:

- if $A = \{a, b, c\}$ then $\mathcal{P}(A) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$
- $ab\varepsilon = a\varepsilon b = \varepsilon ab = ab$
- $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$

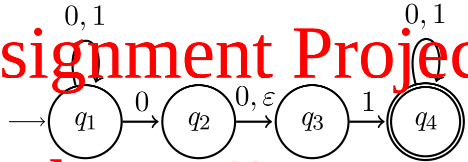
Transition function for NFA

$$\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$$

- $\delta(q, a)$ is a *set* of states which is the union of all the states which can be reached from q on a.
  - This could be the empty set $\emptyset$

- Note: when following epsilon transitions, you can also remain in the same state (*as if* $q \in \delta(q, \varepsilon)$ for all $q \in Q$)

Example



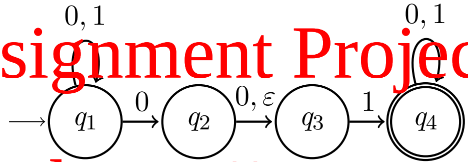
- $Q = \{q_1, q_2, q_3, q_4\}$
- $\Sigma = ?$
- The start state is q_1
- The set of accept states is $\{q_4\}$

$\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$ is given by:

	0	1	ε
q_1	$\{q_1, q_2\}$	$\{q_1\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$

Some strings where N reaches the accept state: $01, 0000001$

EXAMPLE



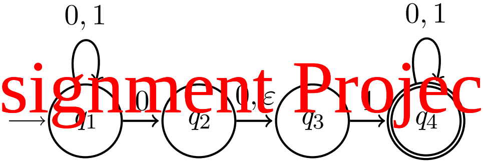
- $Q = \{q_1, q_2, q_3, q_4\}$
- $\Sigma = \{0, 1\}$
- The start state is q_1
- The set of accept states is $\{q_4\}$

$\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$ is given by:

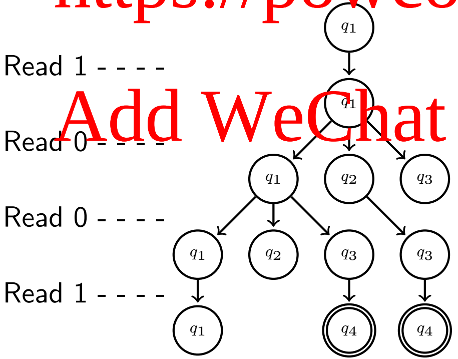
	0	1	ε
q_1	$\{q_1, q_2\}$	$\{q_1\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$

Some strings where N reaches the accept state: $01, 0000001$

NFA computation : Parallelism



Several choices that exist can be seen as parallel computation or as a tree of possibilities

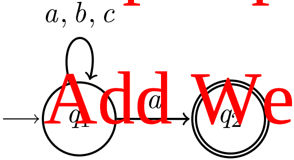


Language of an NFA

- A string w is accepted by an NFA if at least one sequence of transitions starting from q_0 on input w leads to an accept state
  - i.e. if we are in at least one accept state, after reading all the input
- The language recognised by an NFA (i.e. the set of strings it accepts) is called regular

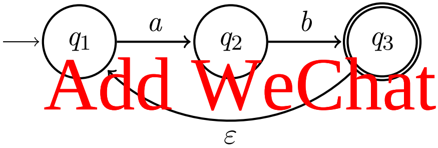
Examples of NFA (1)

L_1: Strings over $\{a, b, c\}$ ending with an a



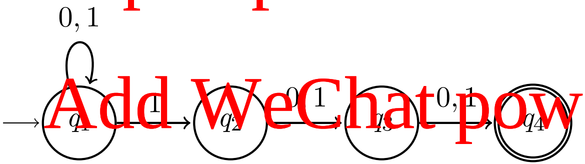
Examples of NFA (2)

L_2: Strings over $\{a, b, c\}$ composed of at least one repetition of the substring ab



Examples of NFA (3)

L_3: Strings over $\{0, 1\}$ with a 1 in the third last position



Outline

- Non-Deterministic Finite Automata (NFA)
  - Non-determinism
  - ε-transitions

- **Equivalence of NFA and DFA**

- Minimal DFA

- Regular Languages and Closure properties

- Regular Expressions (introduction)

Equivalence of NFAs and DFAs

- DFA: exactly one transition per input from each state
- NFA: zero, one or several possible transitions, ε-transitions
- $\Longrightarrow$ any DFA is a NFA (NFA more general)

But are there languages that are recognised by some NFA but not by a DFA?

No. If a language is recognised by a NFA then we can always devise a DFA which recognises it.

Theorem: Every NFA has an equivalent DFA

Regular languages and finite automata

Theorem (Kleene 56):
The class of regular languages is exactly the same as the class of languages accepted by DFAs

Theorem (Rabin & Scott 59):
The class of regular languages is exactly the same as the class of languages accepted by NFAs

$\Longrightarrow$ Any regular language must be accepted by at least one DFA and at least one NFA

Can we transform NFAs into DFAs?

Transforming a NFA into a DFA: intro

Key idea: Read the NFA as if it was a DFA and keep in memory the possible set of states it could be in for each given input (i.e. similar to the levels of the decision tree example.)

Each state of the new DFA is a subset of the NFA states (subset construction)

When we transition from a state X, with a given input symbol i, we must also consider all the ways in which we could follow epsilon transitions before and after reading i

- i.e. We transition to the set of states reachable via paths using any number of ε transitions before and/or after i

Epsilon-Closure

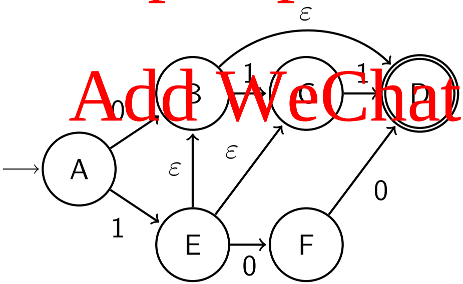
The *Epsilon-closure* of a state q, denoted $E(q)$, is the set of all states which can be reached from q by 0 or more ε-transitions

- $q \in E(q)$
- If $p \in E(q)$ and there is an ε-transition from p to r, then $r \in E(q)$

Epsilon-Closure

The *Epsilon-closure* of a state q, denoted $E(q)$, is the set of all states which can be reached from q by 0 or more ε-transitions

- $q \in E(q)$
- If $p \in E(q)$ and there is an ε-transition from p to r, then $r \in E(q)$



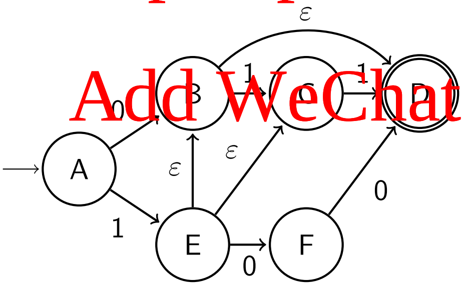
Example:

- $E(A) =$
- $E(B) =$
- $E(E) =$

Epsilon-Closure

The *Epsilon-closure* of a state q, denoted $E(q)$, is the set of all states which can be reached from q by 0 or more ε-transitions

- $q \in E(q)$
- If $p \in E(q)$ and there is an ε-transition from p to r, then $r \in E(q)$



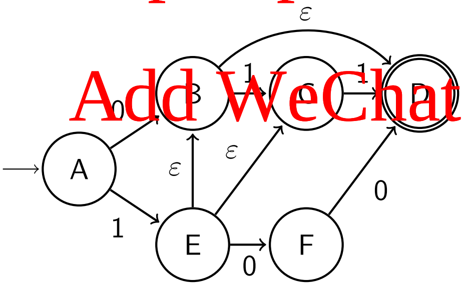
Example:

- $E(A) = \{A\}$
- $E(B) =$
- $E(E) =$

Epsilon-Closure

The *Epsilon-closure* of a state q, denoted $E(q)$, is the set of all states which can be reached from q by 0 or more ε-transitions

- $q \in E(q)$
- If $p \in E(q)$ and there is an ε-transition from p to r, then $r \in E(q)$



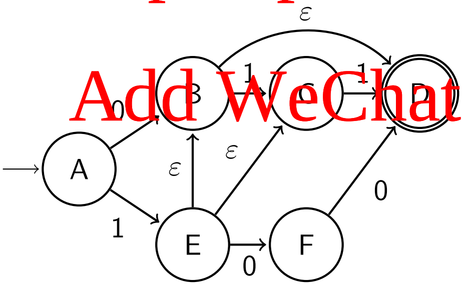
Example:

- $E(A) = \{A\}$
- $E(B) = \{B, D\}$
- $E(E) =$

Epsilon-Closure

The *Epsilon-closure* of a state q, denoted $E(q)$, is the set of all states which can be reached from q by 0 or more ε-transitions

- $q \in E(q)$
- If $p \in E(q)$ and there is an ε-transition from p to r, then $r \in E(q)$



Example:

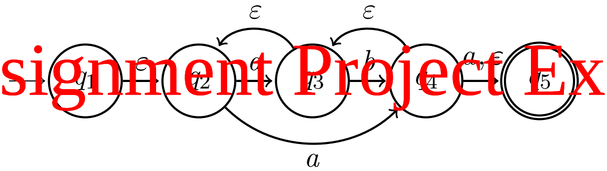
- $E(A) = \{A\}$
- $E(B) = \{B, D\}$
- $E(E) = \{B, C, D, E\}$

Epsilon-Closure for sets

The Epsilon-closure of a set of states S, denoted $E(S)$, is the union of the Epsilon-closure of each state.

$$E(S \cup T) = E(S) \cup E(T)$$

Constructing epsilon-closures



	a	b	ε
q_1	$\emptyset$	$\emptyset$	$\{q_2\}$
q_2	$\{q_3, q_4\}$	$\emptyset$	$\emptyset$
q_3	$\emptyset$	$\{q_4\}$	$\{q_2\}$
q_4	$\{q_5\}$	$\emptyset$	$\{q_3, q_5\}$
q_5	$\emptyset$	$\emptyset$	$\emptyset$

$E(\{q_1\}) =$

$E(\{q_2\}) =$

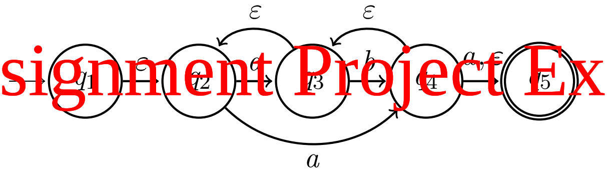
$E(\{q_3\}) =$

$E(\{q_4\}) =$

$E(\{q_5\}) =$

$E(\{q_1, q_3, q_5\}) =$

Constructing epsilon-closures



	a	b	ε
q_1	$\emptyset$	$\emptyset$	$\{q_2\}$
q_2	$\{q_3, q_4\}$	$\emptyset$	$\emptyset$
q_3	$\emptyset$	$\{q_4\}$	$\{q_2\}$
q_4	$\{q_5\}$	$\emptyset$	$\{q_3, q_5\}$
q_5	$\emptyset$	$\emptyset$	$\emptyset$

