2.

FINITE AUTOMATA

Overview of today’s lecture

- Review – from last lecture
- Deterministic finite automata – DFA
- Non-deterministic finite automata – NFA
- From DFA to NFA
- By the way …
- Reading material and exercises

REVIEW – FROM LAST LECTURE

Alphabet, strings and languages

- An alphabet Σ is a set of symbols (or letters)
- A string $w = a_1\, a_2 \ldots a_k$ is a sequence of symbols from Σ
- The empty string is written ε
- A language L is a set of strings over Σ, that is, a subset of Σ^*
  - The empty language: $\emptyset$
  - The language of the empty string: $\{\varepsilon\}$

Regular languages

- How is the language specified?
  - Deterministic finite automaton
  - Non-deterministic finite automaton
  - Regular expression

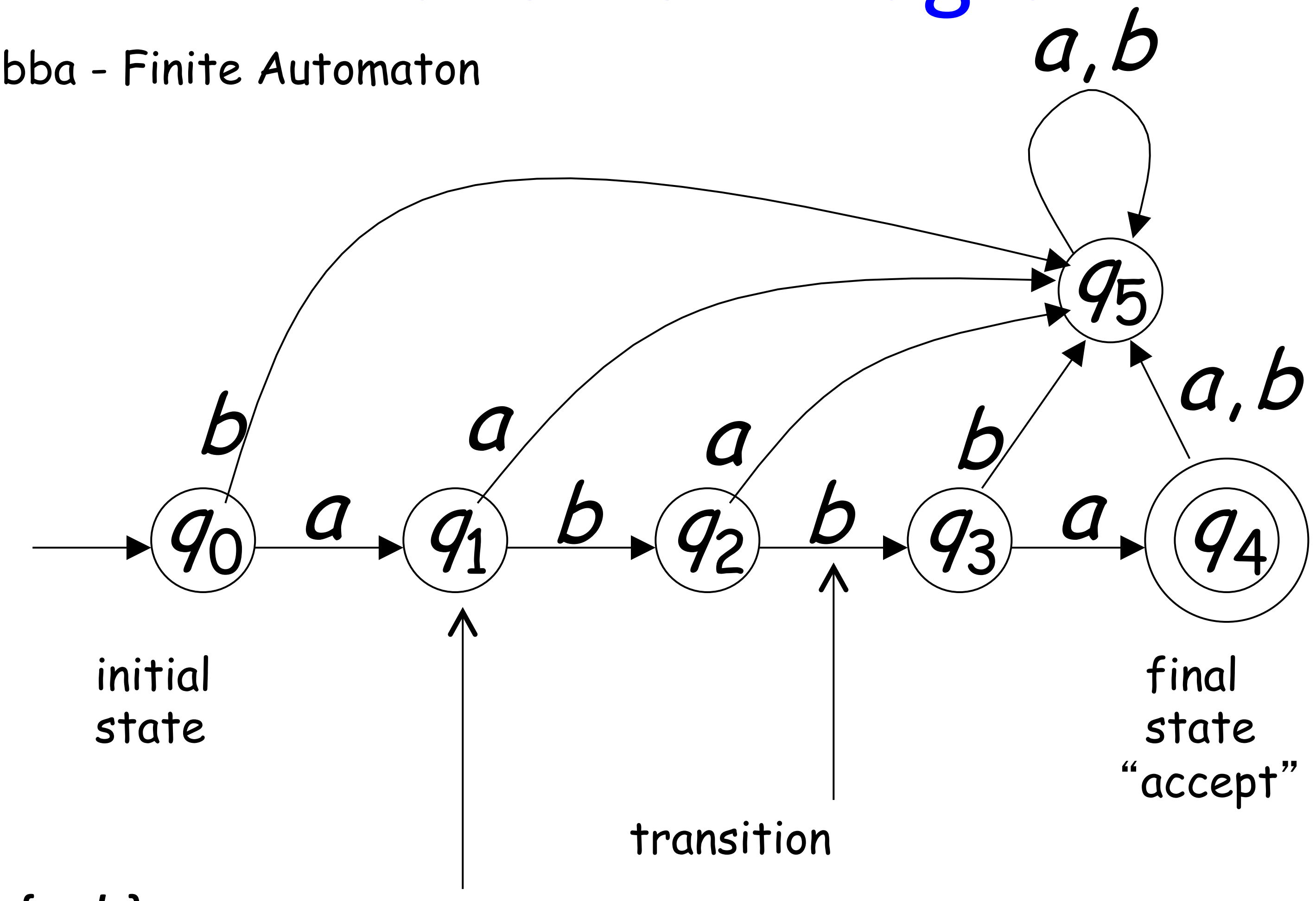
  The three methods are equivalent!

