

# Inf2A: Converting from NFAs to DFAs and Closure Properties

Stuart Anderson

School of Informatics  
University of Edinburgh

October 13, 2009

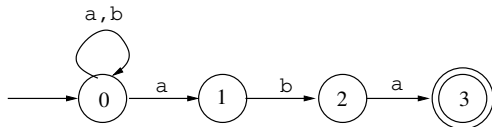
## Starter Questions

- 1 Can you devise a way of testing for any FSM  $M$  whether  $L(M) = \emptyset$ ?
- 2 Can you devise a way of testing for any pair of FSMs  $M$  and  $M'$  whether  $L(M) = L(M')$ ?

# Outline

- 1 Non-Deterministic Finite Automata
  - NFA - the definitions
  - NFA: Example
- 2 The Conversion Theorem
  - Converting from NFAs to DFAs
  - Example: Converting from an NFA to a DFA
- 3 Closure Properties
  - Demonstrating Closure under various operators
  - Closure: Example
- 4 Summary

# A non-deterministic automaton



Formally:

$$N = (\{0, 1, 2, 3\}, \{a, b\}, 0, \{3\}, \Delta),$$

where the transition relation  $\Delta$  is given by the following table:

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

# Non-deterministic Finite Automata

**Definition:** A *nondeterministic finite automaton* (or *NFA*) is a tuple

$$N = (Q, \Sigma, q_0, F, \Delta)$$

consisting of:

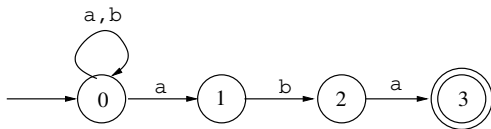
- ① a finite set  $Q$  of *states*,
- ② a finite *alphabet*  $\Sigma$ ,
- ③ a distinguished *starting state*  $q_0 \in Q$ ,
- ④ a set  $F \subseteq Q$  of *final states* (the ones that indicate acceptance),
- ⑤ a description  $\Delta$  of all the possible *transitions*.  
 $\Delta$  must be given by a table that answers the following question: “Given a state  $q$  and an input symbol  $a$ , what are the possible next states?”  
 $\Delta$  is called the *transition function* or *transition relation*.

## The Language of an NFA

- As opposed to a DFA, an NFA may have *several possible computations* for the same input string — or it may get stuck and have *no* (complete) computation at all.
- An NFA  $N$  is said to *accept* a string  $x$  if there is *at least one* computation of  $N$  on input  $x$  that leads to an accepting final state.
- The language *recognized* by an NFA  $N = (Q, \Sigma, q_0, F, \Delta)$  is the language

$$L(N) = \{x \in \Sigma^* \mid N \text{ accepts } x\}.$$

# Example



Two possible computations of this NFA on input *aaaba*:

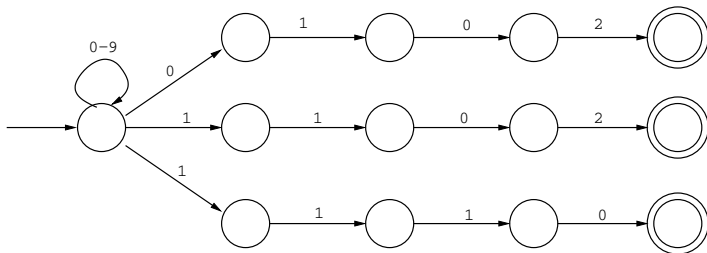
$$0 \xrightarrow{a} 0 \xrightarrow{a} 0 \xrightarrow{a} 0 \xrightarrow{b} 0 \xrightarrow{a} 0$$

$$0 \xrightarrow{a} 0 \xrightarrow{a} 0 \xrightarrow{a} 1 \xrightarrow{b} 2 \xrightarrow{a} 3$$

The NFA accepts the language

$$\{xaba \mid x \in \{a, b\}^*\}.$$

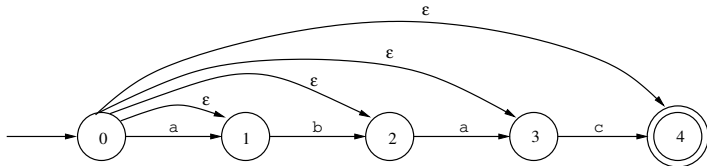
# Example



## NFAs with $\epsilon$ -Transitions

**$\epsilon$ -transitions** are transitions not triggered by any input symbol. If in a state  $q$  an  $\epsilon$ -transition to a state  $q'$  is possible, then whenever the automaton reaches state  $q$  in a computation it can immediately proceed to state  $q'$  without needing any further input. To describe this formally the function  $\Delta$  is defined for  $\Sigma \cup \{\epsilon\}$ .

