COMP2022: Formal Languages and Logic

2018, Semester 2, Week 5

THE UNIVERSITY OF SYDNEY

Outline

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

-

- Minimal DFA

- Regular Languages and Closure properti

- Regular Expressions (introduction)

Non deterministic Finite Automata (NFA)

DFA

- has exactly one transition per input from each state
-

NFA

- can have any number of transitions per input f
- so some steps of the computation might be stic
- can also have ε-transitions
  - i.e. transitions which the automaton can follow without scanning any input

NON-DETERMINISM

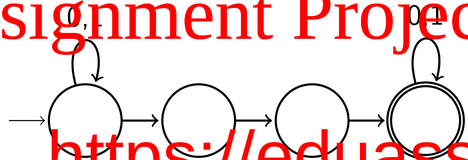
- As we scan through a string, the NFA can be in many states at the same time
- *set* of

- to a final state
  - i.e., if we are in at least one accept state, after re input

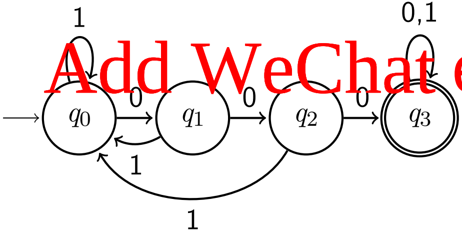
- Parallel computation

EXAMPLE

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

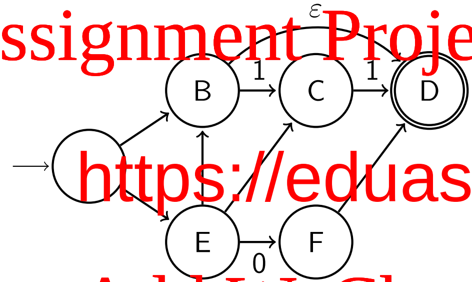


The c



NFA with ε-Transitions

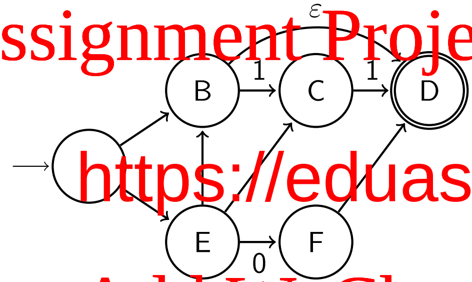
Example



- From A, with a 0:
- From A, with a 1:

⋮

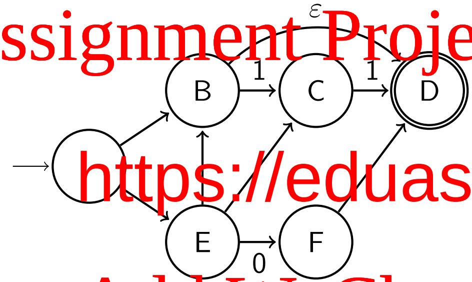
Example



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

$\vdots$

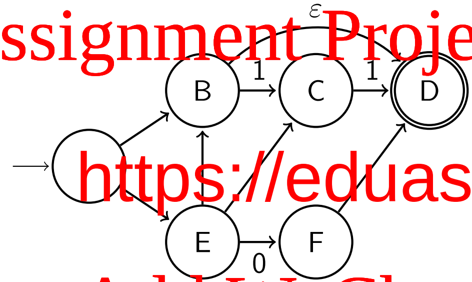
Example



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

⋮

Example

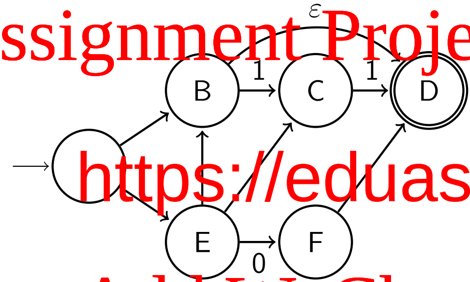


- From A, with a 0: B
- From A, with a 1: E

: D

:

Example

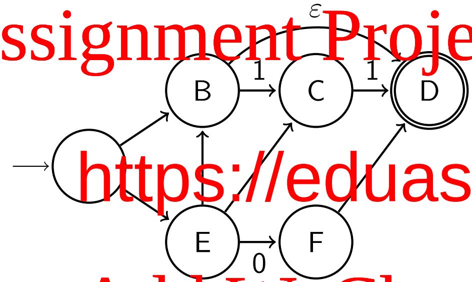


- From A, with a 0: B
- From A, with a 1: E

: D

: B and C

Example



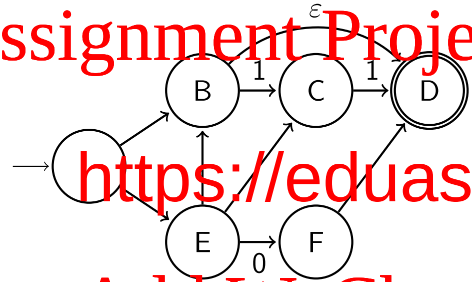
- From A, with a 0: B
- From A, with a 1: E

: D

: 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

: D

: B and C

			ε
			$\}$
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,
-
-
-

- $F \subseteq Q$ 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

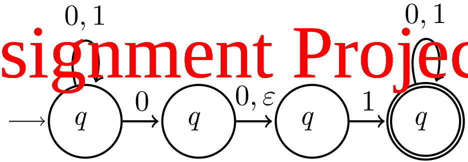
$$\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$$