- Three fundamental problems:
  - The membership problem
  - The emptiness problem
  - The equivalence problem

DETERMINISTIC FINITE AUTOMATA – DFA

Transition Diagram

abba - Finite Automaton



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

Formal definition

A *deterministic finite automaton* consists of

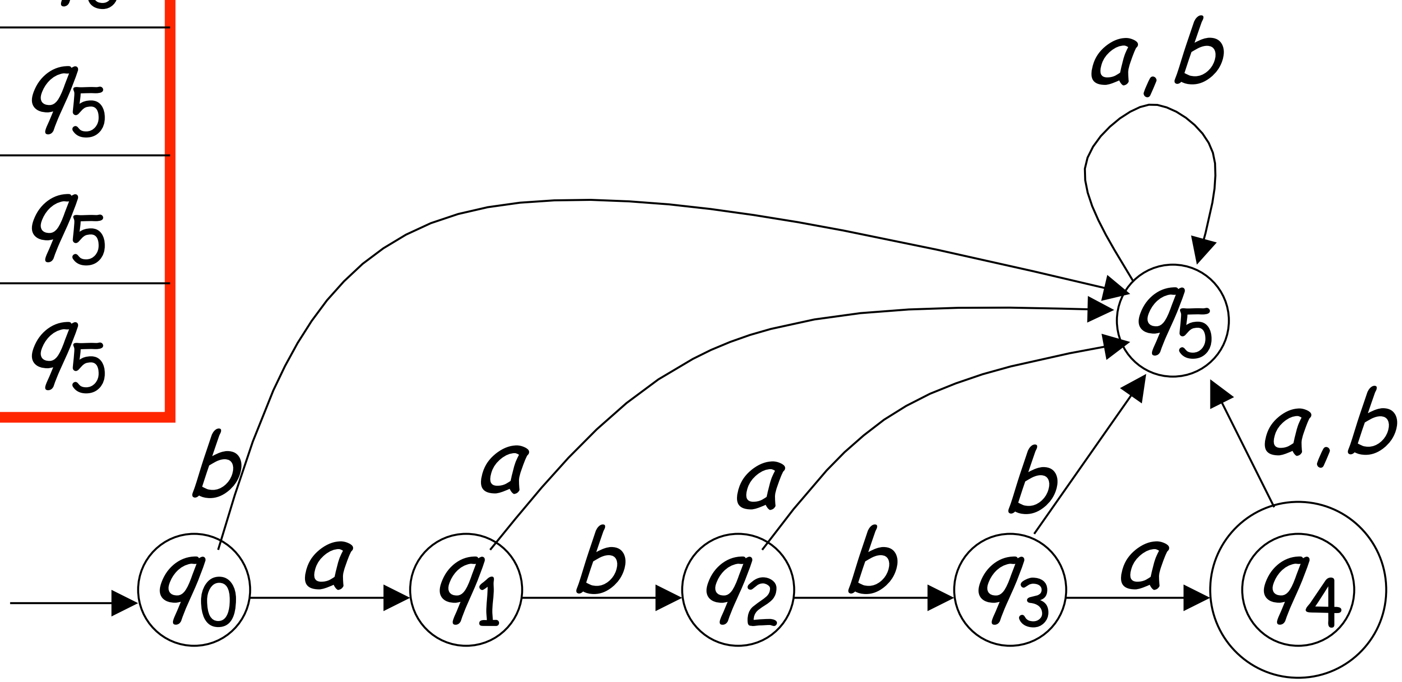
- Q: a finite set of states
- Σ: a finite set of input symbols, an alphabet
- $\delta: Q \times \Sigma \rightarrow Q$: a transition function
  for each state q in Q and each symbol a in Σ it determines a new state $\delta(q,a)$ in Q
- q_0: the initial state; an element of Q
- F: the final states; a subset of Q

Often written as $A = (Q, \Sigma, \delta, q_0, F)$

Transition Function δ

δ	a	b
q_0	q_1	q_5
q_1	q_5	q_2
q_2	q_5	q_3
q_3	q_4	q_5
q_4	q_5	q_5
q_5	q_5	q_5

$$\boxed{\delta: Q \times \Sigma \rightarrow Q}$$



Let's do it!

Exercise 2.2.4 (b)

- Consider the alphabet {0,1} and specify a DFA that accepts the set of all strings with three consecutive 0's (not necessarily at the end).

- (We will do the rest of exercise 2.2.4 later)

How does a DFA define a language?