**Example:**



An NFA with  $\epsilon$ -transitions recognizing the language

$\{\epsilon, c, ac, bac, abac\}$ .

## The languages of NFAs

**Obvious fact 1:** Any language recognized by some DFA can also be recognized by some NFA.

**Obvious fact 2:** Any language recognized by some NFA can also be recognized by some NFA with  $\epsilon$ -transitions.

# The Conversion Theorem

**Theorem:** Any language accepted by some NFA with  $\epsilon$ -transitions is also accepted by some DFA.

Therefore, every language accepted by an NFA, with or without  $\epsilon$ -transitions, is regular.

## Proof: The Powerset Construction

Let  $N = (Q, \Sigma, q_0, F, \Delta)$  be an NFA.

**Idea:** Simulate all possible computations of  $N$  “in parallel”.

**Construction:** Let  $M = (S, \Sigma, s_0, F', \delta)$ , where

- $S = \mathcal{P}(Q) = \{s \mid s \subseteq Q\}$  (the *powerset* of  $Q$ )
- $s_0 = \{q_0\} \cup \{q \mid q \text{ can be reached from } q_0 \text{ using only } \varepsilon\text{-transitions}\}$
- $\delta$  is defined by

$\delta(s, a) =$  set of all states reachable from a state in  $s$  by one  $a$ -transition and  $\varepsilon$ -transitions.

- $F' = \{s \in S \mid s \cap F \neq \emptyset\}$ .

# The conversion algorithm

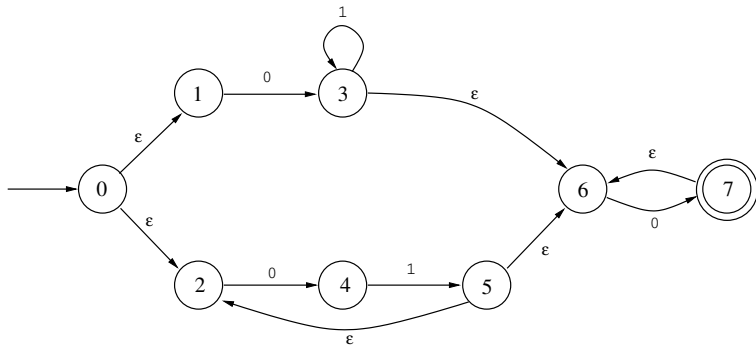
- 1 Construct the **start state**  $s_0$  consisting of  $q_0$  and all states of  $N$  that can be reached from  $q_0$  by one or several  $\varepsilon$ -transitions. Mark  $s_0$  as “unfinished”.
- 2 While there still are “unfinished” states:
  - 1 Take an “unfinished” state  $s$ .
  - 2 For each  $a \in \Sigma$ , let
 
$$\delta(s, a) = \text{set of all states reachable from a state in } s$$

$$\text{by one } a\text{-transition and } \varepsilon\text{-transitions}$$

$$= \bigcup_{q \in s} \text{set of all states reachable from } q$$

$$\text{by one } a\text{-transition and } \varepsilon\text{-transitions}$$
 If  $\delta(s, a)$  is neither “finished” nor “unfinished” yet, then mark  $\delta(s, a)$  as “unfinished”.
  - 3 Mark  $s$  as “finished”.
- 3 Mark all states that contain a final state from  $N$  as **final states of  $M$** .

## An example conversion



**The start state:**

$$s_0 = \{0, 1, 2\}.$$

# Computing the transition table

$\delta$	0	1
$\{0, 1, 2\}$		

$\delta$	0	1
$\{0, 1, 2\}$	$\{3, 4, 6\}$	$\emptyset$
$\{3, 4, 6\}$		
$\emptyset$		

$\delta$	0	1	$\delta$	0	1
$\{0, 1, 2\}$	$\{3, 4, 6\}$	$\emptyset$	$\{0, 1, 2\}$	$\{3, 4, 6\}$	$\emptyset$
$\{3, 4, 6\}$	$\{6, 7\}$	$\{2, 3, 5, 6\}$	$\{3, 4, 6\}$	$\{6, 7\}$	$\{2, 3, 5, 6\}$
$\emptyset$			$\emptyset$	$\emptyset$	$\emptyset$
$\{6, 7\}$			$\{6, 7\}$		
$\{2, 3, 5, 6\}$			$\{2, 3, 5, 6\}$		