-

$$\emptyset$$

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

Example



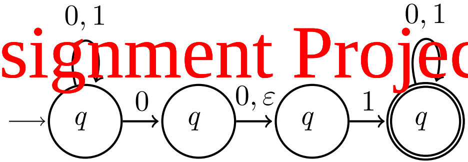
- $Q = \{q_1, q_2, q_3, q_4\}$
- $\Sigma = ?$
- The start state is q_1
- The set of accept states is $\{q_4\}$

by:

			ε
q_1			
q_2			
q_3			
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$

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

Example



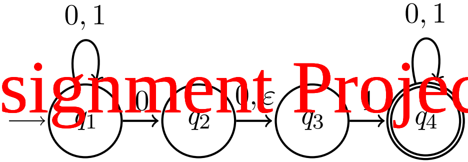
- Q 1 2 3 4
- $\Sigma = \{0, 1\}$
- The start state is q_1
- The set of accept states is $\{q_4\}$

by.

			ε
q_1			
q_2			
q_3			
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$

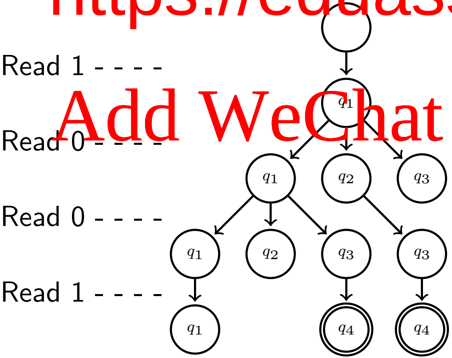
Some strings where N reaches the accept state: $01, 0000001$

NFA computation : Parallelism



Seve

a tree o



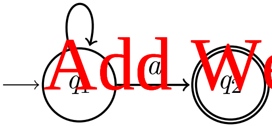
Language of an NFA

- A string w is accepted by an NFA if at least one sequence of

- The language recognised by an NFA (i.e. the accepts) is called regular

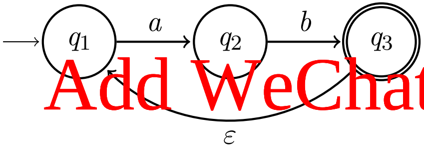
Examples of NFA (1)

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 su



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

- Minimal DFA

- Regular Languages and Closure properti
- 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
-

But ar
by a DFA?

No. If a language is recognised by a NFA then we can al
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

The
The cl
languages accepted by NFAs

$\Longrightarrow$ Any regular language must be accepted b
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 s [illegible] cons [illegible]

When we transition from a state X, wit [illegible] i we must also consider all the ways in which we could f [illegible] 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

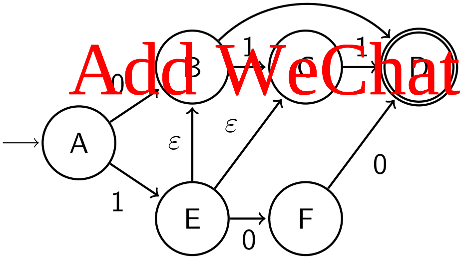
The *Epsilon-closure* of a state q, denoted $E(q)$, is the set of all states which can be reached from q by 0 or more ε-transitions

- $q \in E(q)$
-

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

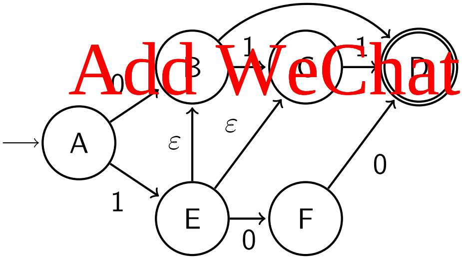
- $q \in E(q)$
-



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

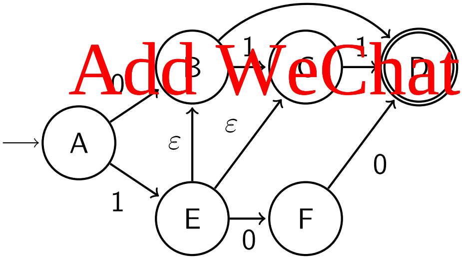
- $q \in E(q)$
-



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

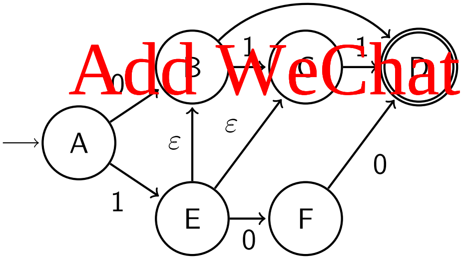
- $q \in E(q)$
-



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

- $q \in E(q)$
-



$E\}$

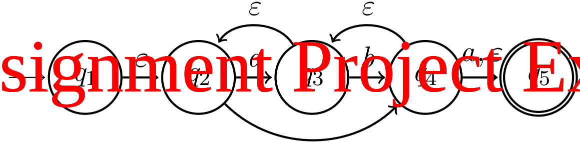
Epsilon-Closure for sets

The E

unio

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

Constructing epsilon-closures



	[illegible]	[illegible]	[illegible]
q_1	[illegible]	[illegible]	[illegible]
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	$\emptyset$	$\emptyset$	$\emptyset$

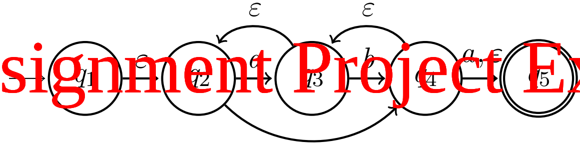
E [illegible]