$E(\{q_1\}) = \{q_1, q_2\}$

$E(\{q_2\}) =$

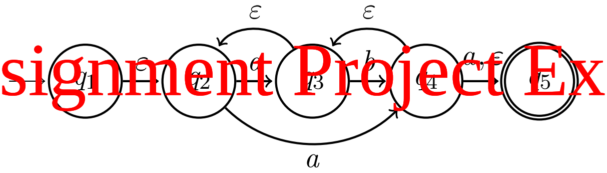
$E(\{q_3\}) =$

$E(\{q_4\}) =$

$E(\{q_5\}) =$

$E(\{q_1, q_3, q_5\}) =$

Constructing epsilon-closures

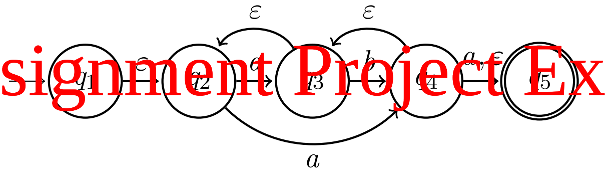


	a	b	ε
q_1	$\emptyset$	$\emptyset$	$\{q_2\}$
q_2	$\{q_3, q_4\}$	$\emptyset$	$\emptyset$
q_3	$\emptyset$	$\{q_4\}$	$\{q_2\}$
q_4	$\{q_5\}$	$\emptyset$	$\{q_3, q_5\}$
q_5	$\emptyset$	$\emptyset$	$\emptyset$

$$E(\{q_1\}) = \{q_1, q_2\}$$
$$E(\{q_2\}) = \{q_2\}$$
$$E(\{q_3\}) =$$
$$E(\{q_4\}) =$$
$$E(\{q_5\}) =$$

$$E(\{q_1, q_3, q_5\}) =$$

Constructing epsilon-closures

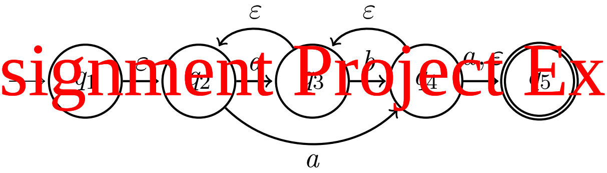


	a	b	ε
q_1	$\emptyset$	$\emptyset$	$\{q_2\}$
q_2	$\{q_3, q_4\}$	$\emptyset$	$\emptyset$
q_3	$\emptyset$	$\{q_4\}$	$\{q_2\}$
q_4	$\{q_5\}$	$\emptyset$	$\{q_3, q_5\}$
q_5	$\emptyset$	$\emptyset$	$\emptyset$

$E(\{q_1\}) = \{q_1, q_2\}$
$E(\{q_2\}) = \{q_2\}$
$E(\{q_3\}) = \{q_2, q_3\}$
$E(\{q_4\}) =$
$E(\{q_5\}) =$

$E(\{q_1, q_3, q_5\}) =$

Constructing epsilon-closures

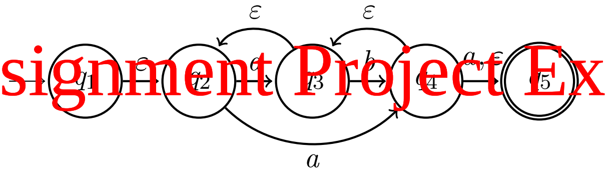


	a	b	ε
q_1	$\emptyset$	$\emptyset$	$\{q_2\}$
q_2	$\{q_3, q_4\}$	$\emptyset$	$\emptyset$
q_3	$\emptyset$	$\{q_4\}$	$\{q_2\}$
q_4	$\{q_5\}$	$\emptyset$	$\{q_3, q_5\}$
q_5	$\emptyset$	$\emptyset$	$\emptyset$

$$E(\{q_1\}) = \{q_1, q_2\}$$
$$E(\{q_2\}) = \{q_2\}$$
$$E(\{q_3\}) = \{q_2, q_3\}$$
$$E(\{q_4\}) = \{q_2, q_3, q_4, q_5\}$$
$$E(\{q_5\}) =$$

$$E(\{q_1, q_3, q_5\}) =$$

Constructing epsilon-closures

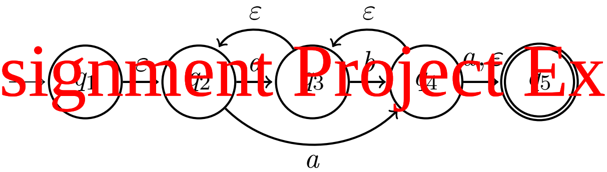


	a	b	ε
q_1	$\emptyset$	$\emptyset$	$\{q_2\}$
q_2	$\{q_3, q_4\}$	$\emptyset$	$\emptyset$
q_3	$\emptyset$	$\{q_4\}$	$\{q_2\}$
q_4	$\{q_5\}$	$\emptyset$	$\{q_3, q_5\}$
q_5	$\emptyset$	$\emptyset$	$\emptyset$

$$E(\{q_1\}) = \{q_1, q_2\}$$
$$E(\{q_2\}) = \{q_2\}$$
$$E(\{q_3\}) = \{q_2, q_3\}$$
$$E(\{q_4\}) = \{q_2, q_3, q_4, q_5\}$$
$$E(\{q_5\}) = \{q_5\}$$

$$E(\{q_1, q_3, q_5\}) =$$

Constructing epsilon-closures

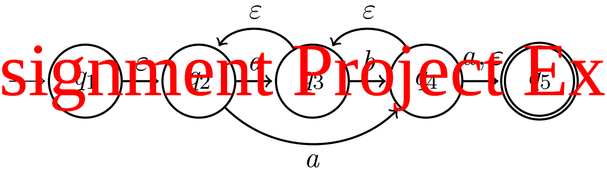


	a	b	ε
q_1	$\emptyset$	$\emptyset$	$\{q_2\}$
q_2	$\{q_3, q_4\}$	$\emptyset$	$\emptyset$
q_3	$\emptyset$	$\{q_4\}$	$\{q_2\}$
q_4	$\{q_5\}$	$\emptyset$	$\{q_3, q_5\}$
q_5	$\emptyset$	$\emptyset$	$\emptyset$

$$E(\{q_1\}) = \{q_1, q_2\}$$
$$E(\{q_2\}) = \{q_2\}$$
$$E(\{q_3\}) = \{q_2, q_3\}$$
$$E(\{q_4\}) = \{q_2, q_3, q_4, q_5\}$$
$$E(\{q_5\}) = \{q_5\}$$

$$E(\{q_1, q_3, q_5\})$$
$$= E(\{q_1\}) \cup E(\{q_3\}) \cup E(\{q_5\})$$
$$=$$

Constructing epsilon-closures



	a	b	ε
q_1	$\emptyset$	$\emptyset$	$\{q_2\}$
q_2	$\{q_3, q_4\}$	$\emptyset$	$\emptyset$
q_3	$\emptyset$	$\{q_4\}$	$\{q_2\}$
q_4	$\{q_5\}$	$\emptyset$	$\{q_3, q_5\}$
q_5	$\emptyset$	$\emptyset$	$\emptyset$

$$E(\{q_1\}) = \{q_1, q_2\}$$
$$E(\{q_2\}) = \{q_2\}$$
$$E(\{q_3\}) = \{q_2, q_3\}$$
$$E(\{q_4\}) = \{q_2, q_3, q_4, q_5\}$$
$$E(\{q_5\}) = \{q_5\}$$

$$E(\{q_1, q_3, q_5\})$$
$$= E(\{q_1\}) \cup E(\{q_3\}) \cup E(\{q_5\})$$
$$= \{q_1, q_2, q_3, q_5\}$$

NFA to DFA algorithm

We want to convert a NFA $N = (Q, \Sigma, \delta, q_0, F)$ into an equivalent DFA $M = (Q', \Sigma, \delta', q_0', F')$ which recognises $L(N)$

- $Q' \subseteq \mathcal{P}(Q)$
- Σ does not change
- $\delta'(R, a) = \cup_{r \in R} E(\delta(r, a))$
- $q_0' = E(q_0)$
- $F' = \{R \in Q' | R$ contains an accept state of $N\}$

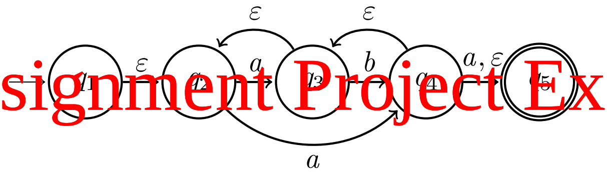
NFA to DFA algorithm

We want to convert a NFA $N = (Q, \Sigma, \delta, q_0, F)$ into an equivalent DFA $M = (Q', \Sigma, \delta', q_0', F')$ which recognises $L(N)$

Algorithm

1. DFA start state is $E(q)$, where q is the NFA start state
2. If $R \subseteq \mathcal{P}(Q)$ is a DFA state and $a \in \Sigma$ then construct the following DFA state as a DFA table entry in either 2 ways:
    - $\delta'(R, a) = E(\cup_{r \in R} \delta(r, a))$ closure of union
    - $\delta'(R, a) = \cup_{r \in R} E(\delta(r, a))$ union of closures
3. A DFA state is accepting if any of its elements are an NFA accept state.

From NFA to DFA: example

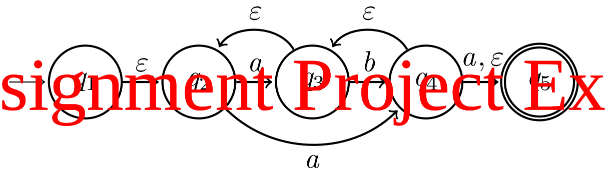


Transition table:

	a	b

Resulting DFA:

From NFA to DFA: example

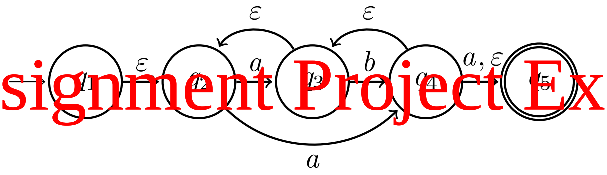


Transition table:

	a	b
$E(start) =$		

Resulting DFA:

From NFA to DFA: example

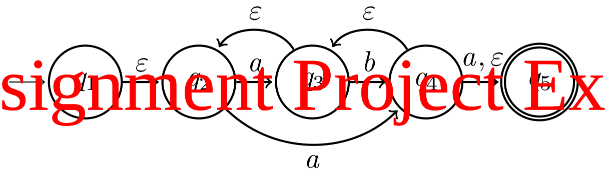


Transition table:

	a	b
$E(start) = \{q_1, q_2\}$		

Resulting DFA:

From NFA to DFA: example

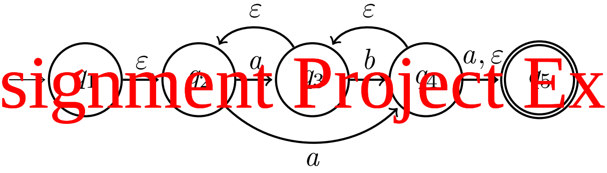


Transition table:

	a	b
$E(start) = \{q_1, q_2\}$	$\{q_2, q_3, q_4, q_5\}$	

Resulting DFA:

From NFA to DFA: example

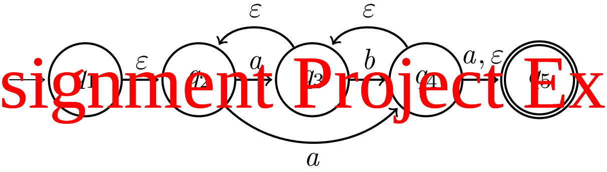


Transition table:

	a	b
$E(start) = \{q_1, q_2\}$	$\{q_2, q_3, q_4, q_5\}$	
$\{q_2, q_3, q_4, q_5\}$		

Resulting DFA:

From NFA to DFA: example

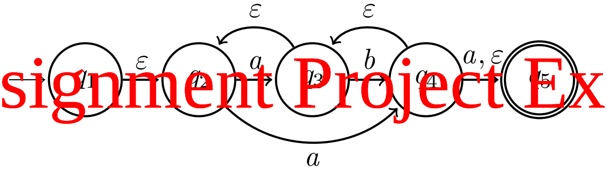


Transition table:

	a	b
$E(start) = \{q_1, q_2\}$	$\{q_2, q_3, q_4, q_5\}$	$\emptyset$
$\{q_2, q_3, q_4, q_5\}$		

Resulting DFA:

FROM NFA TO DFA: EXAMPLE

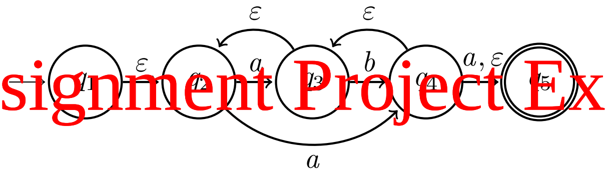


Transition table:

	a	b
$E(start) = \{q_1, q_2\}$	$\{q_2, q_3, q_4, q_5\}$	$\emptyset$
$\{q_2, q_3, q_4, q_5\}$		
$\emptyset$		

Resulting DFA:

From NFA to DFA: example

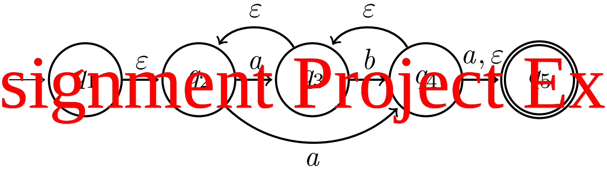


Transition table:

	a	b
$E(start) = \{q_1, q_2\}$	$\{q_2, q_3, q_4, q_5\}$	$\emptyset$
$\{q_2, q_3, q_4, q_5\}$	$\{q_2, q_3, q_4, q_5\}$	
$\emptyset$		

Resulting DFA:

From NFA to DFA: example

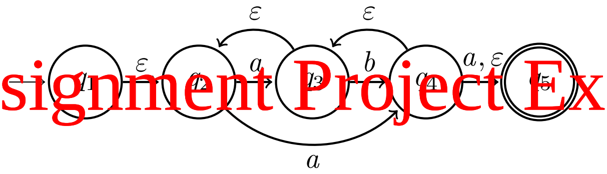


Transition table:

	a	b
$E(start) = \{q_1, q_2\}$	$\{q_2, q_3, q_4, q_5\}$	$\emptyset$
$\{q_2, q_3, q_4, q_5\}$	$\{q_2, q_3, q_4, q_5\}$	$\{q_2, q_3, q_4, q_5\}$
$\emptyset$		

Resulting DFA:

From NFA to DFA: example

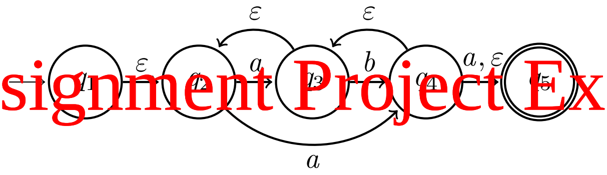


Transition table:

	a	b
$E(start) = \{q_1, q_2\}$	$\{q_2, q_3, q_4, q_5\}$	$\emptyset$
$\{q_2, q_3, q_4, q_5\}$	$\{q_2, q_3, q_4, q_5\}$	$\{q_2, q_3, q_4, q_5\}$
$\emptyset$	$\emptyset$	$\emptyset$

Resulting DFA:

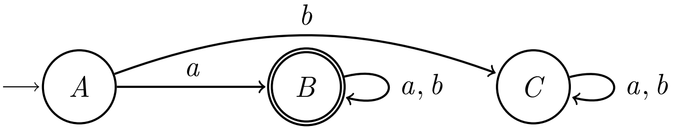
From NFA to DFA: example



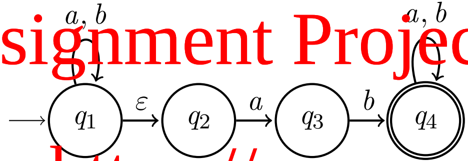
Transition table:

		a	b
A	$E(start) = \{q_1, q_2\}$	$\{q_2, q_3, q_4, q_5\}$	$\emptyset$
B	$\{q_2, q_3, q_4, q_5\}$	$\{q_2, q_3, q_4, q_5\}$	$\{q_2, q_3, q_4, q_5\}$
C	$\emptyset$	$\emptyset$	$\emptyset$

Resulting DFA:



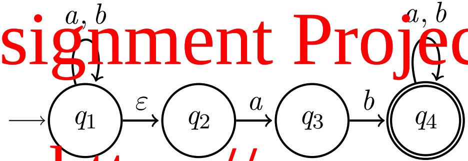
From NFA to DFA: example



Transition table:

	a	b

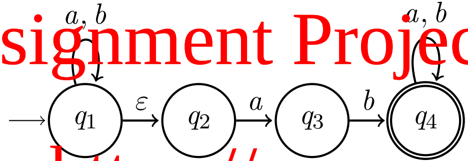
From NFA to DFA: example



Transition table:

	a	b
$E(start) =$		

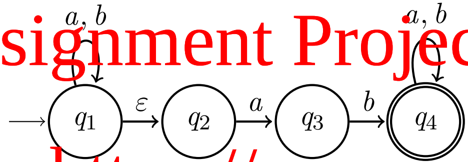
From NFA to DFA: example



Transition table:

	a	b
$E(start) = \{q_1, q_2\}$		

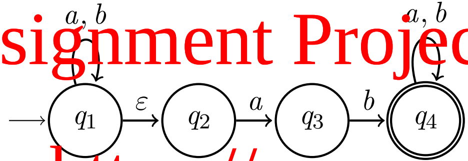
From NFA to DFA: example



Transition table:

	a	b
$E(start) = \{q_1, q_2\}$	$\{q_1, q_2, q_3\}$	

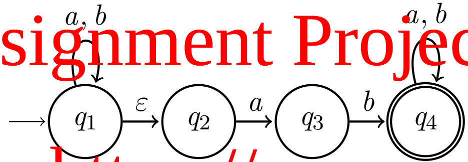
From NFA to DFA: example



Transition table:

	a	b
$E(start) = \{q_1, q_2\}$	$\{q_1, q_2, q_3\}$	
$\{q_1, q_2, q_3\}$		

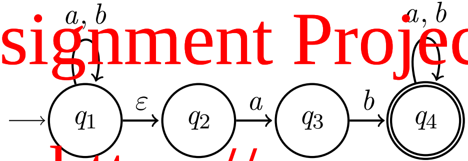
From NFA to DFA: example



Transition table:

	a	b
$E(start) = \{q_1, q_2\}$	$\{q_1, q_2, q_3\}$	$\{q_1, q_2\}$
$\{q_1, q_2, q_3\}$		

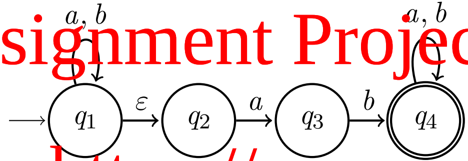
From NFA to DFA: example



Transition table:

	a	b
$E(start) = \{q_1, q_2\}$	$\{q_1, q_2, q_3\}$	$\{q_1, q_2\}$
$\{q_1, q_2, q_3\}$	$\{q_1, q_2, q_3\}$	

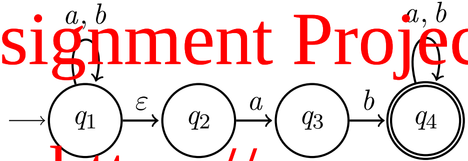
From NFA to DFA: example



Transition table:

	a	b
$E(start) = \{q_1, q_2\}$	$\{q_1, q_2, q_3\}$	$\{q_1, q_2\}$
$\{q_1, q_2, q_3\}$	$\{q_1, q_2, q_3\}$	$\{q_1, q_2, q_4\}$

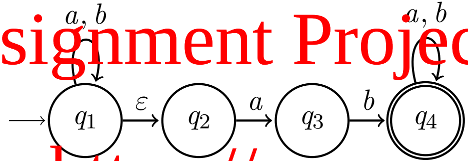
From NFA to DFA: example



Transition table:

	a	b
$E(start) = \{q_1, q_2\}$	$\{q_1, q_2, q_3\}$	$\{q_1, q_2\}$
$\{q_1, q_2, q_3\}$	$\{q_1, q_2, q_3\}$	$\{q_1, q_2, q_4\}$
$\underline{\{q_1, q_2, q_4\}}$		

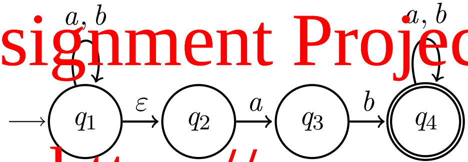
From NFA to DFA: example



Transition table:

	a	b
$E(start) = \{q_1, q_2\}$	$\{q_1, q_2, q_3\}$	$\{q_1, q_2\}$
$\{q_1, q_2, q_3\}$	$\{q_1, q_2, q_3\}$	$\{q_1, q_2, q_4\}$
$\{q_1, q_2, q_4\}$	$\{q_1, q_2, q_3, q_4\}$	

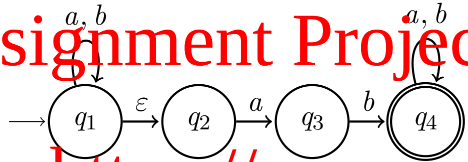
FROM NFA TO DFA: EXAMPLE



Transition table:

	a	b
$E(start) = \{q_1, q_2\}$	$\{q_1, q_2, q_3\}$	$\{q_1, q_2\}$
$\{q_1, q_2, q_3\}$	$\{q_1, q_2, q_3\}$	$\{q_1, q_2, q_4\}$
$\underline{\{q_1, q_2, q_4\}}$	$\{q_1, q_2, q_3, q_4\}$	
$\underline{\{q_1, q_2, q_3, q_4\}}$		