$\delta$	0	1
{0, 1, 2}	{3, 4, 6}	$\emptyset$
{3, 4, 6}	{6, 7}	{2, 3, 5, 6}
$\emptyset$	$\emptyset$	$\emptyset$
{6, 7}	{6, 7}	$\emptyset$
{2, 3, 5, 6}		

$\delta$	0	1
{0, 1, 2}	{3, 4, 6}	$\emptyset$
{3, 4, 6}	{6, 7}	{2, 3, 5, 6}
$\emptyset$	$\emptyset$	$\emptyset$
{6, 7}	{6, 7}	$\emptyset$
{2, 3, 5, 6}	{4, 6, 7}	{3, 6}
{4, 6, 7}		
{3, 6}		

$\delta$	0	1
$\{0, 1, 2\}$	$\{3, 4, 6\}$	$\emptyset$
$\{3, 4, 6\}$	$\{6, 7\}$	$\{2, 3, 5, 6\}$
$\emptyset$	$\emptyset$	$\emptyset$
$\{6, 7\}$	$\{6, 7\}$	$\emptyset$
$\{2, 3, 5, 6\}$	$\{4, 6, 7\}$	$\{3, 6\}$
$\{4, 6, 7\}$	$\{6, 7\}$	$\{2, 5, 6\}$
$\{3, 6\}$		
$\{2, 5, 6\}$		

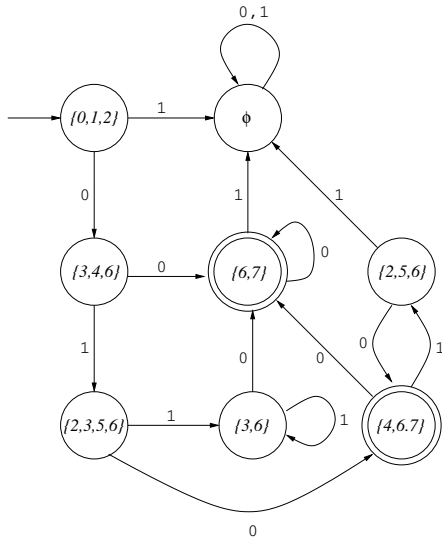
$\delta$	0	1
$\{0, 1, 2\}$	$\{3, 4, 6\}$	$\emptyset$
$\{3, 4, 6\}$	$\{6, 7\}$	$\{2, 3, 5, 6\}$
$\emptyset$	$\emptyset$	$\emptyset$
$\{6, 7\}$	$\{6, 7\}$	$\emptyset$
$\{2, 3, 5, 6\}$	$\{4, 6, 7\}$	$\{3, 6\}$
$\{4, 6, 7\}$	$\{6, 7\}$	$\{2, 5, 6\}$
$\{3, 6\}$	$\{6, 7\}$	$\{3, 6\}$
$\{2, 5, 6\}$		

$\delta$	0	1
$\{0, 1, 2\}$	$\{3, 4, 6\}$	$\emptyset$
$\{3, 4, 6\}$	$\{6, 7\}$	$\{2, 3, 5, 6\}$
$\emptyset$	$\emptyset$	$\emptyset$
$\{6, 7\}$	$\{6, 7\}$	$\emptyset$
$\{2, 3, 5, 6\}$	$\{4, 6, 7\}$	$\{3, 6\}$
$\{4, 6, 7\}$	$\{6, 7\}$	$\{2, 5, 6\}$
$\{3, 6\}$	$\{6, 7\}$	$\{3, 6\}$
$\{2, 5, 6\}$	$\{4, 6, 7\}$	$\emptyset$

**The final states:**

$$\begin{aligned}
 F' &= \{s \in S \mid s \cap \{7\} \neq \emptyset\} \\
 &= \{\{6, 7\}, \{4, 6, 7\}\}.
 \end{aligned}$$

# The resulting DFA



## What do we need to show about the Conversion Algorithm?

- If  $M$  is an NFA and  $M'$  is the DFA derived from  $M$  via the conversion algorithm, what should we be able to prove about  $M$  and  $M'$
- What technique do you think we need for the proof?