E [illegible]

E [illegible]

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

Constructing epsilon-closures



q_1			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	$\emptyset$	$\emptyset$	$\emptyset$

2

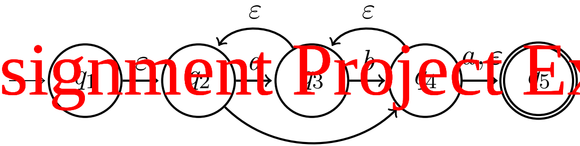
E

E

E

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

Constructing epsilon-closures



q_1			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	$\emptyset$	$\emptyset$	$\emptyset$

2 2

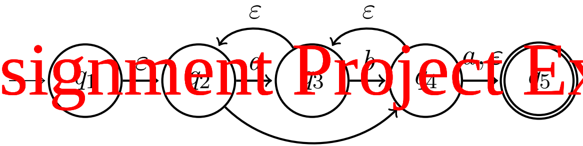
E

E

E

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

CONSTRUCTING EPSILON-CLOSURES



q_1			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	$\emptyset$	$\emptyset$	$\emptyset$

2 2

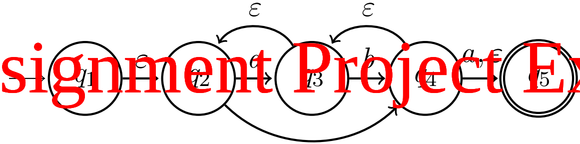
E

E

E

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

Constructing epsilon-closures



	[illegible]	[illegible]	[illegible]
q_1	[illegible]	[illegible]	[illegible] 2 [illegible]
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	$\emptyset$	$\emptyset$	$\emptyset$

[illegible] 2 [illegible] 2 [illegible]

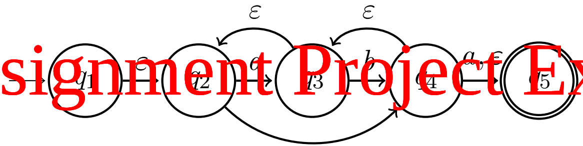
E [illegible]

E [illegible]

E [illegible]

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

Constructing epsilon-closures



	[illegible]	[illegible]	[illegible]
q_1	[illegible]	[illegible]	[illegible]
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	$\emptyset$	$\emptyset$	$\emptyset$

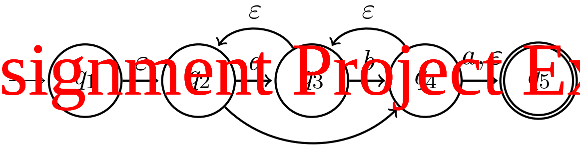
E

E

E

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

Constructing epsilon-closures



	[illegible]	[illegible]	[illegible]
q_1	[illegible]	[illegible]	[illegible]
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	$\emptyset$	$\emptyset$	$\emptyset$

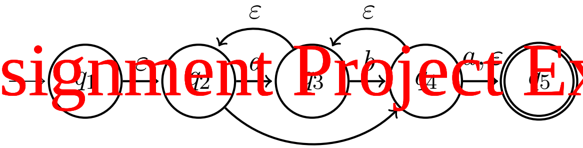
E [illegible]

E [illegible]

E [illegible]

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

Constructing epsilon-closures



	[illegible]	[illegible]	[illegible]
q_1	[illegible]	[illegible]	[illegible]
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 [illegible]
E [illegible]
E [illegible]
E [illegible]

$$\begin{aligned} & E(\{q_1, q_3, q_5\}) \\ &= E(\{q_1\}) \cup E(\{q_3\}) \cup E(\{q_5\}) \\ &= \{q_1, q_2, q_3, q_5\} \end{aligned}$$

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 $L(N)$

-
- 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

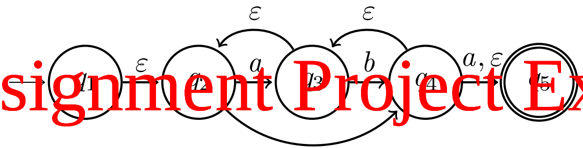
We want to convert a NFA $N = (Q, \Sigma, \delta, q_0, F)$ into an equivalent DFA $M = (Q', \Sigma, \delta', q_0', F')$ which recognises $L(N)$

Algo

1.
2. $R \subseteq \mathcal{P}(Q)$ a Σ e following DFA state as a DFA table entry in eit
   - $\delta'(R, a) = E(\cup_{r \in R} \delta(r, a))$ clos
   - $\delta'(R, a) = \cup_{r \in R} E(\delta(r, a))$ uni

3. A DFA state is accepting if any of its elements are an NFA accept state.

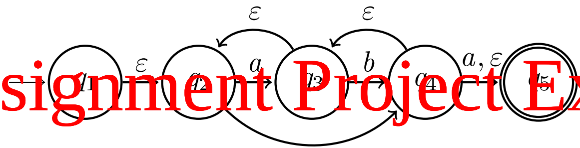
From NFA to DFA: example



Tran

Resulting DFA:

From NFA to DFA: example

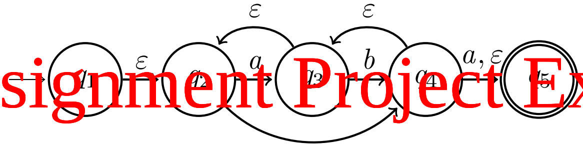


Tran

$E(start) =$		

Resulting DFA:

From NFA to DFA: example

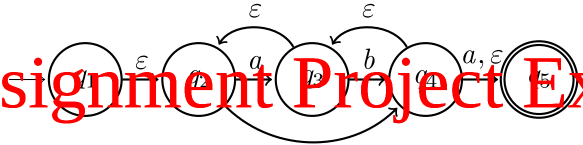


Tran

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

Resulting DFA:

From NFA to DFA: example

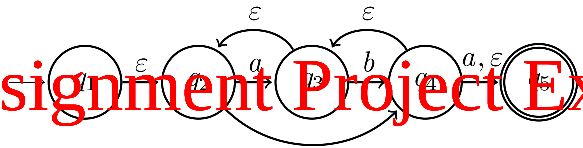


Tran

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

Resulting DFA:

From NFA to DFA: example

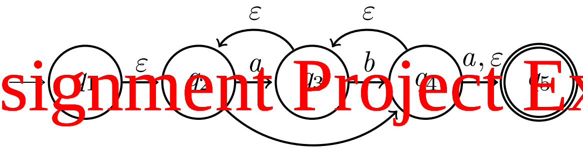


Tran

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

Resulting DFA:

From NFA to DFA: example

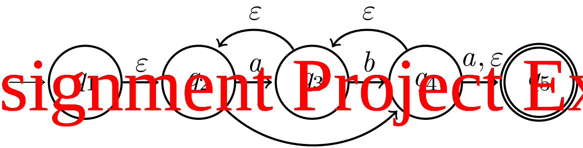


Tran

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

Resulting DFA:

From NFA to DFA: example

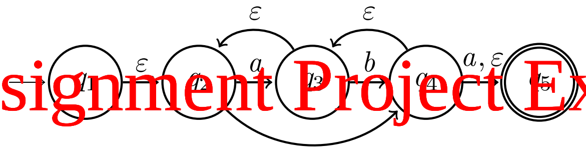


Tran

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

Resulting DFA:

From NFA to DFA: example

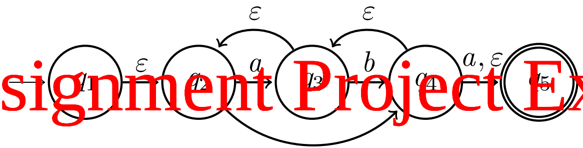


Tran

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

Resulting DFA:

From NFA to DFA: example

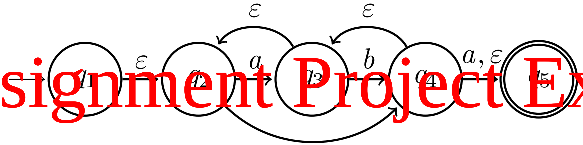


Tran

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

Resulting DFA:

From NFA to DFA: example

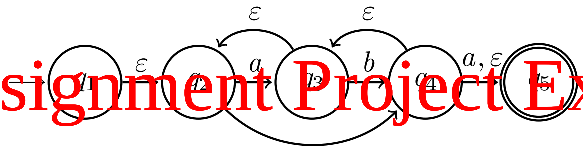


Tran

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

Resulting DFA:

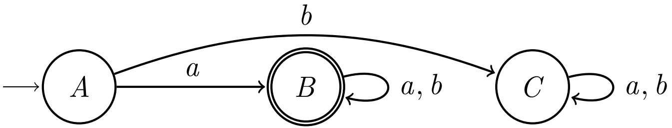
From NFA to DFA: example



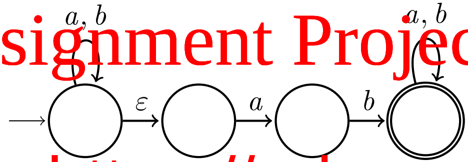
Tran

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

Resulting DFA:



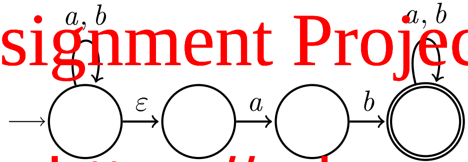
From NFA to DFA: example



Tran

	a	

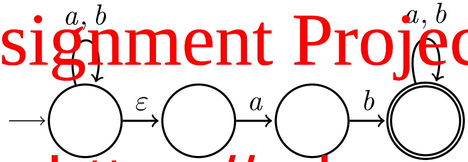
From NFA to DFA: example



Tran

	a	
$E(start) =$		

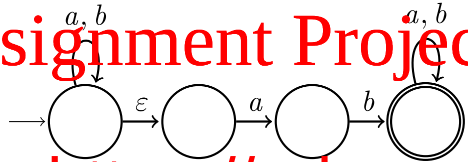
From NFA to DFA: example



Tran

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

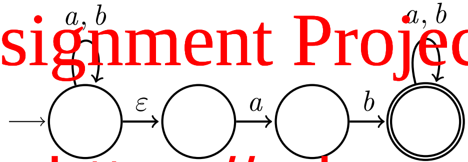
From NFA to DFA: example



Tran

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

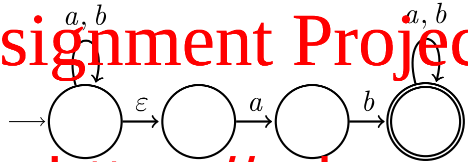
From NFA to DFA: example



Tran

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

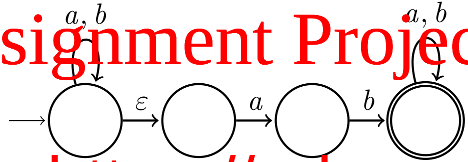
From NFA to DFA: example



Tran

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

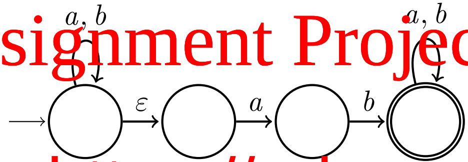
From NFA to DFA: example



Tran

	a	
$E(start) = \{q_1, q_2\}$	$\{q_1, q_2,$ [illegible]	
$\{q_1, q_2, q_3\}$	$\{q_1, q_2,$ [illegible] 3	

