# Deterministic simulation of a NFA with k-symbol lookahead

SOFSEM 2007

Bala Ravikumar, California State University

(joint work with Nic Santean, University of Waterloo)

# Overview

- Definitions: DFA, NFA and lookahead DFA
- Motivation: automated e-service composition and delegation
- The NFA delegation model, examples, properties

delegator

- Characterization of existence of delegator
- Complexity of finding a delegator for unambiguous and general NFA
- An algorithm for general NFA delegation
- Conclusion, further work

# DFA – Deterministic Finite Automaton

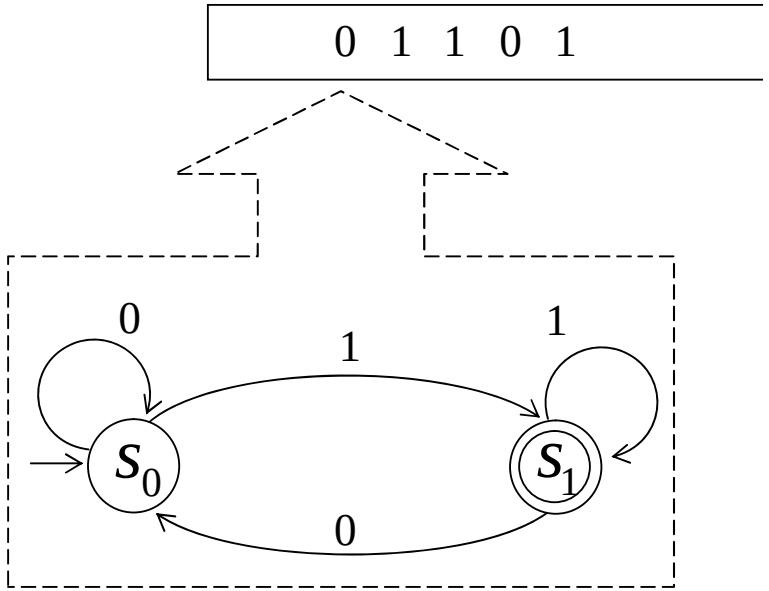
state set, alphabet, state, accepting states

$$A = (Q, \Sigma, \delta, s_0, F)$$

$$\delta : Q \times \Sigma \xrightarrow{\sim} Q$$

$\delta(s_0, 0) = s_0$

$\delta(s_0, 1) = s_1$ etc.



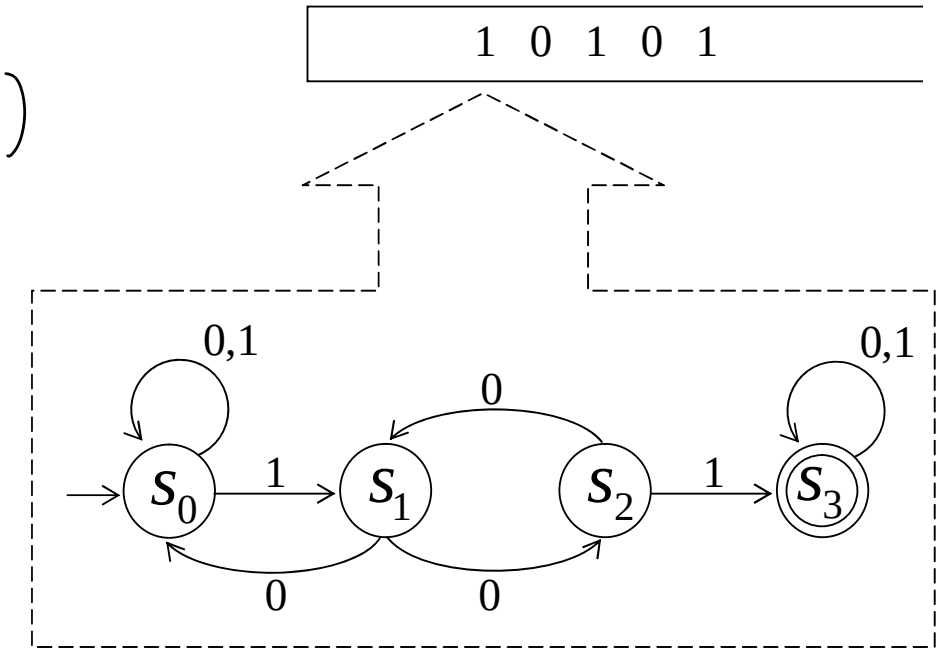

$$L(A) = \left\{ w \in \Sigma^* \,\middle|\, (w)_2 \text{ is odd} \right\} = (0+1)^* 1$$