# Outline

- 1 Non-Deterministic Finite Automata
  - NFA - the definitions
  - NFA: Example
- 2 The Conversion Theorem
  - Converting from NFAs to DFAs
  - Example: Converting from an NFA to a DFA
- 3 Closure Properties
  - Demonstrating Closure under various operators
  - Closure: Example
- 4 Summary

## Recall Regular expressions

$R$	$L(R)$
$\varepsilon$	$\{\varepsilon\}$
$\emptyset$	$\emptyset$
$a$	$\{a\}$ (for any $a \in \Sigma$ )
$R_1 + R_2$	$L(R_1) \cup L(R_2)$
$R_1 R_2$	$\{xy \mid x \in L(R_1), y \in L(R_2)\}$
$R_1^*$	$\{x_1 \dots x_n \mid n \in \mathbb{N}_0, x_1, \dots, x_n \in L(R_1)\}$ .

We know we can construct an NFA for any Regular Expression.

# Closure under union and concatenation

**Theorem:** If  $L_1$  and  $L_2$  are regular languages, then the following two languages are also regular:

- 1  $L_1 \cup L_2$ ,
- 2  $L_1 L_2 = \{xy \mid x \in L_1, y \in L_2\}$ .

**Proof:** Let  $R_1, R_2$  be regular expressions such that  $L(R_1) = L_1, L(R_2) = L_2$ . Then

$$L_1 \cup L_2 = L(R_1) \cup L(R_2) = L(R_1 + R_2)$$

and

$$L_1 L_2 = L(R_1)L(R_2) = L(R_1 R_2).$$

## Closure under complementation

**Theorem:** If  $L \subseteq \Sigma^*$  is a regular language, then the following language is also regular:

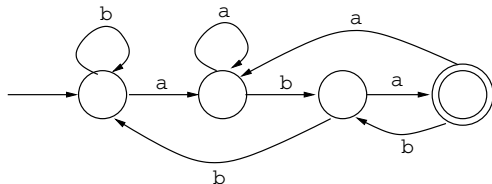
$$\sim L = \{x \in \Sigma^* \mid x \notin L\}.$$

**Proof:** Let  $M$  be a DFA that recognises  $L$ . Let  $M'$  be the DFA obtained from  $M$  by making all states that are not final states of  $M$  final states of  $M'$  and vice versa. Then  $M'$  recognises  $\sim L$ :

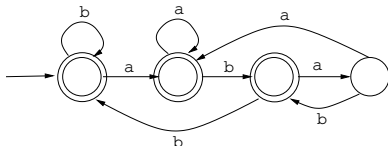
$M'$  accepts string  $x$

- $\iff$  running  $M'$  on input  $x$  ends in a final state of  $M'$
- $\iff$  running  $M'$  on input  $x$  does not end in a final state of  $M$
- $\iff$  running  $M$  on input  $x$  does not end in a final state of  $M$
- $\iff M$  does not accept  $x \iff x \in \sim L$ .

# Example



A DFA for the language  
 $L = \{xaba \mid x \in \{a,b\}^*\}$ .



A DFA for  $\sim L$ .

# Closure under intersection and difference

**Theorem:** If  $L_1$  and  $L_2$  are regular languages, then the following two languages are also regular:

- 1  $L_1 \cap L_2$ ,
- 2  $L_1 \setminus L_2 = \{x \mid x \in L_1 \text{ and } x \notin L_2\}$ .

**Proof:** We use the facts that

$$L_1 \cap L_2 = \sim (\sim L_1 \cup \sim L_2) \quad (\text{by DeMorgan's rule})$$

and

$$L_1 \setminus L_2 = L_1 \cap \sim L_2 = \sim (\sim L_1 \cup L_2).$$

# Constructions

*Problem:* Suppose we are given DFAs for  $L_1$  and  $L_2$ .  
How do we construct DFAs for

$$\sim L_1, \quad L_1 \cup L_2, \quad L_1 \cap L_2, \quad L_1 \setminus L_2 \quad ?$$

# Complementation

We have already seen a construction.

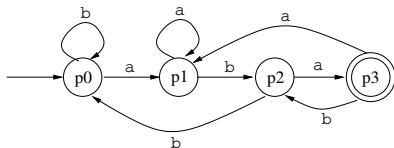
## Union

- 1 Compute an NFA with  $\varepsilon$ -transitions.
- 2 Convert it to a DFA.

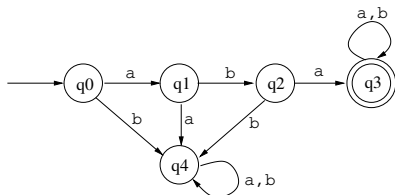
## Intersection and difference

Combine the constructions for complementation and union.