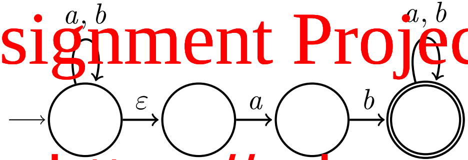
From NFA to DFA: example



Tran

	a	
$E(start) = \{q_1, q_2\}$	$\{q_1, q_2,$	
$\{q_1, q_2, q_3\}$	$\{q_1, q_2,$ 3	1 2 4

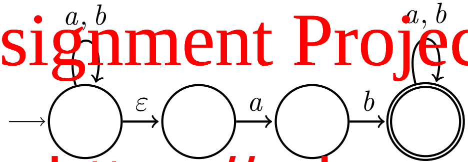
From NFA to DFA: example



Tran

	a	
$E(start) = \{q_1, q_2\}$	$\{q_1, q_2,$	
$\{q_1, q_2, q_3\}$	$\{q_1, q_2,$ 3	1 2 4
$\{q_1, q_2, q_4\}$		

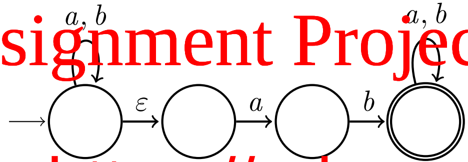
From NFA to DFA: example



Tran

	a	
$E(start) = \{q_1, q_2\}$	$\{q_1, q_2,$	
$\{q_1, q_2, q_3\}$	$\{q_1, q_2,$ 3	1 2 4
$\{q_1, q_2, q_4\}$	$\{q_1, q_2, q_3, q_4\}$	

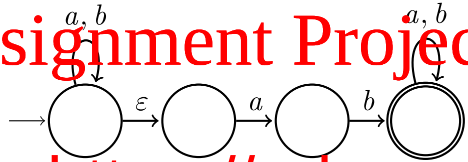
From NFA to DFA: example



Tran

	a	
$E(start) = \{q_1, q_2\}$	$\{q_1, q_2,$	
$\{q_1, q_2, q_3\}$	$\{q_1, q_2,$ 3	1 2 4
$\{q_1, q_2, q_4\}$	$\{q_1, q_2, q_3, q_4\}$	
$\{q_1, q_2, q_3, q_4\}$		

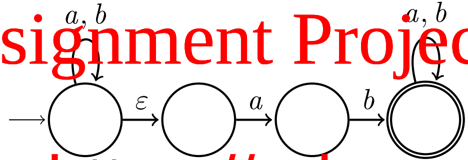
From NFA to DFA: example



Tran

	a	
$E(start) = \{q_1, q_2\}$	$\{q_1, q_2,$	
$\{q_1, q_2, q_3\}$	$\{q_1, q_2,$ 3	1 2 4
$\{q_1, q_2, q_4\}$	$\{q_1, q_2, q_3, q_4\}$	$\{q_1, q_2, q_4\}$
$\{q_1, q_2, q_3, q_4\}$		

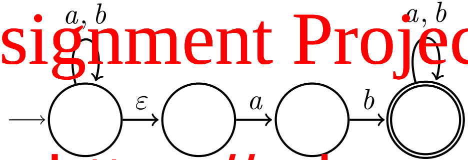
From NFA to DFA: example



Tran

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

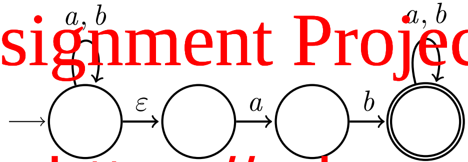
From NFA to DFA: example



Tran[illegible]

	a	[illegible]
$E(start) = \{q_1, q_2\}$	$\{q_1, q_2,$ [illegible]	[illegible]
$\{q_1, q_2, q_3\}$	$\{q_1, q_2,$ [illegible] 3	[illegible] 1 [illegible] 2 [illegible] 4 [illegible]
$\{q_1, q_2, q_4\}$	$\{q_1, q_2, q_3, q_4\}$	$\{q_1, q_2, q_4\}$
$\{q_1, q_2, q_3, q_4\}$	$\{q_1, q_2, q_3, q_4\}$	$\{q_1, q_2, q_4\}$

From NFA to DFA: example



Tran

		a	
A	$E(start) = \{q_1, q_2\}$	$\{q_1, q_2,$	
B	$\{q_1, q_2, q_3\}$	$\{q_1, q_2,$ 3	1 2 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
-
- **Minimal DFA**
- Regular Languages and Closure properti
- 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 f

from

To re

1. If s is accepting and t non acceptin equivalent

2. If, with input x, there is a transition 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 i.e satisfying either of:
   1.1 s is accepting and t is not or vice versa

   ate t'

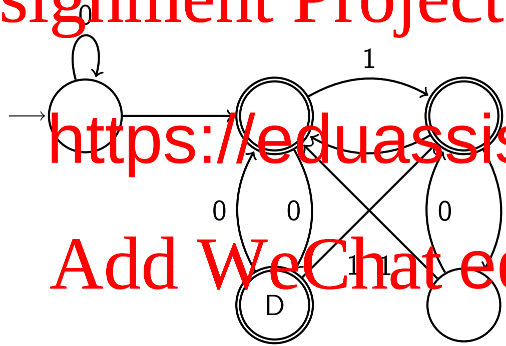
   ivalent

2.