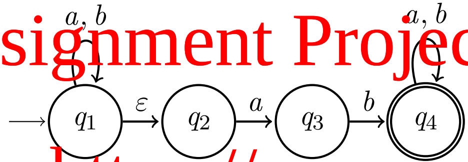
From NFA to DFA: example



Transition table:

	a	b
$E(start) = \{q_1, q_2\}$	$\{q_1, q_2, q_3\}$	$\{q_1, q_2\}$
$\{q_1, q_2, q_3\}$	$\{q_1, q_2, q_3\}$	$\{q_1, q_2, q_4\}$
$\underline{\{q_1, q_2, q_4\}}$	$\{q_1, q_2, q_3, q_4\}$	$\{q_1, q_2, q_4\}$
$\underline{\{q_1, q_2, q_3, q_4\}}$		

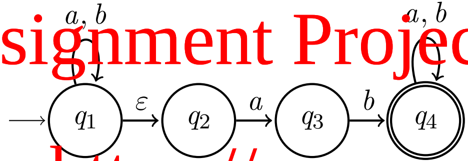
From NFA to DFA: example



Transition table:

	a	b
$E(start) = \{q_1, q_2\}$	$\{q_1, q_2, q_3\}$	$\{q_1, q_2\}$
$\{q_1, q_2, q_3\}$	$\{q_1, q_2, q_3\}$	$\{q_1, q_2, q_4\}$
$\underline{\{q_1, q_2, q_4\}}$	$\{q_1, q_2, q_3, q_4\}$	$\{q_1, q_2, q_4\}$
$\underline{\{q_1, q_2, q_3, q_4\}}$	$\{q_1, q_2, q_3, q_4\}$	

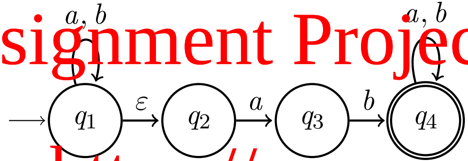
From NFA to DFA: example



Transition table:

	a	b
$E(start) = \{q_1, q_2\}$	$\{q_1, q_2, q_3\}$	$\{q_1, q_2\}$
$\{q_1, q_2, q_3\}$	$\{q_1, q_2, q_3\}$	$\{q_1, q_2, q_4\}$
$\underline{\{q_1, q_2, q_4\}}$	$\{q_1, q_2, q_3, q_4\}$	$\{q_1, q_2, q_4\}$
$\underline{\{q_1, q_2, q_3, q_4\}}$	$\{q_1, q_2, q_3, q_4\}$	$\{q_1, q_2, q_4\}$

From NFA to DFA: example



Transition table:

		a	b
A	$E(start) = \{q_1, q_2\}$	$\{q_1, q_2, q_3\}$	$\{q_1, q_2\}$
B	$\{q_1, q_2, q_3\}$	$\{q_1, q_2, q_3\}$	$\{q_1, q_2, q_4\}$
C	$\underline{\{q_1, q_2, q_4\}}$	$\{q_1, q_2, q_3, q_4\}$	$\{q_1, q_2, q_4\}$
D	$\underline{\{q_1, q_2, q_3, q_4\}}$	$\{q_1, q_2, q_3, q_4\}$	$\{q_1, q_2, q_4\}$

Outline

- Non-Deterministic Finite Automata (NFA)
  - Non-determinism
  - ε-transitions
- Equivalence of NFA and DFA
- **Minimal DFA**
- Regular Languages and Closure properties
- Regular Expressions (introduction)

Minimisation of DFA (idea)

Every DFA has a unique equivalent *minimal DFA*

Definition: Two states s and t are *equivalent* if: for any string, the path from s leads to an accepting state if and only if the path from t does.

To reduce an automaton, find all non-equivalent states:

1. If s is accepting and t non accepting, then they are not equivalent

2. If, with input x, there is a transition from s to s' and from t to to t' and we know that s' and t' are not equivalent, then s and t are not equivalent

Then merge equivalent states.

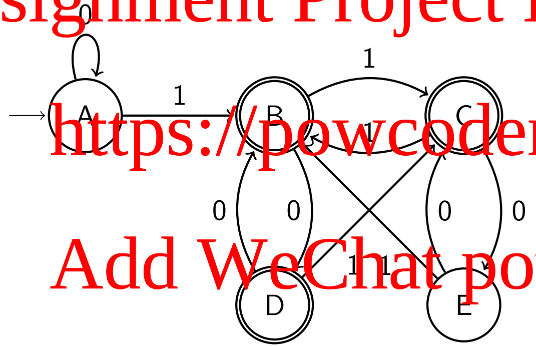
Minimisation of DFA: Table-filling Algorithm

1. Examine all pairs of states and find all pairs s,t that are NOT equivalent, ie satisfying either of:
    1.1 s is accepting and t is not or vice versa
    1.2 with the same input x, s goes to a state s' and t to a state t' and you have already proven that s' and t' are NOT equivalent
2. When no further non-equivalence can be proven, then the remaining pairs can be merged respectively into one state.

Theorems:

1. If two states are not distinguished by the table-filling algorithm, then the states are equivalent
2. The equivalence of states is transitive

Example



The table of state non-equivalences

A	-	-	-	-	-
B		-	-	-	-
C			-	-	-
D				-	-
E					-
	A	B	C	D	E

1. Cross out all the pairs of accept/non-accept states. A, E are non-accepting and all the others are.

THE TABLE OF STATE NON-EQUIVALENCES

A	-	-	-	-	-
B	x	-	-	-	-
C	x		-	-	-
D	x			-	-
E		x	x	x	-
	A	B	C	D	E

1. Cross out all the pairs of accept/non-accept states. A, E are non-accepting and all the others are.
2. Look at each remaining pair.

The table of state non-equivalences

A	-	-	-	-	-
B	x	-	-	-	-
C	x		-	-	-
D	x			-	-
E	x	x	x	x	-
	A	B	C	D	E

1. Cross out all the pairs of accept/non-accept states. A, E are non-accepting and all the others are.
2. Look at each remaining pair.
   - $\delta(A, 0) = A$ and $\delta(E, 0) = C$, but $A \not\equiv C$, therefore $A \not\equiv E$

THE TABLE OF STATE NON-EQUIVALENCES

A	-	-	-	-	-
B	x	-	-	-	-
C	x	x	-	-	-
D	x			-	-
E	x	x	x	x	-
	A	B	C	D	E

1. Cross out all the pairs of accept/non-accept states. A, E are non-accepting and all the others are.
2. Look at each remaining pair.
   - $\delta(A, 0) = A$ and $\delta(E, 0) = C$, but $A \not\equiv C$, therefore $A \not\equiv E$
   - $\delta(B, 0) = D$ and $\delta(C, 0) = E$, but $D \not\equiv E$, therefore $B \not\equiv C$

The table of state non-equivalences

A	-	-	-	-	-
B	x	-	-	-	-
C	x	x	-	-	-
D	x			-	-
E	x	x	x	x	-
	A	B	C	D	E

1. Cross out all the pairs of accept/non-accept states. A, E are non-accepting and all the others are.
2. Look at each remaining pair.
   - $\delta(A, 0) = A$ and $\delta(E, 0) = C$, but $A \not\equiv C$, therefore $A \not\equiv E$
   - $\delta(B, 0) = D$ and $\delta(C, 0) = E$, but $D \not\equiv E$, therefore $B \not\equiv C$
   - We can't find a reason to claim $B \not\equiv D$ yet

The table of state non-equivalences

A	-	-	-	-	-
B	x	-	-	-	-
C	x	x	-	-	-
D	x		x	-	-
E	x	x	x	x	-
	A	B	C	D	E

1. Cross out all the pairs of accept/non-accept states. A, E are non-accepting and all the others are.
2. Look at each remaining pair.
   - $\delta(A, 0) = A$ and $\delta(E, 0) = C$, but $A \not\equiv C$, therefore $A \not\equiv E$
   - $\delta(B, 0) = D$ and $\delta(C, 0) = E$, but $D \not\equiv E$, therefore $B \not\equiv C$
   - We can't find a reason to claim $B \not\equiv D$ yet
   - $\delta(C, 0) = E$ and $\delta(D, 0) = B$, and $E \not\equiv B$, therefore $C \not\equiv D$

The table of state non-equivalences

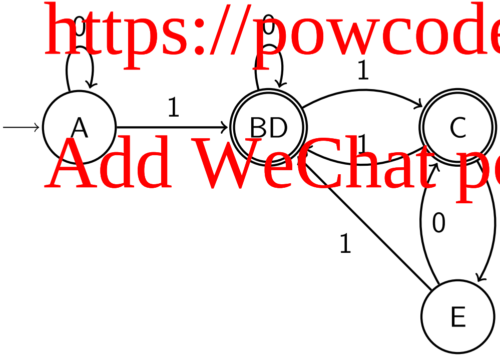
A	-	-	-	-	-
B	x	-	-	-	-
C	x	x	-	-	-
D	x		x	-	-
E	x	x	x	x	-
	A	B	C	D	E

1. Cross out all the pairs of accept/non-accept states. A, E are non-accepting and all the others are.
2. Look at each remaining pair.
   - $\delta(A,0) = A$ and $\delta(E,0) = C$, but $A \not\equiv C$, therefore $A \not\equiv E$
   - $\delta(B,0) = D$ and $\delta(C,0) = E$, but $D \not\equiv E$, therefore $B \not\equiv C$
   - We can't find a reason to claim $B \not\equiv D$ yet
   - $\delta(C,0) = E$ and $\delta(D,0) = B$, and $E \not\equiv B$, therefore $C \not\equiv D$
3. Repeat step 2 until no changes are found

Merging equivalent states

Theorem: If two states are not distinguished by the table-filling algorithm, then that pair of states are equivalent.

We can merge together equivalent states. The minimal DFA for the example becomes:



Resulting DFA

- is minimal
- is unique

Outline

- Non-Deterministic Finite Automata (NFA)
    - Non-determinism
    - ε-transitions

- Equivalence of NFA and DFA

- Minimal DFA

- **Regular Languages and Closure properties**

- Regular Expressions (introduction)

Revision

Definition:

A language is *regular* if and only if there exists a finite automaton which recognises it.

Minimal examples:

- $\emptyset$ is a regular language
- $\{\varepsilon\}$ is a regular language
- For all $a \in \Sigma$, the set $\{a\}$ is a regular language

Think about how to make a DFA for each of these languages

Closure properties of Regular languages

Let L_1 and L_2 be regular languages.