Consider a string w

- Start in the initial state q_0
- Read the first symbol, say a_1, of w
- Determine the new state $q_1 = \delta(q_0,a_1)$
- Read the second symbol, say a_2, of w
- Determine the new state $q_2 = \delta(q_1,a_2)$
- …
- Let q_k be the state obtained after having read the last symbol of w
- If q_k is in F then accept w; otherwise reject w

Formalisation

By induction on the length of the string w define:

- $\delta^*(q,\varepsilon) = q$
- $\delta^*(q,aw) = \delta^*(q',w)$ where $q' = \delta(q,a)$

  or alternatively $\delta^*(q,aw) = \delta^*(\delta(q,a),w)$

The language of $A = (Q, \Sigma, \delta, q_0, F)$

$L(A) = \{w \mid \delta^*(q_0,w) \text{ is in } F\}$

We say that *A accepts w* if w is in L(A)

Formalisation

$\hat{\delta}$ is written $\underline{\delta}$ on these slides

The definition in the book is different:

By induction on the length of the string w:

- $\underline{\delta}(q,\varepsilon) = q$
- $\underline{\delta}(q,wa) = \delta(\underline{\delta}(q,w),a)$

What is the difference?

Exercise 2.2.2 and 2.2.3 show that it does not matter:

$\delta^*(q,\varepsilon) = q$
$\delta^*(q,aw) = \delta^*(\delta(q,a),w)$

For all q and w: $\delta^*(q,w) = \underline{\delta}(q,w)$

Proof by induction

- Suppose we want to prove that the property $S(n)$ holds for all integers $n \geq 0$.
- Then we proceed as follows:
  - **Base case**: We prove that $S(0)$ holds.
  - **Induction step**: Here we assume that $S(n)$ holds and we prove that $S(n+1)$ holds.
- These two parts are sufficient to show that $S(n)$ holds for all $n \geq 0$
- Why?

Let's do it!

Exercises 2.2.2 and 2.2.3

$\hat{\delta}$ is written $\underline{\delta}$ on these slides

Exercise 2.2.2: Prove that for all q, x and y:

$$\underline{\delta}(q,xy) = \underline{\delta}(\underline{\delta}(q,x),y)$$

Definition from the book:
$\underline{\delta}(q,\varepsilon) = q$
$\underline{\delta}(q,wa) = \delta(\underline{\delta}(q,w),a)$

Exercise 2.2.3: Prove that for all q, a and x:

$$\underline{\delta}(q,ax)=\underline{\delta}(\delta(q,a),x)$$

Next: Conclude that for all q and x:

$$\delta^*(q,x) = \underline{\delta}(q,x)$$

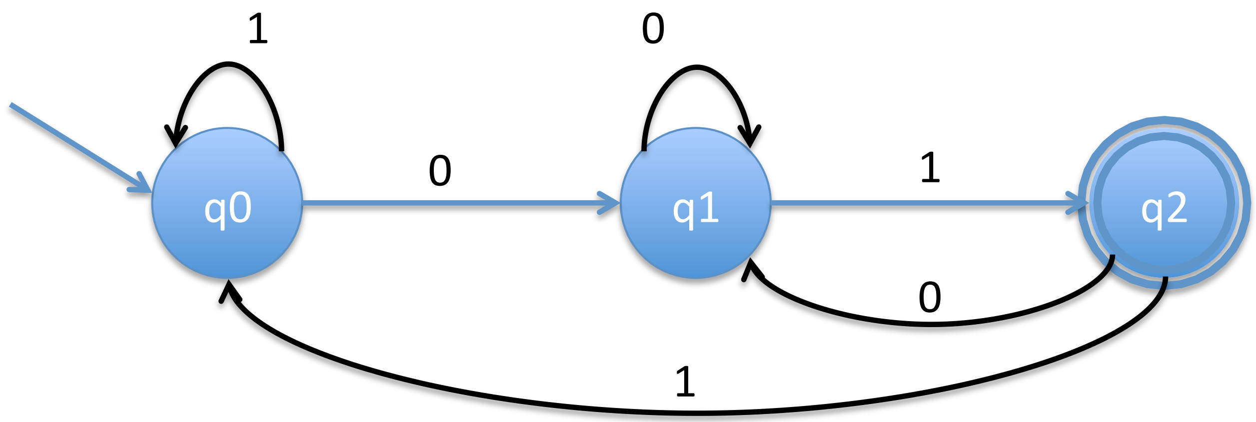
Definition from the slides:
$\delta^*(q,\varepsilon) = q$
$\delta^*(q,aw) = \delta^*(\delta(q,a),w)$