Theorems:

1. If two states are not distinguished by the tabl 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					

1. , E are non-accepting and all the others are.

The table of state non-equivalences

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

1. , 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					

1. , E are non-accepting and all the others are.
2. Look at each remaining pair.
   - $\delta(A,0) = A$ and $\delta(E,0) = C$, bu / / E

The table of state non-equivalences

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

1. , E are non-accepting and all the others are.
2. Look at each remaining pair:
   - $\delta(A,0) = A$ and $\delta(E,0) = C$, bu / E
   - $\delta(B,0) = D$ and $\delta(C,0) = E$, b $D \not\equiv E$ $B \not\equiv C$

The table of state non-equivalences

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

1. , E are non-accepting and all the others are.
2. Look at each remaining pair:
   - $\delta(A, 0) = A$ and $\delta(E, 0) = C$, bu / E
   - $\delta(B, 0) = D$ and $\delta(C, 0) = E$, b $D \not\equiv E$ $B \not\equiv C$
   - 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					

1. , E are non-accepting and all the others are.
2. Look at each remaining pair:
   - $\delta(A,0) = A$ and $\delta(E,0) = C$, bu ... $\not\equiv E$
   - $\delta(B,0) = D$ and $\delta(C,0) = E$, b ... $D \not\equiv E$ ... $B \not\equiv C$
   - 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

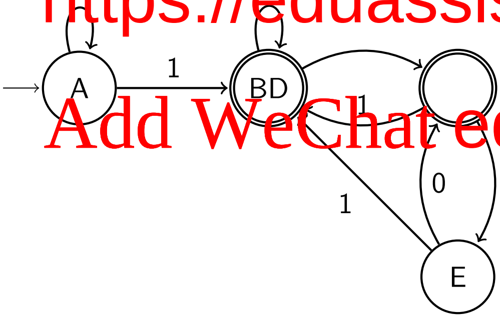
A	-	-	-	-	-
B	x	-	-	-	-
C	x	x	-	-	-
D	x		x	-	-
E					

1. , E are non-accepting and all the others are.
2. Look at each remaining pair:
   - $\delta(A, 0) = A$ and $\delta(E, 0) = C$, bu / E
   - $\delta(B, 0) = D$ and $\delta(C, 0) = E$, b $D \not\equiv E$ $B \not\equiv C$
   - 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 ex



Resulting DFA

Outline

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

Revision

Definition:

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

Mini

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

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:

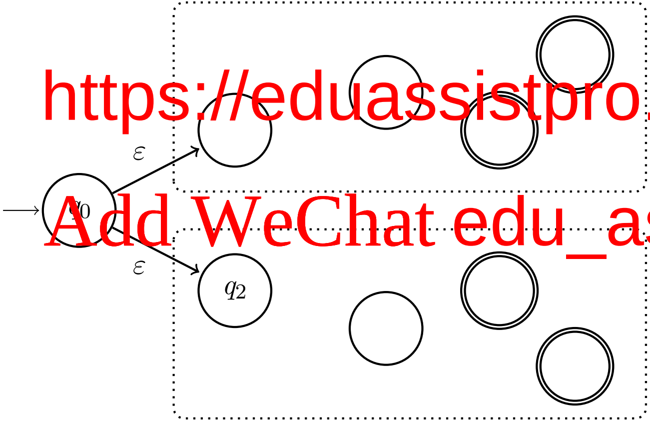
-
-
- $_1$ $_2$ $_k$ $_i$ k

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 $L_1 \cup L_2 = \{w \mid w \in L_1 \text{ or } w \in L_2\}$:



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
- $\in \Sigma_\varepsilon$

$$\delta(q, a) = \begin{cases} \delta_2(q, a) & \text{i} \\ \{q_1, q_2\} & \text{i} \\ \emptyset & \text{i} \end{cases}$$

- Accepting set $F = F_1 \cup F_2$

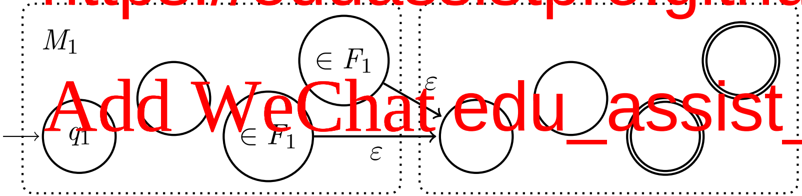
M accepts if and only if M_1 or M_2 does
i.e. $L(M) = \{w | w \in L_1 \text{ or } w \in L_2\} = L_1 \cup L_2$

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 ca

$L_1 L_2$



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:

- $Q = Q_1 \cup Q_2$
-
- $\in \Sigma_\epsilon$

$$\delta(q, a) = \begin{cases} \delta_1(q, a) & q \quad Q_1 \quad q \,/\, F_1 \\ \delta_1(q, a) & / \\ \delta_1(q, a) \cup \{q_2\} & \\ \delta_2(q, a) & q \in Q_2 \end{cases}$$