The *regular operations* are defined as follows:

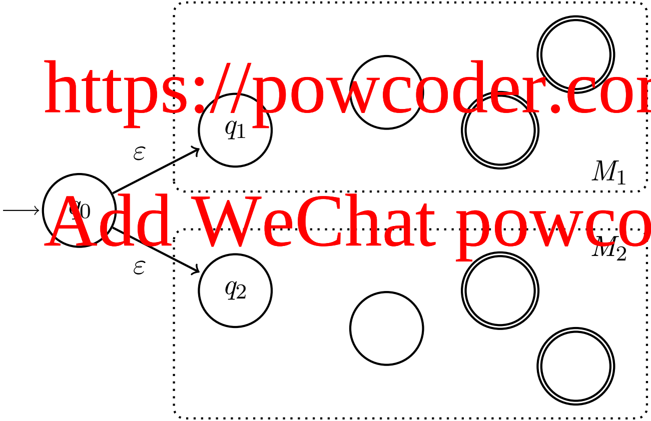
- Union: $L_1 \cup L_2 = \{w | w \in L_1$ or $w \in L_2\}$
- Concatenation: $L_1 L_2 = \{w_1 w_2 | w_1 \in L_1$ and $w_2 \in L_2\}$
- Star: $L^* = \{w_1 w_2 ... w_k | k \geq 0$ and $w_i \in L$ for $i = 1, ..., k\}$

Theorems:

- The union of two regular languages is regular
- The concatenation of two regular languages is regular
- The star closure of a regular language is regular

Union of two regular languages

If L_1 and L_2 are regular, then finite automata M_1 and M_2 exist which recognise them. we can construct an automaton M which recognises $L_1 \cup L_2 = \{w | w \in L_1 \text{ or } w \in L_2\}$



Union of two regular languages

Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$

Let $M = (Q, \Sigma, \delta, q_0, F)$ where:

- $Q = \{q_0\} \cup Q_1 \cup Q_2$
- q_0 is the new start state
- Transition function δ defined for any $q \in Q$ and any $a \in \Sigma_\varepsilon$

$$\delta(q, a) = \begin{cases} \delta_1(q, a) & \text{if } q \in Q_1 \\ \delta_2(q, a) & \text{if } q \in Q_2 \\ \{q_1, q_2\} & \text{if } q = q_0 \text{ and } a = \varepsilon \\ \emptyset & \text{if } q = q_0 \text{ and } a \neq \varepsilon \end{cases}$$

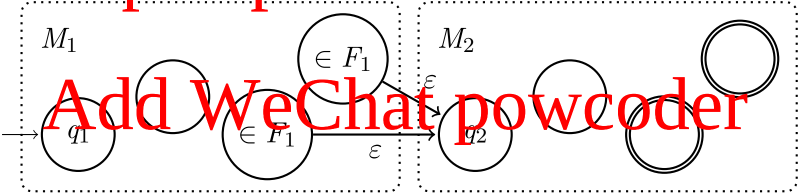
- Accepting set $F = F_1 \cup F_2$

M accepts if and only if M_1 or M_2 does
i.e. $L(M) = \{w | w \in L_1 \text{ or } w \in L_2\} = L_1 \cup L_2$

Concatenation of two regular languages

If L_1 and L_2 are regular, then finite automata M_1 and M_2 exist which recognise them.

we can construct an automaton M which recognises $L_1 L_2 = \{w_1 w_2 | w_1 \in L_1$ and $w_2 \in L_2\}$:



Note that the accept states of M_1 are *not* accept states in M

Concatenation of two regular languages

Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$

Let $M = (Q, \Sigma, \delta, q_0, F)$ where:

- $Q = Q_1 \cup Q_2$
- q_1 is start state (same as M_1)
- Transition function δ defined for any $q \in Q$ and any $a \in \Sigma_\varepsilon$

$$\delta(q, a) = \begin{cases} \delta_1(q, a) & \text{if } q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q, a) & \text{if } q \in F_1 \text{ and } a \neq \varepsilon \\ \delta_1(q, a) \cup \{q_2\} & \text{if } q \in F_1 \text{ and } a = \varepsilon \\ \delta_2(q, a) & \text{if } q \in Q_2 \end{cases}$$

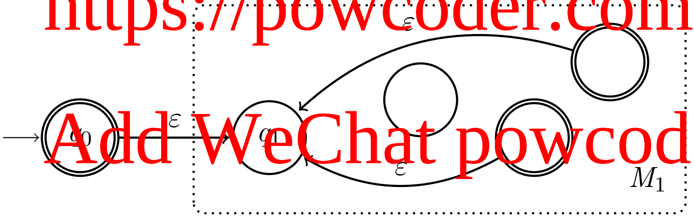
- Accepting set $F = F_2$ (same as M_2)

$L(M) = \{w_1 w_2 | w_1 \in L_1$ and $w_2 \in L_2\} = L_1 L_2$

Star closure of a regular language

If L is regular, then a finite automaton M_1 exists which recognises it.

we can construct an automaton M which recognises $L^* = \{w_1 w_2 ... w_k | k \geq 0 \text{ and } w_i \in L \text{ for } i = 1, ..., k\}$:



e.g. if $L = \{a, b\}$ then this automaton recognises $\{\varepsilon, a, b, aa, ab, bb, aaa, ...\} = L^*$

Star closure of a regular language

Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$

Let $M = (Q, \Sigma, \delta, q_0, F)$ where:

- $Q = Q_1 \cup \{q_0\}$
- q_0 is the new start state
- Transition function δ defined for any $q \in Q$ and any $a \in \Sigma_\varepsilon$

$$\delta(q, a) = \begin{cases} \delta_1(q, a) & \text{if } q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q, a) & \text{if } q \in F_1 \text{ and } a \neq \varepsilon \\ \delta_1(q, a) \cup \{q_1\} & \text{if } q \in F_1 \text{ and } a = \varepsilon \\ \{q_1\} & \text{if } q = q_0 \text{ and } a = \varepsilon \\ \emptyset & \text{if } q = q_0 \text{ and } a \neq \varepsilon \end{cases}$$

- Accepting set $F = F_1 \cup \{q_0\}$

$L(M) = \{w_1 w_2 \ldots w_k | k \geq 0 \text{ and } w_i \in L \text{ for } i = 1, \ldots, k\} = L^*$

Example

Construct an NFA recognising the set of strings containing the pattern $aaab$ or the strings composed of 0 or more sequences of ab

Example from construction to minimal DFA

Let $\Sigma = \{0, 1\}$

Let L_1 be the language of strings which do not contain consecutive 0's

Let L_2 be the language of strings ending with 0

Complement of a regular language

The complement of a regular language is regular

The *complement* of a language L is $\{x|x \notin L\}$

Example

Let $L_1 = \{x|x$ contains an odd number of a's$\}$

Then the complement of L_1 is $L_2 = \{x|x \notin L_1\} = \{x|x$ contains an even number of a's$\}$

You will explore how to construct a suitable automata in this week's tutorial

Intersection of a regular language

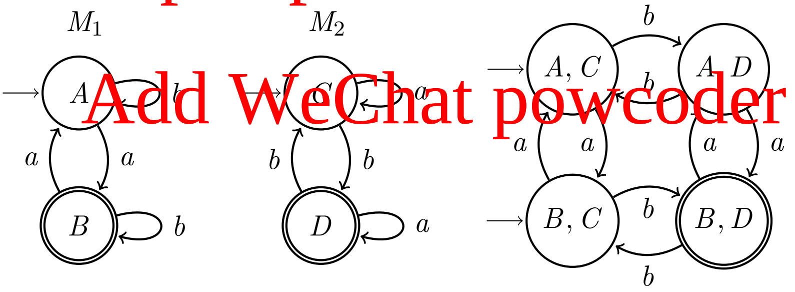
If L_1 and L_2 are regular, then $L_1 \cap L_2$ is regular.

We can use the cross product technique to build the DFA

Example:

$L_1 = \{x | x \in \{a, b\}^*$ and x contains an odd number of a's$\}$

$L_2 = \{x | x \in \{a, b\}^*$ and x contains an odd number of b's$\}$



Cross-product of DFAs

Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ be two *DFA*

Let $M = (Q, \Sigma, \delta_1, q_0, F)$ where:

- $Q = Q_1 \times Q_2$ (i.e. all ordered pairs (u, v) such that $u \in Q_1$ and $v \in Q_2$)

- The start state (q_1, q_2) is the pair of start states from Q_1 and Q_2
- The transition function δ is defined as:

$$\delta((u, v), x) = (\delta_1(u, x), \delta_2(v, x))$$

- Accepting set $F = \{(u, v) | u \in F_1 \text{ and } v \in F_2\}$

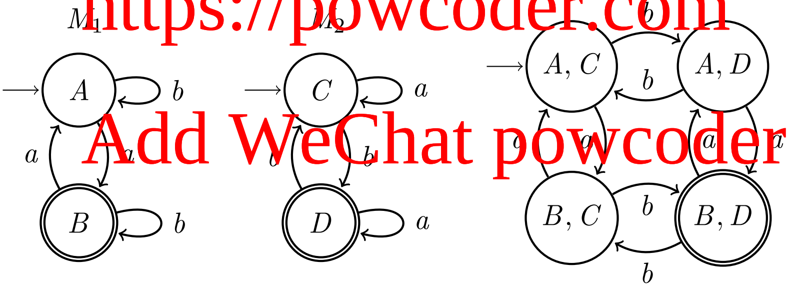
Note that M is also deterministic

Example Cross-product for Intersection

Example:

$L_1 = \{x|x \in \{a, b\}^*$ and x contains an odd number of a's$\}$

$L_2 = \{x|x \in \{a, b\}^*$ and x contains an odd number of b's$\}$

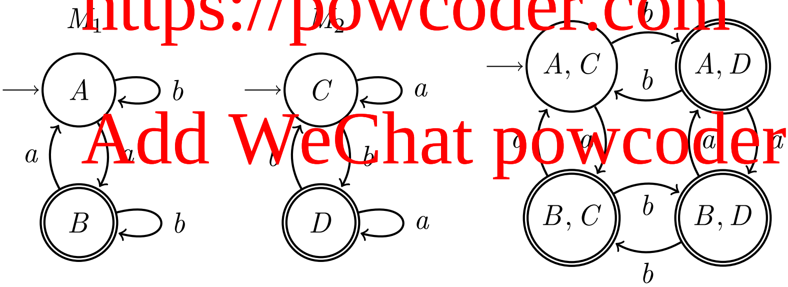


Cross-product can be used for Union

Example:

$L_1 = \{x | x \in \{a, b\}^*$ and x contains an odd number of a's$\}$

$L_2 = \{x | x \in \{a, b\}^*$ and x contains an odd number of b's$\}$



Outline

- Non-Deterministic Finite Automata (NFA)
  - Non-determinism
  - ε-transitions
- Equivalence of NFA and DFA
- Minimal DFA
- Regular Languages and Closure properties
- **Regular Expressions (introduction)**

Regular Expressions

In arithmetic we use operations to build up expressions, describing numbers such as

$(5 + 3) \times 4$ (value is 32)

$4/2 - 1$ (value is 1)

With regular languages, we use regular operations to build up expressions describing languages such as

$(0 \mid 1)0^\star$ (strings starting with 0 or 1, then any number of 0's)

$(ab)^\star a$ (any number of ab's, followed by a)

Regular Expressions vs Finite Automata

- RegEx: algebraic description of the strings in a language
- FA: a machine-like description

- RegEx serve as the input language for many systems which process strings
  - Search commands
    - UNIX grep
    - Web browsers
    - Text editors
    - Text formatting systems
    - ...
  - Lexical-analyser generators (Lex, Flex, etc.)