NON-DETERMINISTIC FINITE AUTOMATA – NFA

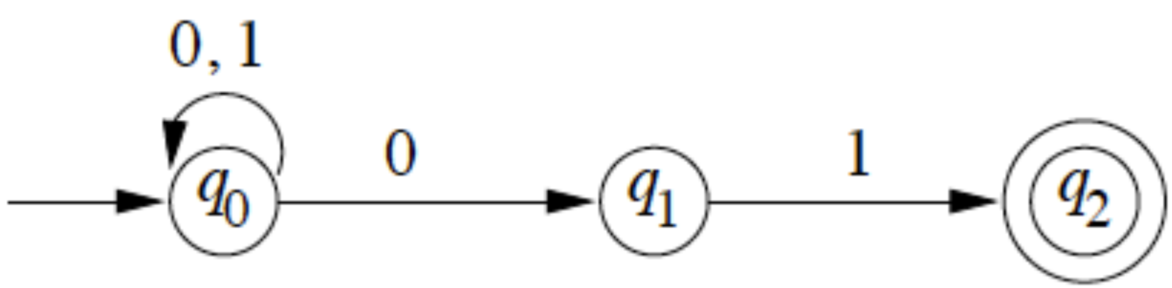
Why non-determinism?

- Language of all strings over $\Sigma=\{0,1\}$ ending in 01

DFA

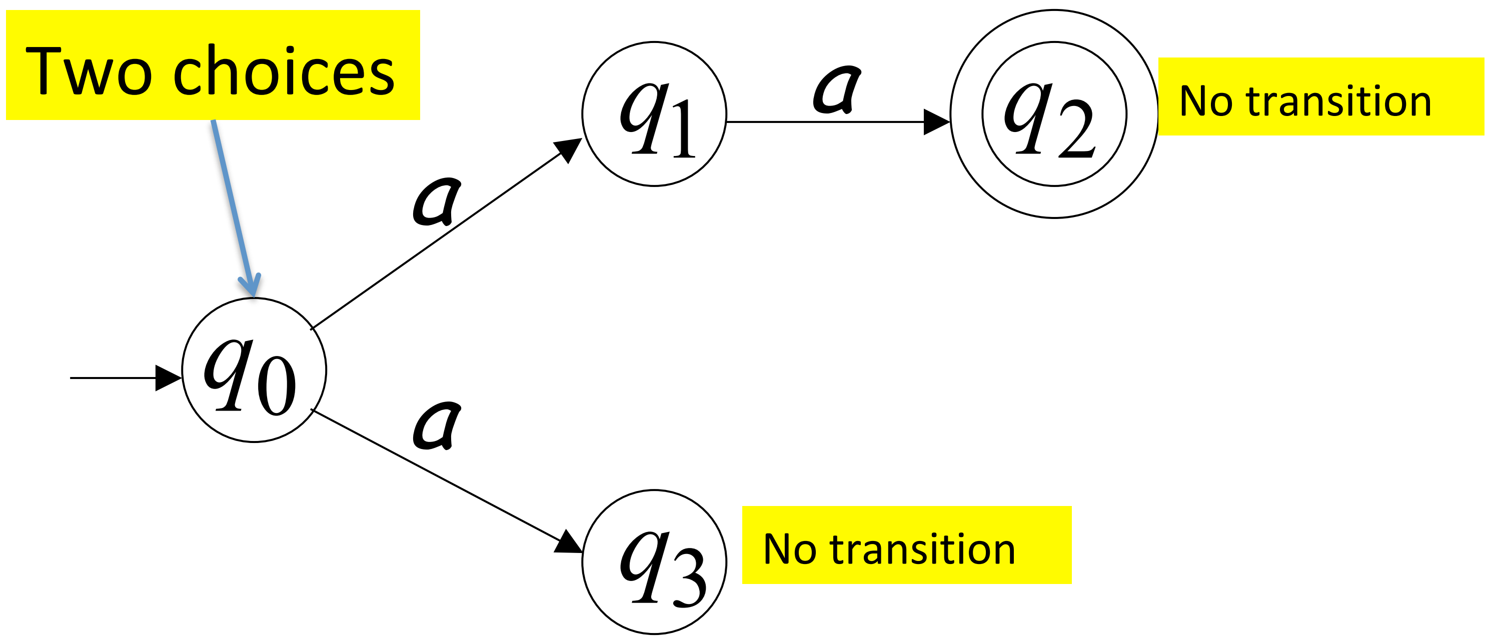


NFA



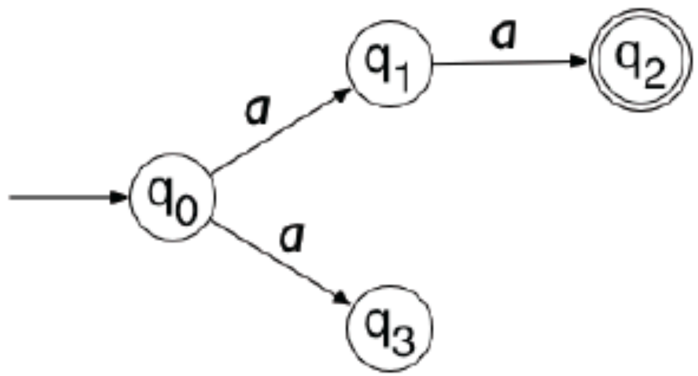
Non-deterministic Finite Automata (NFA)