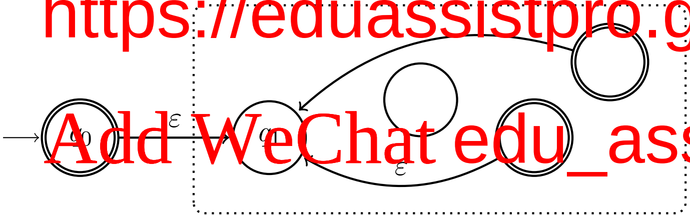
- Accepting set $F = F_2$ (same as M_2)

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

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
$L^* =$



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
- $\in \Sigma_\varepsilon$

$\notin F_1$

$\notin$

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

- 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

Cons
patt ab

Example from construction to minimal DFA

Let $\Sigma = \; 0, 1$

Let

0's

Let

Complement of a regular language

The complement of a regular language is regular

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

Exa[illegible]

Let [illegible]

Then the complement of L_1 is $L_2 = \{$ [illegible] $/$ [illegible] ns 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

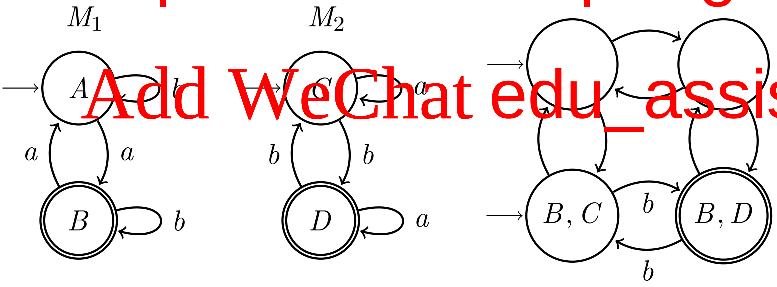
If L_1 and L_2 are regular, then $L_1 \cap L_2$ is regular.

We can use the cross product technique to build the DFA

Example:

$L_1 = \qquad\qquad\qquad\qquad \}$

$L_2 = \qquad\qquad\qquad\qquad \}$



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:

- $\in Q_1$
- Q_1 and 2

- The transition function δ is defined a

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

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

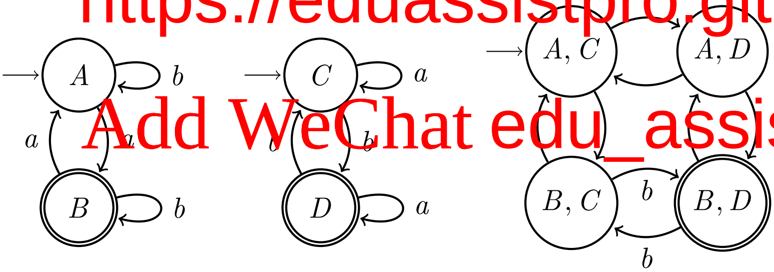
Note that M is also deterministic

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$\}$

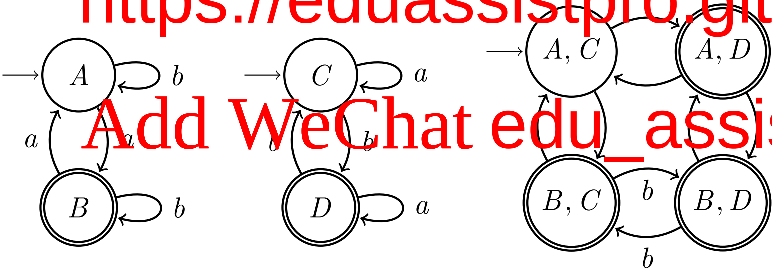


Cross-product can be used for Union

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

-

- Minimal DFA

- Regular Languages and Closure properti

- **Regular Expressions (introduction)**

Regular Expressions

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

With regular languages, we use regular operatio
expressions describing languages such as

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

$(ab)^\star a$ (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
-

    - 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 $L(\varepsilon) = \ \varepsilon$

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 $L(\varepsilon) = \ \varepsilon$
-

  $R \mid S$ $\qquad$ $R + S$
  - Concatenation: RS (sometimes d
  - 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 $L(\varepsilon) = \ \varepsilon$

- $R \mid S$ $R + S$
  - Concatenation: RS (sometimes d
  - Star Closure: $R^\star$ is a RegEx

- Precedence of operators: $^\star, \circ, \mid$ to a
  - 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