- Like FA, RegEx define exactly the class of *regular languages*

Definition of a Regular Expression (RegEx)

- Basic expressions and the language they describe
  - If a is any symbol of the input alphabet, then a is a RegEx and its language $L(a) = \{a\}$
  - ε is a RegEx and $L(\varepsilon) = \{\varepsilon\}$
  - $\emptyset$ is a RegEx and $L(\emptyset) = \emptyset$

Definition of a Regular Expression (RegEx)

- Basic expressions and the language they describe
  - If a is any symbol of the input alphabet, then a is a RegEx and its language $L(a) = \{a\}$
  - ε is a RegEx and $L(\varepsilon) = \{\varepsilon\}$
  - $\emptyset$ is a RegEx and $L(\emptyset) = \emptyset$

- Operators on RegEx. If R and S are RegEx, then
  - Union: $R \mid S$ (sometimes denoted $R + S$) is a RegEx
  - Concatenation: RS (sometimes denoted $R \circ S$) is a RegEx
  - Star Closure: $R^\star$ is a RegEx

DEFINITION OF A REGULAR EXPRESSION (REGEX)

- Basic expressions and the language they describe
  - If a is any symbol of the input alphabet, then a is a RegEx and its language $L(a) = \{a\}$
  - ε is a RegEx and $L(\varepsilon) = \{\varepsilon\}$
  - $\emptyset$ is a RegEx and $L(\emptyset) = \emptyset$
- Operators on RegEx. If R and S are RegEx, then
  - Union: $R \mid S$ (sometimes denoted $R + S$) is a RegEx
  - Concatenation: RS (sometimes denoted $R \circ S$) is a RegEx
  - Star Closure: $R^\star$ is a RegEx
- Precedence of operators: $^\star, \circ, \mid$ to avoid excessive parentheses
  - e.g. $R \mid ST^\star = (R \mid (S(T^\star)))$, similar to how $8 + 21/3^2 = (8 + (21/(3^2)))$
  - Use parentheses to change the order of operations

Language of a Regular Expression

Let x, y be symbols from the input alphabet

$$
\begin{aligned}
L(x) &= \{x\} \\
L(\varepsilon) &= \{\varepsilon\} \\
L(\emptyset) &= \{\emptyset\} \\
L(x \mid y) &= \{x, y\} \\
L(x^*) &= \{\varepsilon, x, xx, xxx, \ldots\} \\
L(xy) &= \{xy\}
\end{aligned}
$$

Operators on RegEx: Union $R \mid S$

Definition: $L(R \mid S) = L(R) \cup L(S)$

Operators on RegEx: Union $R \mid S$

Definition: $L(R \mid S) = L(R) \cup L(S)$

$L(a \mid b) =$

Operators on RegEx: Union $R \mid S$

Definition: $L(R \mid S) = L(R) \cup L(S)$

$$L(a \mid b) = L(a) \cup L(b) = \{a\} \cup \{b\} = \{a, b\}$$

$$L(b \mid a) =$$

Operators on RegEx: Union $R \mid S$

Definition: $L(R \mid S) = L(R) \cup L(S)$

$$L(a \mid b) = L(a) \cup L(b) = \{a\} \cup \{b\} = \{a, b\}$$

$$L(b \mid a) = L(b) \cup L(a) = \{b\} \cup \{a\} = \{a, b\} = L(a \mid b)$$

$L((a \mid b) \mid c) =$

Operators on RegEx: Union $R \mid S$

Definition: $L(R \mid S) = L(R) \cup L(S)$

$$L(a \mid b) = L(a) \cup L(b) = \{a\} \cup \{b\} = \{a, b\}$$

$$L(b \mid a) = L(b) \cup L(a) = \{b\} \cup \{a\} = \{a, b\} = L(a \mid b)$$

$$L((a \mid b) \mid c) = L(a \mid b) \cup L(c)$$

$$=$$

Operators on RegEx: Union $R \mid S$

Definition: $L(R \mid S) = L(R) \cup L(S)$

$$L(a \mid b) = L(a) \cup L(b) = \{a\} \cup \{b\} = \{a, b\}$$

$$L(b \mid a) = L(b) \cup L(a) = \{b\} \cup \{a\} = \{a, b\} = L(a \mid b)$$

$$\begin{aligned} L((a \mid b) \mid c) &= L(a \mid b) \cup L(c) \\ &= \{a, b\} \cup \{c\} \\ &= \end{aligned}$$

Operators on RegEx: Union $R \mid S$

Definition: $L(R \mid S) = L(R) \cup L(S)$

$$L(a \mid b) = L(a) \cup L(b) = \{a\} \cup \{b\} = \{a, b\}$$

$$L(b \mid a) = L(b) \cup L(a) = \{b\} \cup \{a\} = \{a, b\} = L(a \mid b)$$

$$\begin{aligned} L((a \mid b) \mid c) &= L(a \mid b) \cup L(c) \\ &= \{a, b\} \cup \{c\} \\ &= \{a, b, c\} \end{aligned}$$

$$L(a \mid (b \mid c)) =$$

Operators on RegEx: Union $R \mid S$

Definition: $L(R \mid S) = L(R) \cup L(S)$

$$L(a \mid b) = L(a) \cup L(b) = \{a\} \cup \{b\} = \{a, b\}$$

$$L(b \mid a) = L(b) \cup L(a) = \{b\} \cup \{a\} = \{a, b\} = L(a \mid b)$$

$$\begin{aligned} L((a \mid b) \mid c) &= L(a \mid b) \cup L(c) \\ &= \{a, b\} \cup \{c\} \\ &= \{a, b, c\} \end{aligned}$$

$$\begin{aligned} L(a \mid (b \mid c)) &= L(a) \cup L(b \mid c) \\ &= \end{aligned}$$

Operators on RegEx: Union $R \mid S$

Definition: $L(R \mid S) = L(R) \cup L(S)$

$$L(a \mid b) = L(a) \cup L(b) = \{a\} \cup \{b\} = \{a, b\}$$

$$L(b \mid a) = L(b) \cup L(a) = \{b\} \cup \{a\} = \{a, b\} = L(a \mid b)$$

$$\begin{aligned} L((a \mid b) \mid c) &= L(a \mid b) \cup L(c) \\ &= \{a, b\} \cup \{c\} \\ &= \{a, b, c\} \end{aligned}$$

$$\begin{aligned} L(a \mid (b \mid c)) &= L(a) \cup L(b \mid c) \\ &= \{a\} \cup \{b, c\} \\ &= \end{aligned}$$

Operators on RegEx: Union $R \mid S$

Definition: $L(R \mid S) = L(R) \cup L(S)$

$$L(a \mid b) = L(a) \cup L(b) = \{a\} \cup \{b\} = \{a, b\}$$

$$L(b \mid a) = L(b) \cup L(a) = \{b\} \cup \{a\} = \{a, b\} = L(a \mid b)$$

$$\begin{aligned} L((a \mid b) \mid c) &= L(a \mid b) \cup L(c) \\ &= \{a, b\} \cup \{c\} \\ &= \{a, b, c\} \end{aligned}$$

$$\begin{aligned} L(a \mid (b \mid c)) &= L(a) \cup L(b \mid c) \\ &= \{a\} \cup \{b, c\} \\ &= \{a, b, c\} \end{aligned}$$

Operators on RegEx: Union $R \mid S$

Definition: $L(R \mid S) = L(R) \cup L(S)$

$$L(a \mid b) = L(a) \cup L(b) = \{a\} \cup \{b\} = \{a, b\}$$

$$L(b \mid a) = L(b) \cup L(a) = \{b\} \cup \{a\} = \{a, b\} = L(a \mid b)$$

$$\begin{aligned} L((a \mid b) \mid c) &= L(a \mid b) \cup L(c) \\ &= \{a, b\} \cup \{c\} \\ &= \{a, b, c\} \end{aligned}$$

$$\begin{aligned} L(a \mid (b \mid c)) &= L(a) \cup L(b \mid c) \\ &= \{a\} \cup \{b, c\} \\ &= \{a, b, c\} \end{aligned}$$

Because it is associative we can write $L(a \mid b \mid c)$

Operators on RegEx: Concatenation RS

Definition: $L(RS) = \{rs \mid r \in L(R) \text{ and } s \in L(S)\}$

Operators on RegEx: Concatenation RS

Definition: $L(RS) = \{rs \mid r \in L(R)$ and $s \in L(S)\}$

$L(ab) =$

Operators on RegEx: Concatenation RS

Definition: $L(RS) = \{rs \mid r \in L(R) \text{ and } s \in L(S)\}$

$$L(ab) = \{rs \mid r \in L(a) \text{ and } s \in L(b)\} = \{ab\}$$

$$L(ba) =$$

Operators on RegEx: Concatenation RS

Definition: $L(RS) = \{rs \mid r \in L(R) \text{ and } s \in L(S)\}$

$$L(ab) = \{rs \mid r \in L(a) \text{ and } s \in L(b)\} = \{ab\}$$

$$L(ba) = \{rs \mid r \in L(b) \text{ and } s \in L(a)\} = \{ba\}$$

$L((ab)c) =$

Operators on RegEx: Concatenation RS

Definition: $L(RS) = \{rs \mid r \in L(R) \text{ and } s \in L(S)\}$

$$L(ab) = \{rs \mid r \in L(a) \text{ and } s \in L(b)\} = \{ab\}$$

$$L(ba) = \{rs \mid r \in L(b) \text{ and } s \in L(a)\} = \{ba\}$$

$$L((ab)c) = \{rs \mid r \in L(ab) \text{ and } s \in L(c)\}$$

$$=$$

Operators on RegEx: Concatenation RS

Definition: $L(RS) = \{rs \mid r \in L(R) \text{ and } s \in L(S)\}$

$$L(ab) = \{rs \mid r \in L(a) \text{ and } s \in L(b)\} = \{ab\}$$

$$L(ba) = \{rs \mid r \in L(b) \text{ and } s \in L(a)\} = \{ba\}$$

$$\begin{aligned} L((ab)c) &= \{rs \mid r \in L(ab) \text{ and } s \in L(c)\} \\ &= \{rs \mid r \in \{ab\} \text{ and } s \in \{c\}\} \\ &= \end{aligned}$$

Operators on RegEx: Concatenation RS

Definition: $L(RS) = \{rs \mid r \in L(R) \text{ and } s \in L(S)\}$

$$
\begin{aligned}
L(ab) &= \{rs \mid r \in L(a) \text{ and } s \in L(b)\} = \{ab\} \\
L(ba) &= \{rs \mid r \in L(b) \text{ and } s \in L(a)\} = \{ba\} \\
L((ab)c) &= \{rs \mid r \in L(ab) \text{ and } s \in L(c)\} \\
&= \{rs \mid r \in \{ab\} \text{ and } s \in \{c\}\} \\
&= \{abc\} \\
L(a(bc)) &=
\end{aligned}
$$