Alphabet = $\{a\}$



Examples

Input	State sequence	Input consumed?	Stops in final state?	Accepted?	Accepted by NFA?
aa	$q_0q_1q_2$	✓	✓	✓	✓
aa	q_0q_3	✗	✗	✗	
a					
a					
aaa					
aaa					



Non-deterministic Finite Automata (NFA)

- An NFA *accepts* a string *w* if and only if there is at least one computation of the NFA that accepts *w* i.e. a computation that
  - consumes the entire input, and
  - stops in a final state

Formal definition

A non-deterministic finite automaton consists of

- Q: a finite set of states
- Σ: a finite set of input symbols, an alphabet
- $\delta: Q \times \Sigma \rightarrow \mathcal{P}(Q)$: a transition function
  for each state q in Q and each symbol a in Σ it determines a set of new states $\delta(q,a)$
- q_0: the initial state; an element of Q
- F: the final states; a subset of Q

Often written as $A = (Q, \Sigma, \delta, q_0, F)$

The language of the NFA

By induction on the length of the string w:

- $\delta^*(q,\varepsilon) = \{q\}$
- $\delta^*(q,aw) = \delta^*(q_1,w) \cup \ldots \cup \delta^*(q_k,w)$

  where $\delta(q,a) = \{q_1,\ldots,q_k\}$

The language of $A = (Q, \Sigma, \delta, q_0, F)$

$L(A) = \{w \mid \delta^*(q_0,w) \text{ contains a state from } F\}$

so at least one of the attempts must lead to a final state

Formalisation

The definition in the book is different

By induction on the length of the string w:

- $\underline{\delta}(q,\varepsilon) = \{q\}$
- $\underline{\delta}(q,wa) = \delta(q_1,a) \cup \ldots \cup \delta(q_k,a)$

  where $\underline{\delta}(q,w) = \{q_1,\ldots,q_k\}$

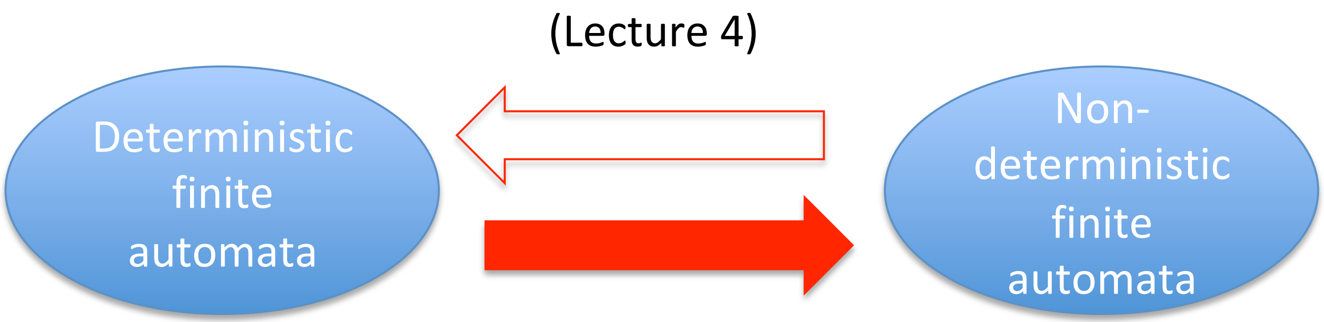
- Again one can prove that the two definitions are equivalent: $\underline{\delta}(q,w) = \delta^*(q,w)$

Let’s do it!

Exercise 2.3.4 (a)

- Consider the alphabet {0,1,…,9}. Specify a NFA for the set of strings such that the final digit has appeared before – try to take advantage of non-determinism as much as possible.

- (We will do the rest of exercise 2.3.4 later)



FROM DFA TO NFA

The embedding constructi…

Complexity: O(n)

- Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA.
- Construct the NFA $A = (Q, \Sigma, \delta, q_0, F)$ as follows:
  - Q equals Q
  - q_0 is the set q_0
  - F equals F
  - $\delta: Q \times \Sigma \rightarrow P(Q)$ is constructed from $\delta: Q \times \Sigma \rightarrow Q$

$$\boxed{\text{if } \delta(q,a) = p \text{ then } \delta(q,a) = \{p\}}$$

Theorem

- Let $A = (Q, \Sigma, \delta, q_0, F)$ is a DFA and assume that the NFA $A = (Q, \Sigma, \delta, q_0, F)$ is constructed by the embedding construction.
- Then $L(A) = L(A)$

- Proof: It is sufficient to show that $\underline{\delta}(q,w) = \{\underline{\delta}(q,w)\}$ for all q and w