$$L(\emptyset) = \{\emptyset\}$$

$$L(x \mid y) = \{x, y\}$$

$$L(x^*) = \{\epsilon, x, xx, \ldots\}$$

$$L(xy) = \{xy\}$$

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

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

Operators on RegEx: Union $R \mid S$

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

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

$a \mid b)$

$L($

Operators on RegEx: Union $R \mid S$

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

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

$a \mid b)$

$L($

Operators on RegEx: Union $R \mid S$

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

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

$a \mid b)$

$L($

$=$

Operators on RegEx: Union $R \mid S$

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

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

$a \mid b)$

$L($

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

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

Operators on RegEx: Union $R \mid S$

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

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

$a \mid b)$

$L($

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

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

$$=$$

Operators on RegEx: Union $R \mid S$

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

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

$a \mid b)$

$L($

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

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

$$= \{a\} \cup \{b, c\}$$

$$=$$

Operators on RegEx: Union $R \mid S$

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

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

$a \mid b)$

$L($

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

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

$$= \{a\} \cup \{b, c\}$$
$$= \{a, b, c\}$$

Operators on RegEx: Union $R \mid S$

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

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

$a \mid b)$

$L($

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

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

$$= \{a\} \cup \{b, c\}$$

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

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

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

Operators on RegEx: Concatenation RS

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

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

Operators on RegEx: Concatenation RS

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

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

Operators on RegEx: Concatenation RS

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

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

$=$

Operators on RegEx: Concatenation RS

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

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

$= \{abc\}$

$L(a(bc)) =$

Operators on RegEx: Concatenation RS

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

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

$$= \{abc\}$$

$L(a(bc)) = \{rs \mid r \in L(a)$ an

$=$

Operators on RegEx: Concatenation RS

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

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

$$= \{abc\}$$

$$L(a(bc)) = \{rs \mid r \in L(a) \text{ an}$$

$$= \{rs \mid r \in \{a\} \text{ and}$$

$$=$$

Operators on RegEx: Concatenation RS

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

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

$= \{abc\}$

$L(a(bc)) = \{rs \mid r \in L(a)$ an

$= \{rs \mid r \in \{a\}$ and

$= \{abc\}$

Operators on RegEx: Concatenation RS

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

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

$$= \{abc\}$$

$L(a(bc)) = \{rs \mid r \in L(a)$ an

$= \{rs \mid r \in \{a\}$ and

$$= \{abc\}$$

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

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

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

Union and Concatenation

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

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

Union and Concatenation

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

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

Union and Concatenation

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

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

Union and Concatenation

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

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

Union and Concatenation

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

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

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

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

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

$L(\emptyset^\star) =$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$L(a \mid (bc)^\star) = \{a, \varepsilon, bc, bcbc,$$

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

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

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

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

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

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

$$L(a \mid (bc)^\star) = \{a, \varepsilon, bc, bcbc,$$

$$L((a \mid b)c^\star) = \{a, b, ac, bc, acc, bcc, accc, bccc, \dots\}$$

Some properties of RegEx

Let R, S, T be regular expressions

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

- 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)

- 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)

- 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

-

SOME PROPERTIES OF REGEX

Let R, S, T be regular expressions

-

Some properties of RegEx

Let R, S, T be regular expressions

-
  - $R^\star = \varepsilon \mid R^\star = (\varepsilon \mid R)^\star = (\varepsilon \mid \quad {}^\star \quad {}^\star$

Some properties of RegEx

Let R, S, T be regular expressions

-

  - $R^\star = \varepsilon \mid R^\star = (\varepsilon \mid R)^\star = (\varepsilon \mid$ $\star$ $\star$
  - $RR^\star = R^\star R$

Some properties of RegEx

Let R, S, T be regular expressions

-

  - $R^\star = \varepsilon \mid R^\star = (\varepsilon \mid R)^\star = (\varepsilon \mid \quad ^\star \quad ^\star$
  - $RR^\star = R^\star R$
  - $(R \mid S)^\star = (R^\star \mid S^\star)^\star = (R^\star S^\star)^\star$

Some properties of RegEx

Let R, S, T be regular expressions

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

Some properties of RegEx

Let R, S, T be regular expressions

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

Some properties of RegEx

Let R, S, T be regular expressions

-

  - $R^\star = \varepsilon \mid R^\star = (\varepsilon \mid R)^\star = (\varepsilon \mid \quad {}^\star \quad {}^\star$
  - $RR^\star = R^\star R$
  - $(R \mid S)^\star = (R^\star \mid S^\star)^\star = (R^\star S^\star \quad)^\star$
  - $R(SR)^\star = (RS)^\star R$
  - $(R^\star S)^\star = \varepsilon \mid (R \mid S)^\star S$
  - $(RS^\star)^\star = \varepsilon \mid R(R \mid S)^\star$

Examples of RegEx

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

- Identifiers for Java programs?

Examples of RegEx

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

-

- Traffic lights?

- Identifiers for Java programs?

Examples of RegEx

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

- Identifiers for Java programs?

Examples of RegEx

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

- Traffic lights?
  $(red \mid red \circ amber \mid amber \mid gree$

- Identifiers for Java programs?

Examples of RegEx

- All possible strings which can be formed with a, b, or c
  $(a \mid b \mid c)^{\star}$
-
- Traffic lights?
  $(red \mid red \circ amber \mid amber \mid gree$

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

Examples of RegEx

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

-

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

- $a^\star \emptyset$ represents

Examples of RegEx

- $a^\star ba^\star$ represents

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

-

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

- $a^\star\emptyset$ represents

Examples of RegEx

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

-

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

- $a^\star \emptyset$ represents

Examples of RegEx

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

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

- $a^\star\emptyset$ represents

Examples of RegEx

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

-

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

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

Equivalence of RegEx and FA

Theorem:

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

Proo

Sho

1.

   recognises the same language

2. FA ⟹ RegEx:

   Show that for each NFA, there exists a RegEx

   recognises the same language

Equivalence of RegEx and FA

Theorem:

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

Proo

Sho

1.

   recognises the same language

2. FA $\Rightarrow$ RegEx:
   Show that for each NFA, there exists a RegEx
   recognises the same language

Next week: proof

Application: pattern matching

- Provides a technique for finding occurrences of patterns in
-
- pattern is trivial.
- Then we convert the NFA to DFA, then mini 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
-

-
  - despite the fact they often have far more st
- Regular languages are closed under the uni star closure, intersection, and compleme

  - We use these to build the NFA, before conv DFA

Regular Expressions

-
-
- Equivalence with DFA/NFA