Operators on RegEx: Concatenation RS

Definition: $L(RS) = \{rs \mid r \in L(R) \text{ and } s \in L(S)\}$

$$L(ab) = \{rs \mid r \in L(a) \text{ and } s \in L(b)\} = \{ab\}$$

$$L(ba) = \{rs \mid r \in L(b) \text{ and } s \in L(a)\} = \{ba\}$$

$$\begin{aligned} L((ab)c) &= \{rs \mid r \in L(ab) \text{ and } s \in L(c)\} \\ &= \{rs \mid r \in \{ab\} \text{ and } s \in \{c\}\} \\ &= \{abc\} \end{aligned}$$

$$\begin{aligned} L(a(bc)) &= \{rs \mid r \in L(a) \text{ and } s \in L(bc)\} \\ &= \end{aligned}$$

Operators on RegEx: Concatenation RS

Definition: $L(RS) = \{rs \mid r \in L(R) \text{ and } s \in L(S)\}$

$$L(ab) = \{rs \mid r \in L(a) \text{ and } s \in L(b)\} = \{ab\}$$

$$L(ba) = \{rs \mid r \in L(b) \text{ and } s \in L(a)\} = \{ba\}$$

$$\begin{aligned} L((ab)c) &= \{rs \mid r \in L(ab) \text{ and } s \in L(c)\} \\ &= \{rs \mid r \in \{ab\} \text{ and } s \in \{c\}\} \\ &= \{abc\} \end{aligned}$$

$$\begin{aligned} L(a(bc)) &= \{rs \mid r \in L(a) \text{ and } s \in L(bc)\} \\ &= \{rs \mid r \in \{a\} \text{ and } s \in \{bc\}\} \\ &= \end{aligned}$$

Operators on RegEx: Concatenation RS

Definition: $L(RS) = \{rs \mid r \in L(R) \text{ and } s \in L(S)\}$

$$L(ab) = \{rs \mid r \in L(a) \text{ and } s \in L(b)\} = \{ab\}$$

$$L(ba) = \{rs \mid r \in L(b) \text{ and } s \in L(a)\} = \{ba\}$$

$$\begin{aligned} L((ab)c) &= \{rs \mid r \in L(ab) \text{ and } s \in L(c)\} \\ &= \{rs \mid r \in \{ab\} \text{ and } s \in \{c\}\} \\ &= \{abc\} \end{aligned}$$

$$\begin{aligned} L(a(bc)) &= \{rs \mid r \in L(a) \text{ and } s \in L(bc)\} \\ &= \{rs \mid r \in \{a\} \text{ and } s \in \{bc\}\} \\ &= \{abc\} \end{aligned}$$

Operators on RegEx: Concatenation RS

Definition: $L(RS) = \{rs \mid r \in L(R) \text{ and } s \in L(S)\}$

$$L(ab) = \{rs \mid r \in L(a) \text{ and } s \in L(b)\} = \{ab\}$$

$$L(ba) = \{rs \mid r \in L(b) \text{ and } s \in L(a)\} = \{ba\}$$

$$\begin{aligned} L((ab)c) &= \{rs \mid r \in L(ab) \text{ and } s \in L(c)\} \\ &= \{rs \mid r \in \{ab\} \text{ and } s \in \{c\}\} \\ &= \{abc\} \end{aligned}$$

$$\begin{aligned} L(a(bc)) &= \{rs \mid r \in L(a) \text{ and } s \in L(bc)\} \\ &= \{rs \mid r \in \{a\} \text{ and } s \in \{bc\}\} \\ &= \{abc\} \end{aligned}$$

Because it is associative we can write $L(abc)$

Union and Concatenation

$$L((a \mid b)c) =$$

$$L((ab) \mid c) =$$

Union and Concatenation

$$L((a \mid b)c) = \{rs \mid r \in L(a \mid b) \text{ and } s \in L(c)\}$$

$$=$$

$$L((ab) \mid c) =$$

Union and Concatenation

$$
\begin{aligned}
L((a \mid b)c) &= \{rs \mid r \in L(a \mid b) \text{ and } s \in L(c)\} \\
&= \{rs \mid r \in (\{a\} \cup \{b\}) \text{ and } s \in \{c\}\} \\
&=
\end{aligned}
$$

$$L((ab) \mid c) =$$

Union and Concatenation

$$
\begin{aligned}
L((a \mid b)c) &= \{rs \mid r \in L(a \mid b) \text{ and } s \in L(c)\} \\
&= \{rs \mid r \in (\{a\} \cup \{b\}) \text{ and } s \in \{c\}\} \\
&= \{rs \mid r \in \{a, b\} \text{ and } s \in \{c\}\} \\
&=
\end{aligned}
$$

$$L((ab) \mid c) =$$

Union and Concatenation

$$\begin{aligned} L((a \mid b)c) &= \{rs \mid r \in L(a \mid b) \text{ and } s \in L(c)\} \\ &= \{rs \mid r \in (\{a\} \cup \{b\}) \text{ and } s \in \{c\}\} \\ &= \{rs \mid r \in \{a, b\} \text{ and } s \in \{c\}\} \\ &= \{ac, bc\} \end{aligned}$$

$$L((ab) \mid c) =$$

Union and Concatenation

$$
\begin{aligned}
L((a \mid b)c) &= \{rs \mid r \in L(a \mid b) \text{ and } s \in L(c)\} \\
&= \{rs \mid r \in (\{a\} \cup \{b\}) \text{ and } s \in \{c\}\} \\
&= \{rs \mid r \in \{a, b\} \text{ and } s \in \{c\}\} \\
&= \{ac, bc\}
\end{aligned}
$$

$$
\begin{aligned}
L((ab) \mid c) &= L(ab) \cup L(c) \\
&=
\end{aligned}
$$

Union and Concatenation

$$\begin{aligned} L((a \mid b)c) &= \{rs \mid r \in L(a \mid b) \text{ and } s \in L(c)\} \\ &= \{rs \mid r \in (\{a\} \cup \{b\}) \text{ and } s \in \{c\}\} \\ &= \{rs \mid r \in \{a, b\} \text{ and } s \in \{c\}\} \\ &= \{ac, bc\} \end{aligned}$$

$$\begin{aligned} L((ab) \mid c) &= L(ab) \cup L(c) \\ &= \{rs \mid r \in L(a) \text{ and } s \in L(b)\} \cup L(c) \\ &= \end{aligned}$$

Union and Concatenation

$$L((a \mid b)c) = \{rs \mid r \in L(a \mid b) \text{ and } s \in L(c)\}$$
$$= \{rs \mid r \in (\{a\} \cup \{b\}) \text{ and } s \in \{c\}\}$$
$$= \{rs \mid r \in \{a, b\} \text{ and } s \in \{c\}\}$$
$$= \{ac, bc\}$$

$$L((ab) \mid c) = L(ab) \cup L(c)$$
$$= \{rs \mid r \in L(a) \text{ and } s \in L(b)\} \cup L(c)$$
$$= \{rs \mid r \in \{a\} \text{ and } s \in \{b\}\} \cup \{c\}$$
$$=$$

Union and Concatenation

$$
\begin{aligned}
L((a \mid b)c) &= \{rs \mid r \in L(a \mid b) \text{ and } s \in L(c)\} \\
&= \{rs \mid r \in (\{a\} \cup \{b\}) \text{ and } s \in \{c\}\} \\
&= \{rs \mid r \in \{a, b\} \text{ and } s \in \{c\}\} \\
&= \{ac, bc\}
\end{aligned}
$$

$$
\begin{aligned}
L((ab) \mid c) &= L(ab) \cup L(c) \\
&= \{rs \mid r \in L(a) \text{ and } s \in L(b)\} \cup L(c) \\
&= \{rs \mid r \in \{a\} \text{ and } s \in \{b\}\} \cup \{c\} \\
&= \{ab\} \cup \{c\} \\
&= \{ab, c\}
\end{aligned}
$$

Operators on RegEx: Star closure $R^\star$

$R^\star$ means 0 or more occurrences of R

$L(R^\star) = \{\varepsilon\} \cup L(R) \cup L(RR) \cup L(RRR) \cup \ldots$

$$L(0^\star) = \{\varepsilon, 0, 00, 000, \ldots\}$$
$$L((01)^\star) = \{\varepsilon, 01, 0101, 010101, \ldots\}$$
$$L(\emptyset^\star) =$$

Operators on RegEx: Star closure $R^\star$

$R^\star$ means 0 or more occurrences of R

$L(R^\star) = \{\varepsilon\} \cup L(R) \cup L(RR) \cup L(RRR) \cup ...$

$$L(0^\star) = \{\varepsilon, 0, 00, 000, ...\}$$

$$L((01)^\star) = \{\varepsilon, 01, 0101, 010101, ...\}$$

$$L(\emptyset^\star) = \{\varepsilon\} \cup L(\emptyset) \cup ... = \{\varepsilon\}$$

$$L(a \mid bc^\star) =$$

Operators on RegEx: Star closure $R^\star$

$R^\star$ means 0 or more occurrences of R

$L(R^\star) = \{\varepsilon\} \cup L(R) \cup L(RR) \cup L(RRR) \cup \ldots$

$$L(0^\star) = \{\varepsilon, 0, 00, 000, \ldots\}$$

$$L((01)^\star) = \{\varepsilon, 01, 0101, 010101, \ldots\}$$

$$L(\emptyset^\star) = \{\varepsilon\} \cup L(\emptyset) \cup \ldots = \{\varepsilon\}$$

$$L(a \mid bc^\star) = \{a, b, bc, bcc, bccc, \ldots\}$$

$$L(a \mid (bc)^\star) =$$

Operators on RegEx: Star closure $R^\star$

$R^\star$ means 0 or more occurrences of R
$L(R^\star) = \{\varepsilon\} \cup L(R) \cup L(RR) \cup L(RRR) \cup \ldots$

$$
\begin{aligned}
L(0^\star) &= \{\varepsilon, 0, 00, 000, \ldots\} \\
L((01)^\star) &= \{\varepsilon, 01, 0101, 010101, \ldots\} \\
L(\emptyset^\star) &= \{\varepsilon\} \cup L(\emptyset) \cup \ldots = \{\varepsilon\} \\
L(a \mid bc^\star) &= \{a, b, bc, bcc, bccc, \ldots\} \\
L(a \mid (bc)^\star) &= \{a, \varepsilon, bc, bcbc, bcbcbc, \ldots\} \\
L((a \mid b)\, c^\star) &=
\end{aligned}
$$

Operators on RegEx: Star closure $R^\star$

$R^\star$ means 0 or more occurrences of R

$L(R^\star) = \{\varepsilon\} \cup L(R) \cup L(RR) \cup L(RRR) \cup \ldots$

$$
\begin{aligned}
L(0^\star) &= \{\varepsilon, 0, 00, 000, \ldots\} \\
L((01)^\star) &= \{\varepsilon, 01, 0101, 010101, \ldots\} \\
L(\emptyset^\star) &= \{\varepsilon\} \cup L(\emptyset) \cup \ldots = \{\varepsilon\} \\
L(a \mid bc^\star) &= \{a, b, bc, bcc, bccc, \ldots\} \\
L(a \mid (bc)^\star) &= \{a, \varepsilon, bc, bcbc, bcbcbc, \ldots\} \\
L((a \mid b)\, c^\star) &= \{a, b, ac, bc, acc, bcc, accc, bccc, \ldots\}
\end{aligned}
$$