BY THE WAY …
SECURITY AUTOMATA

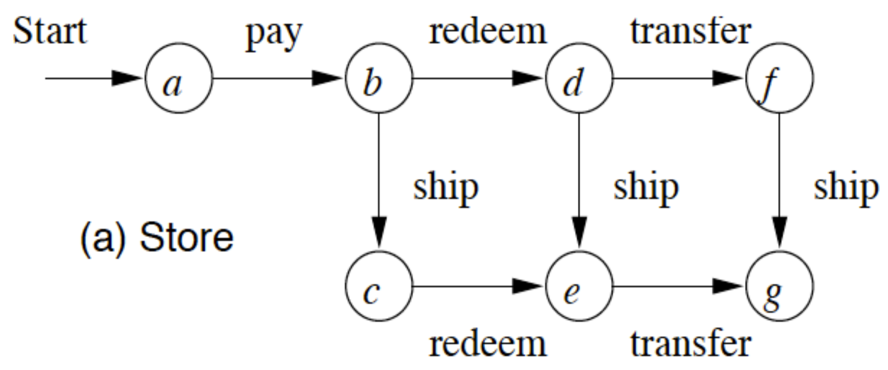
Security Automata

- Idea: construct an automaton describing what is allowed by the program
- Run the program *in parallel* with the automaton and use the automaton to check whether the action of the program is acceptable:
  - if the automaton says ok then proceed
  - if the automaton says not ok then stop

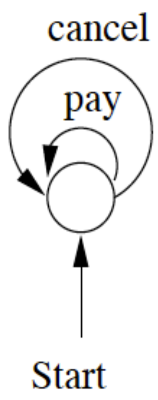
reference monitor

Example: E-commerce

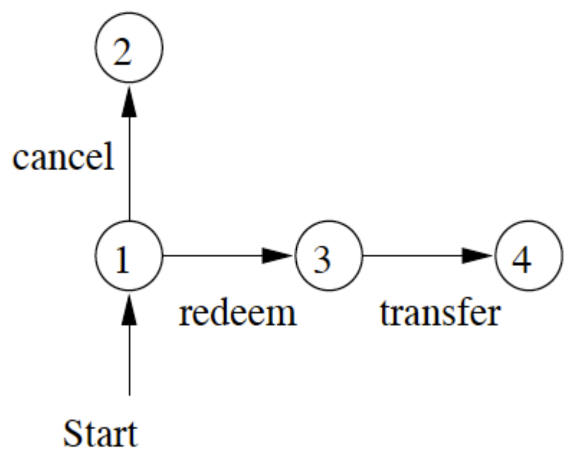
Finite automata
for each principal



(a) Store



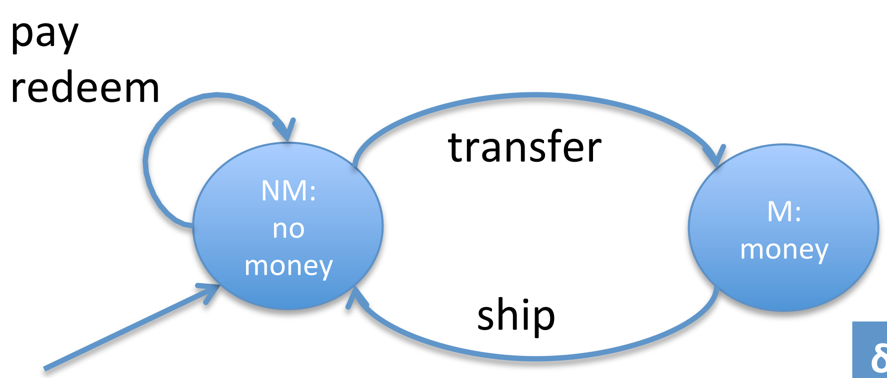
(b) Customer



(c) Bank

Example: Security automaton

- Security policy for the store: never ship goods before the money have been received from the bank:



δ	pay	redeem	transfer	ship
NM	{NM}	{NM}	{M}	Ø
M	Ø	Ø	Ø	{NM}

A security automaton

Consists of

- Q: a finite set of states
- Σ: an alphabet – the symbols being an abstraction of the actions of the program
- $\delta: Q \times \Sigma \rightarrow \mathcal{P}(Q)$: a transition function
  for each state q in Q and each symbol a in Σ it
  determines a set of new states $\delta(q,a)$
- q_0: the initial state; an element of Q
- F: the final states; F=Q

A security automaton

- Idea:
  - if the action of the program is allowed by the security automaton then proceed (that is, accept);
  - if the action of the program is not allowed then stop the execution of the program (that is, reject)
- The language of the security automaton $A = (Q, \Sigma, \delta, q_0, Q)$