# NFA – Nondeterministic Finite Automaton

$$M = (Q, \Sigma, \delta, s_0, F)$$

$$\delta : Q \times \Sigma \to P(Q) \quad (2^Q)$$

$\delta(s_0, 1) = \{s_0, s_1\}$



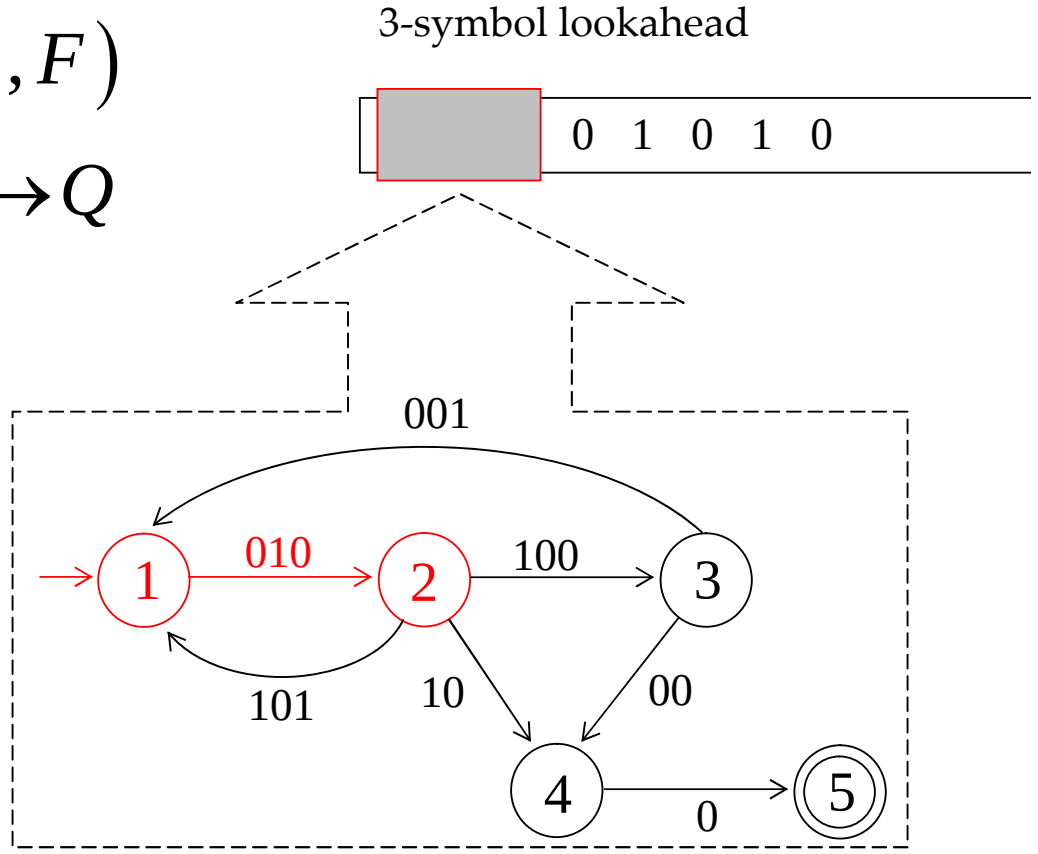

$$L(A) = (0+1)^* 1\, 0 (00)^* 1 (0+1)^*$$

# k-symbols Lookahead DFA

$|\Sigma| + |\Sigma|^2 + \ldots + |\Sigma|^k$

$$D = (Q, \Sigma, \delta, s_0, F)$$