Some properties of RegEx

Let R, S, T be regular expressions

- Union properties:
  - $R \mid S = S \mid R$ (commutative)
  - $R \mid \emptyset = \emptyset \mid R = R$
  - $R \mid R = R$
  - $(R \mid S) \mid T = R \mid (S \mid T)$ (associative)

- Concatenation properties:

- Union and concatenation are *distributive* when combined:

SOME PROPERTIES OF REGEX

Let R, S, T be regular expressions

- Union properties:
  - $R \mid S = S \mid R$ (commutative)
  - $R \mid \emptyset = \emptyset \mid R = R$
  - $R \mid R = R$
  - $(R \mid S) \mid T = R \mid (S \mid T)$ (associative)

- Concatenation properties:
  - $R\emptyset = \emptyset R = \emptyset$
  - $R\varepsilon = \varepsilon R = R$
  - $(RS)T = R(ST)$ (associative)

- Union and concatenation are *distributive* when combined:

Some properties of RegEx

Let R, S, T be regular expressions

- Union properties:
  - $R \mid S = S \mid R$ (commutative)
  - $R \mid \emptyset = \emptyset \mid R = R$
  - $R \mid R = R$
  - $(R \mid S) \mid T = R \mid (S \mid T)$ (associative)
- Concatenation properties:
  - $R\emptyset = \emptyset R = \emptyset$
  - $R\varepsilon = \varepsilon R = R$
  - $(RS)T = R(ST)$ (associative)
- Union and concatenation are *distributive* when combined:
  - $R(S \mid T) = RS \mid RT$
  - $(S \mid T)R = SR \mid TR$

Some properties of RegEx

Let R, S, T be regular expressions

- Star Closure properties:
    - $\emptyset^\star \equiv \varepsilon^\star \equiv \varepsilon$

Some properties of RegEx

Let R, S, T be regular expressions

- Star Closure properties:
    - $\emptyset^\star = \varepsilon^\star = \varepsilon$
    - $R^\star = R^\star R^\star = (R^\star)^\star = R \mid R^\star$

Some properties of RegEx

Let R, S, T be regular expressions

- Star Closure properties:
  - $\emptyset^\star = \varepsilon^\star = \varepsilon$
  - $R^\star = R^\star R^\star = (R^\star)^\star = R \mid R^\star$
  - $R^\star = \varepsilon \mid R^\star = (\varepsilon \mid R)^\star = (\varepsilon \mid R)R^\star = \varepsilon \mid RR^\star$

Some properties of RegEx

Let R, S, T be regular expressions

- Star Closure properties:
  - $\emptyset^\star = \varepsilon^\star = \varepsilon$
  - $R^\star = R^\star R^\star = (R^\star)^\star = R \mid R^\star$
  - $R^\star = \varepsilon \mid R^\star = (\varepsilon \mid R)^\star = (\varepsilon \mid R)R^\star = \varepsilon \mid RR^\star$
  - $RR^\star = R^\star R$

Some properties of RegEx

Let R, S, T be regular expressions

- Star Closure properties:
  - $\emptyset^\star = \varepsilon^\star = \varepsilon$
  - $R^\star = R^\star R^\star = (R^\star)^\star = R \mid R^\star$
  - $R^\star = \varepsilon \mid R^\star = (\varepsilon \mid R)^\star = (\varepsilon \mid R)R^\star = \varepsilon \mid RR^\star$
  - $RR^\star = R^\star R$
  - $(R \mid S)^\star = (R^\star \mid S^\star)^\star = (R^\star S^\star)^\star = (R^\star S)^\star R^\star = R^\star (SR^\star)^\star$

Some properties of RegEx

Let R, S, T be regular expressions

- Star Closure properties:

  - $\emptyset^\star = \varepsilon^\star = \varepsilon$
  - $R^\star = R^\star R^\star = (R^\star)^\star = R \mid R^\star$
  - $R^\star = \varepsilon \mid R^\star = (\varepsilon \mid R)^\star = (\varepsilon \mid R)R^\star = \varepsilon \mid RR^\star$
  - $RR^\star = R^\star R$
  - $(R \mid S)^\star = (R^\star \mid S^\star)^\star = (R^\star S^\star)^\star = (R^\star S)^\star R^\star = R^\star(SR^\star)^\star$
  - $R(SR)^\star = (RS)^\star R$

Some properties of RegEx

Let R, S, T be regular expressions

- Star Closure properties:
  - $\emptyset^\star = \varepsilon^\star = \varepsilon$
  - $R^\star = R^\star R^\star = (R^\star)^\star = R \mid R^\star$
  - $R^\star = \varepsilon \mid R^\star = (\varepsilon \mid R)^\star = (\varepsilon \mid R)R^\star = \varepsilon \mid RR^\star$
  - $RR^\star = R^\star R$
  - $(R \mid S)^\star = (R^\star \mid S^\star)^\star = (R^\star S^\star)^\star = (R^\star S)^\star R^\star = R^\star(SR^\star)^\star$
  - $R(SR)^\star = (RS)^\star R$
  - $(R^\star S)^\star = \varepsilon \mid (R \mid S)^\star S$

Some properties of RegEx

Let R, S, T be regular expressions

- Star Closure properties:
  - $\emptyset^\star = \varepsilon^\star = \varepsilon$
  - $R^\star = R^\star R^\star = (R^\star)^\star = R \mid R^\star$
  - $R^\star = \varepsilon \mid R^\star = (\varepsilon \mid R)^\star = (\varepsilon \mid R)R^\star = \varepsilon \mid RR^\star$
  - $RR^\star = R^\star R$
  - $(R \mid S)^\star = (R^\star \mid S^\star)^\star = (R^\star S^\star)^\star = (R^\star S)^\star R^\star = R^\star(SR^\star)^\star$
  - $R(SR)^\star = (RS)^\star R$
  - $(R^\star S)^\star = \varepsilon \mid (R \mid S)^\star S$
  - $(RS^\star)^\star = \varepsilon \mid R(R \mid S)^\star$

Examples of RegEx

- All possible strings which can be formed with a, b, or c

- Strings over $\{0, 1\}$ which end with 01

- Traffic lights?

- Identifiers for Java programs?

Examples of RegEx

- All possible strings which can be formed with a, b, or c $(a \mid b \mid c)^*$
- Strings over $\{0, 1\}$ which end with 01
- Traffic lights?
- Identifiers for Java programs?

Examples of RegEx

- All possible strings which can be formed with a, b, or c
  $(a \mid b \mid c)^{\star}$

- Strings over $\{0, 1\}$ which end with 01
  $(0 \mid 1)^{\star}01$

- Traffic lights?

- Identifiers for Java programs?

Examples of RegEx

- All possible strings which can be formed with a, b, or c
  $(a \mid b \mid c)^\star$

- Strings over $\{0, 1\}$ which end with 01
  $(0 \mid 1)^\star 01$

- Traffic lights?
  $(red \mid red \circ amber \mid amber \mid green)$

- Identifiers for Java programs?

Examples of RegEx

- All possible strings which can be formed with a, b, or c
  $(a \mid b \mid c)^\star$

- Strings over $\{0, 1\}$ which end with 01
  $(0 \mid 1)^\star 01$

- Traffic lights?
  $(\mathit{red} \mid \mathit{red} \circ \mathit{amber} \mid \mathit{amber} \mid \mathit{green})$

- Identifiers for Java programs?
  $(\mathit{letter} \mid \$ \mid \_)(\mathit{letter} \mid \mathit{digit} \mid \$ \mid \_)^\star$

Examples of RegEx

- $a^\star b a^\star$ represents

- $((a \mid b)(a \mid b))^\star$ represents

- $ab^\star a \mid ba^\star b \mid a \mid b$ represents

- $a^\star \emptyset$ represents

Examples of RegEx

- $a^\star b a^\star$ represents

  Strings over $\{a, b\}$ with exactly one b

- $((a \mid b)(a \mid b))^\star$ represents

- $ab^\star a \mid ba^\star b \mid a \mid b$ represents

- $a^\star \emptyset$ represents

Examples of RegEx

- $a^\star b a^\star$ represents
  Strings over $\{a, b\}$ with exactly one b

- $((a \mid b)(a \mid b))^\star$ represents
  Strings over $\{a, b\}$ with even length

- $ab^\star a \mid ba^\star b \mid a \mid b$ represents

- $a^\star \emptyset$ represents

Examples of RegEx

- $a^\star b a^\star$ represents
  Strings over $\{a, b\}$ with exactly one b

- $((a \mid b)(a \mid b))^\star$ represents
  Strings over $\{a, b\}$ with even length

- $ab^\star a \mid ba^\star b \mid a \mid b$ represents
  Strings of a's starting and ending with b, or strings of b's starting and ending with a, or a single a or b

- $a^\star \emptyset$ represents

Examples of RegEx

- $a^\star b a^\star$ represents
  Strings over $\{a, b\}$ with exactly one b

- $((a \mid b)(a \mid b))^\star$ represents
  Strings over $\{a, b\}$ with even length

- $ab^\star a \mid ba^\star b \mid a \mid b$ represents
  Strings of a's starting and ending with b, or strings of b's starting and ending with a, or a single a or b

- $a^\star \emptyset$ represents
  The empty language, $\emptyset$

Equivalence of RegEx and FA

Theorem:

A language is regular if and only if a regular expression describes it.

Proof:

Show the equivalence of RegEx and FA (RegEx ⇔ FA)

1. RegEx ⇒ FA:
   Show that for each RegEx, there exists an NFA which recognises the same language
2. FA ⇒ RegEx:
   Show that for each NFA, there exists a RegEx which recognises the same language

Equivalence of RegEx and FA

Theorem:
A language is regular if and only if a regular expression describes it.

Proof:
Show the equivalence of RegEx and FA (RegEx ⇔ FA)

1. RegEx ⇒ FA:
   Show that for each RegEx, there exists an NFA which recognises the same language
2. FA ⇒ RegEx:
   Show that for each NFA, there exists a RegEx which recognises the same language

Next week: proof

Application: pattern matching

- Provides a technique for finding occurrences of patterns in text (e.g. web/document searching)

- Some of the best known algorithms are based on Finite Automata!
- Constructing a NFA to recognise the strings containing the pattern is trivial.

- Then we convert the NFA to DFA, then minimal DFA, so the pattern can be matched efficiently

DFAs and NFAs

- NFAs and DFAs define the class of regular languages
- Each NFA can be transformed into a unique minimal DFA
- NFAs are often more intuitive to build and easier to understand

- DFA - *especially minimal DFA* - are more efficient to compute
    - despite the fact they often have far more states than the NFA
- Regular languages are closed under the union, concatenation, star closure, intersection, and complement operations
    - We use these to build the NFA, before converting to a minimal DFA

Regular Expressions

- What they are and how to use them
- Regular operations
- Equivalence with DFA/NFA