# Example



$M_1$



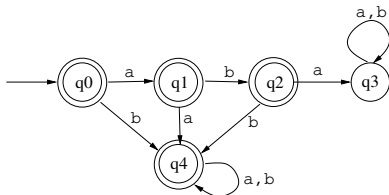
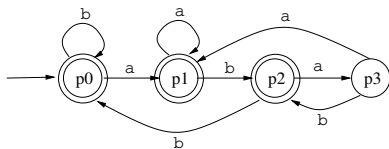
$M_2$

Construct a DFA for

$$L(M_1) \cap L(M_2) = \sim (\sim L(M_1) \cup \sim L(M_2)).$$

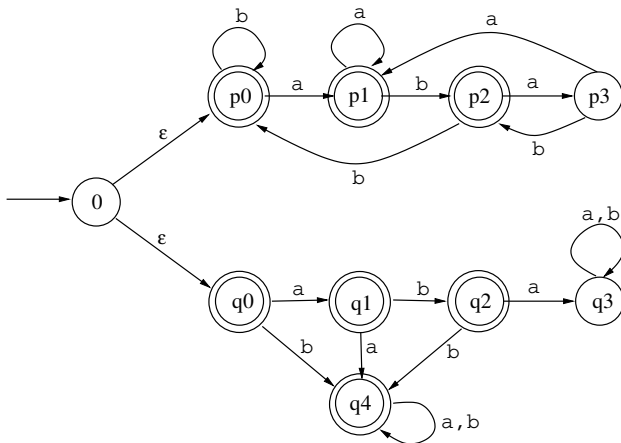
## Example (cont'd)

Step 1: Construct DFAs for  $\sim L(M_1)$  and  $\sim L(M_2)$ .



## Example (cont'd)

Step 2: Construct an NFA with  $\varepsilon$ -transitions for  
 $\sim L(M_1) \cup \sim L(M_2)$ .



## Example (cont'd)

Step 3: Convert the NFA with  $\varepsilon$ -transitions for  
 $\sim L(M_1) \cup \sim L(M_2)$  to a **DFA**.

$\delta$	a	b
$\{0, p0, q0\}$	$\{p1, q1\}$	$\{p0, q4\}$
$\{p1, q1\}$	$\{p1, q4\}$	$\{p2, q2\}$
$\{p0, q4\}$	$\{p1, q4\}$	$\{p0, q4\}$
$\{p1, q4\}$	$\{p1, q4\}$	$\{p2, q4\}$
$\{p2, q2\}$	$\{p3, q3\}$	$\{p0, q4\}$
$\{p2, q4\}$	$\{p3, q4\}$	$\{p0, q4\}$
$\{p3, q3\}$	$\{p1, q3\}$	$\{p2, q3\}$
$\{p3, q4\}$	$\{p1, q4\}$	$\{p2, q4\}$
$\{p1, q3\}$	$\{p1, q3\}$	$\{p2, q3\}$
$\{p2, q3\}$	$\{p3, q3\}$	$\{p0, q3\}$
$\{p0, q3\}$	$\{p1, q3\}$	$\{p0, q3\}$

## Example (cont'd)

Step 4: **Complement** the DFA for  $\sim L(M_1) \cup \sim L(M_2)$ .

Result:

$$\left( Q, \Sigma, \{0, p0, q0\}, \{\{p3, q3\}\}, \delta \right),$$

where

$$Q = \{\{0, p0, q0\}, \{p1, q1\}, \{p0, q4\}, \{p1, q4\}, \{p2, q2\}, \{p2, q4\}, \\ \{p3, q3\}, \{p3, q4\}, \{p1, q3\}, \{p2, q3\}, \{p0, q3\}\},$$

## Union vs. Intersection vs. Difference

The automata we construct for  $L(M_1) \cap L(M_2)$ ,  $L(M_1) \cup L(M_2)$ , and  $L(M_1) \setminus L(M_2)$  only differ in their final states:

**Intersection:** Final states (of the powerset automaton) are states that contain a final state of  $M_1$  *and* a final state of  $M_2$ .

**Union:** Final states (of the powerset automaton) are states that contain a final state of  $M_1$  *or* a final state of  $M_2$ .

**Difference:** Final states (of the powerset automaton) are states that contain a final state of  $M_1$  *but not* a final state of  $M_2$ .

# Summary

- Kleene's theorem tell us that regular expressions and FSAs describe the same class of languages.
- NFA/DFA equivalence tells us that in this case nondeterminism does not change the class of languages recognised (see later).
- Closure results tell us we can augment regular expressions with many operators that allow us to express regular sets succinctly without going outside languages recognisable by FSMs.