  $L(A) = \{w \mid \delta^*(q_0,w) \neq \emptyset\}$

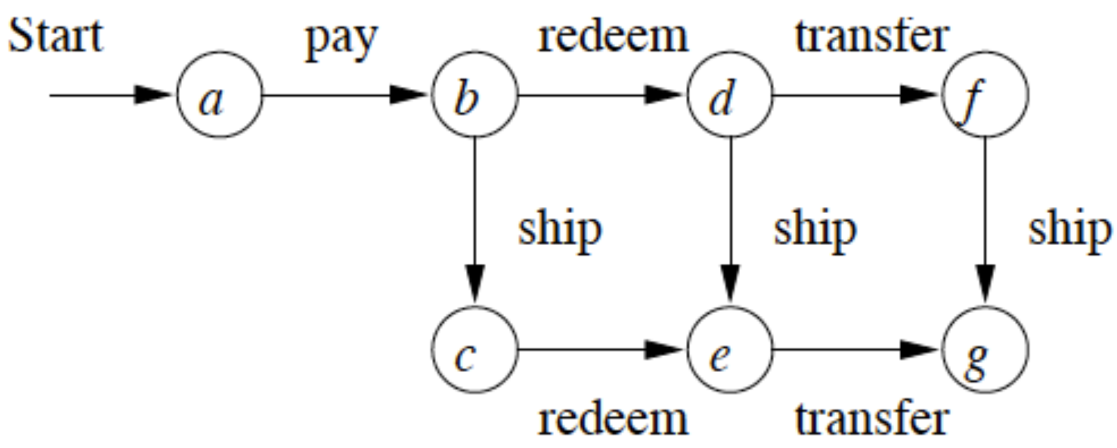
Product construction

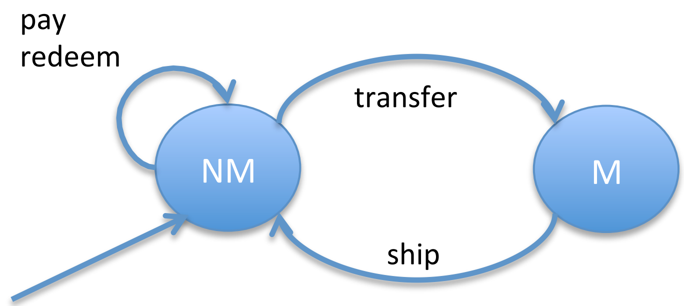
- Assume we have two NFAs with the same alphabet Σ: $(Q, \Sigma, \delta, q_0, F)$ and $(Q, \Sigma, \delta, q_0, F)$
- Then the product automaton has
  - states: $Q \times Q$ – pairs of states from the two automata
  - alphabet: Σ
  - transition function: $\delta: (Q \times Q) \times \Sigma \rightarrow \mathcal{P}(Q \times Q)$ tracks what both automata are doing:

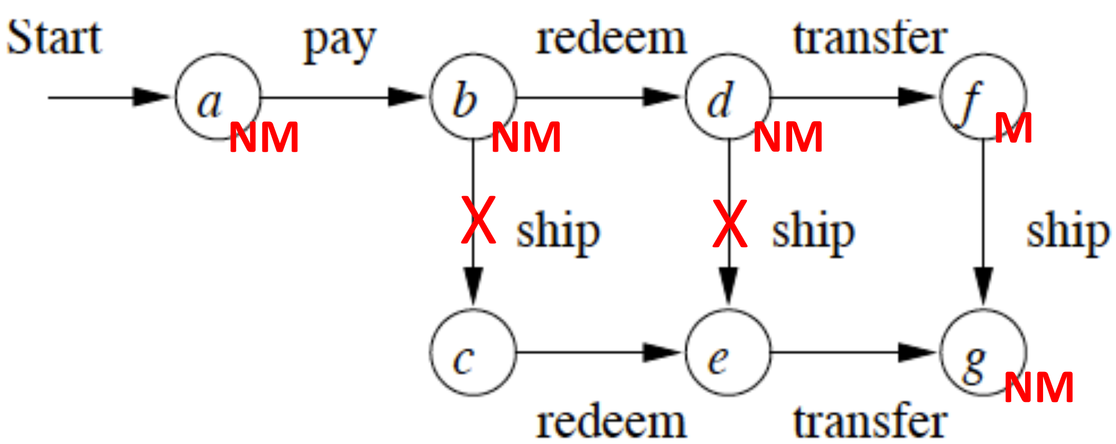
    $$\delta((q,q),a) = \{(q',q') \mid q' \text{ is in } \delta(q,a),\ q' \text{ is in } \delta(q,a)\}$$
  - initial state: $(q_0, {}_{q0})$ – pairs of initial states
  - final states: $F \times F$ – pairs of final states

Note: if that if $\delta(q,a)=\emptyset$ then also $\delta((q,q),a)=\emptyset$ for all q

Combining it with the security automaton



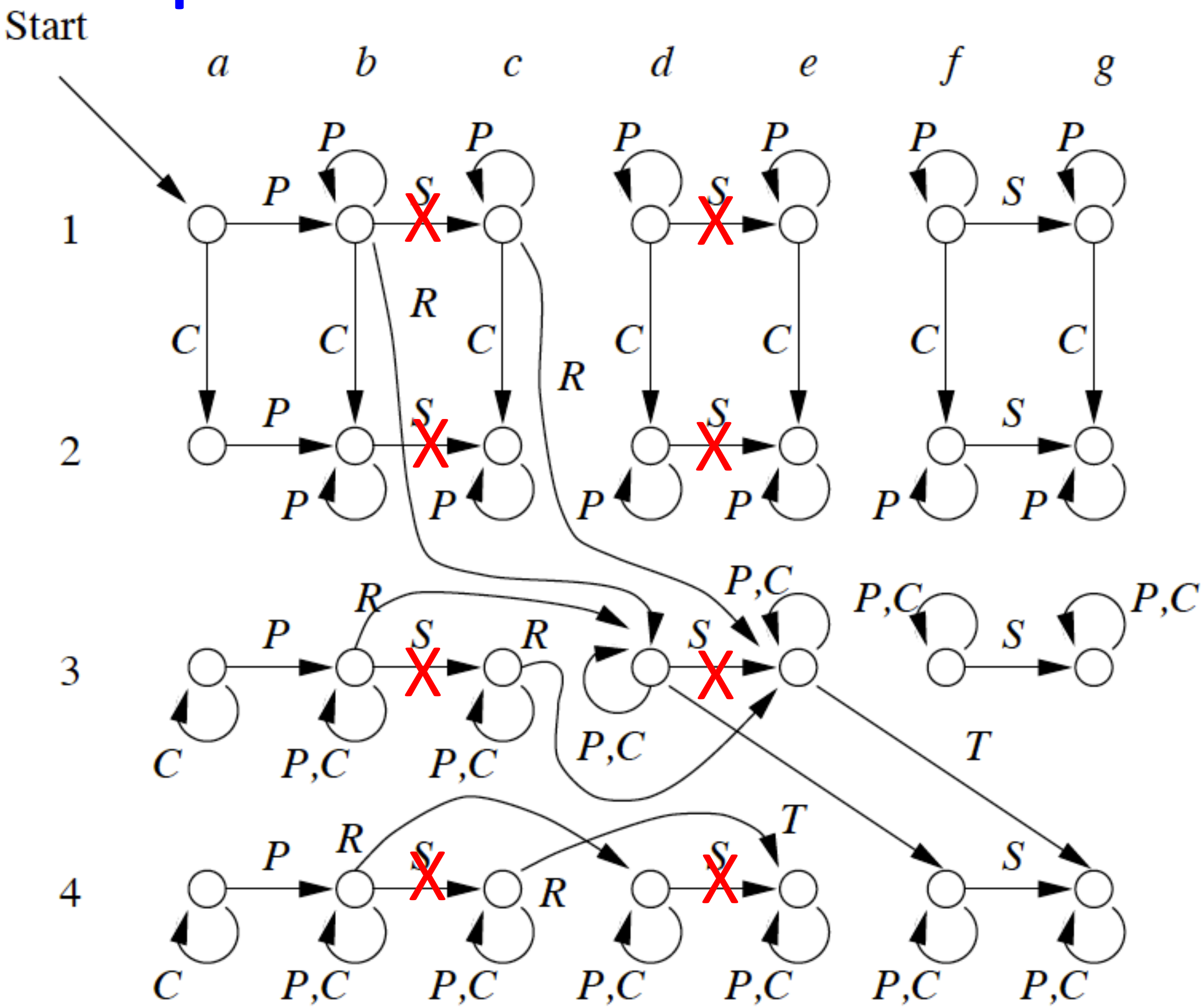




Example: E-commerce

The combined finite automaton

Product automaton: Keep track of the state of all three automata while synchronising on joint actions



P: pay
S: ship
C: cancel
R: redeam
T: transfer

READING MATERIAL AND EXERCISES

Reading material and exercises

- Covered in the lecture today:
  - HMU chapter 1: pages 19-26
  - HMU chapter 2: pages 45-52 and 55-60
- Topic of the next lecture:

  Regular expressions

  to be based on HMU section 3.1, 3.2 and 3.4
- Exercises for today
  - Building and understanding DFAs: HMU 2.2.4, 2.2.10
  - Building and understanding NFAs: HMU 2.3.4
  - Proofs: HMU 2.2.2, 2.2.3