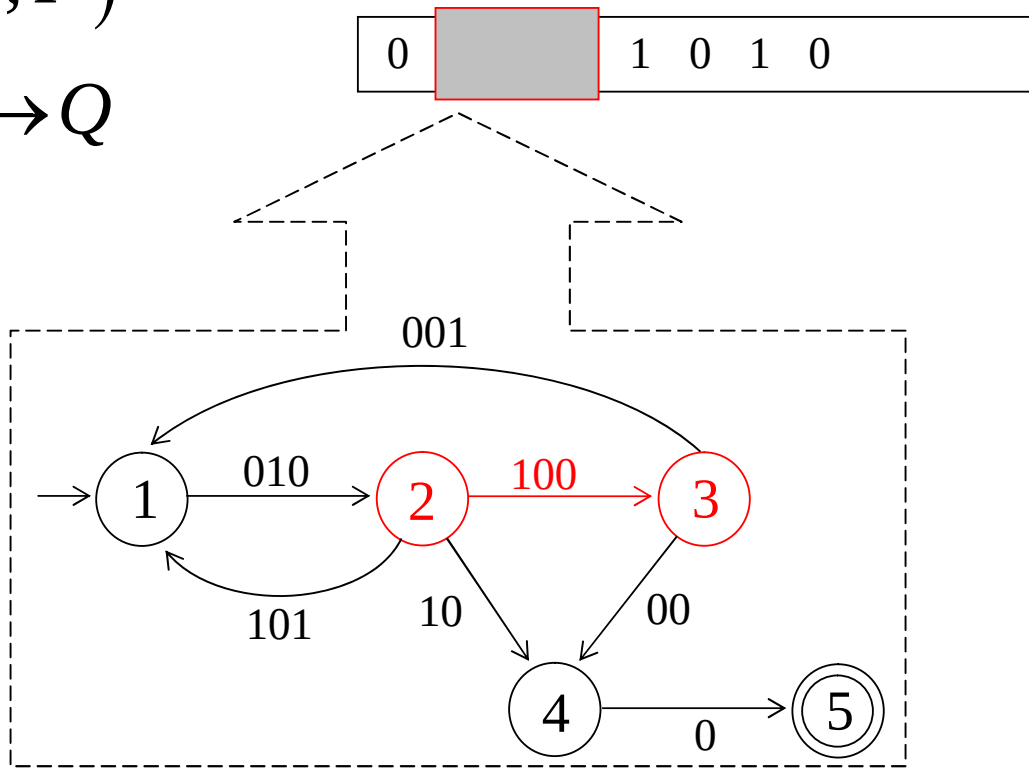
$$\delta : Q \times \Sigma^{\leq k} \xrightarrow{\sim} Q$$



# k-symbols Lookahead DFA

$$D = (Q, \Sigma, \delta, s_0, F)$$

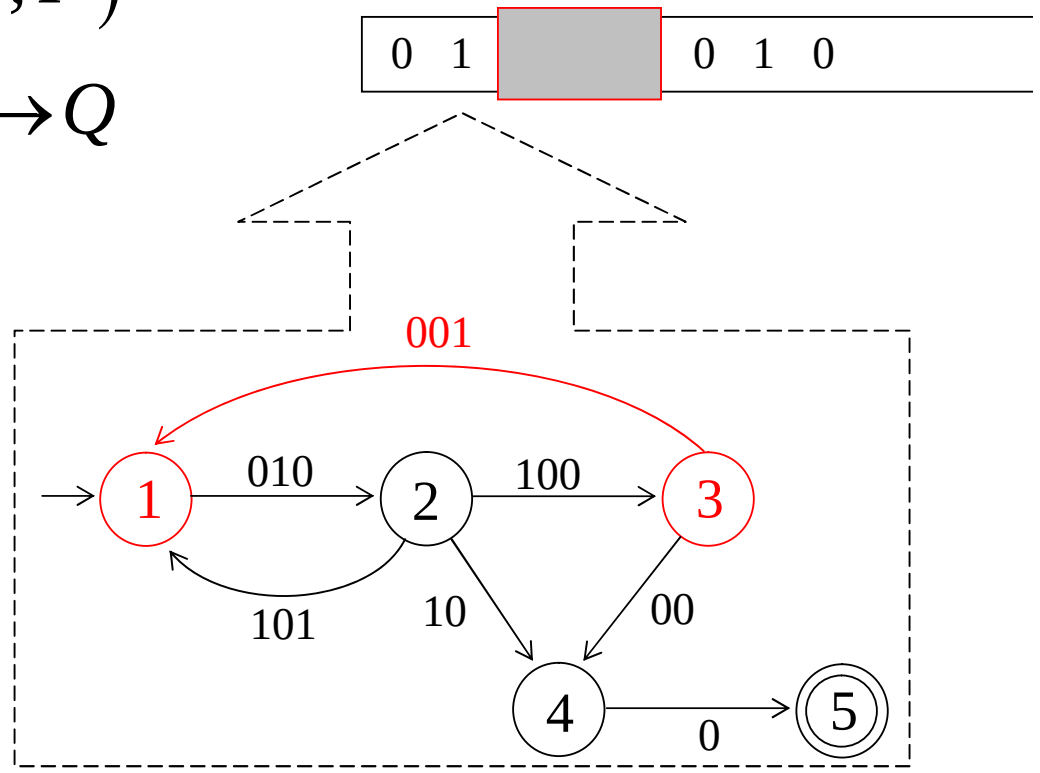
$$\delta : Q \times \Sigma^{\leq k} \xrightarrow{\sim} Q$$



# k-symbols Lookahead DFA

$$D = (Q, \Sigma, \delta, s_0, F)$$

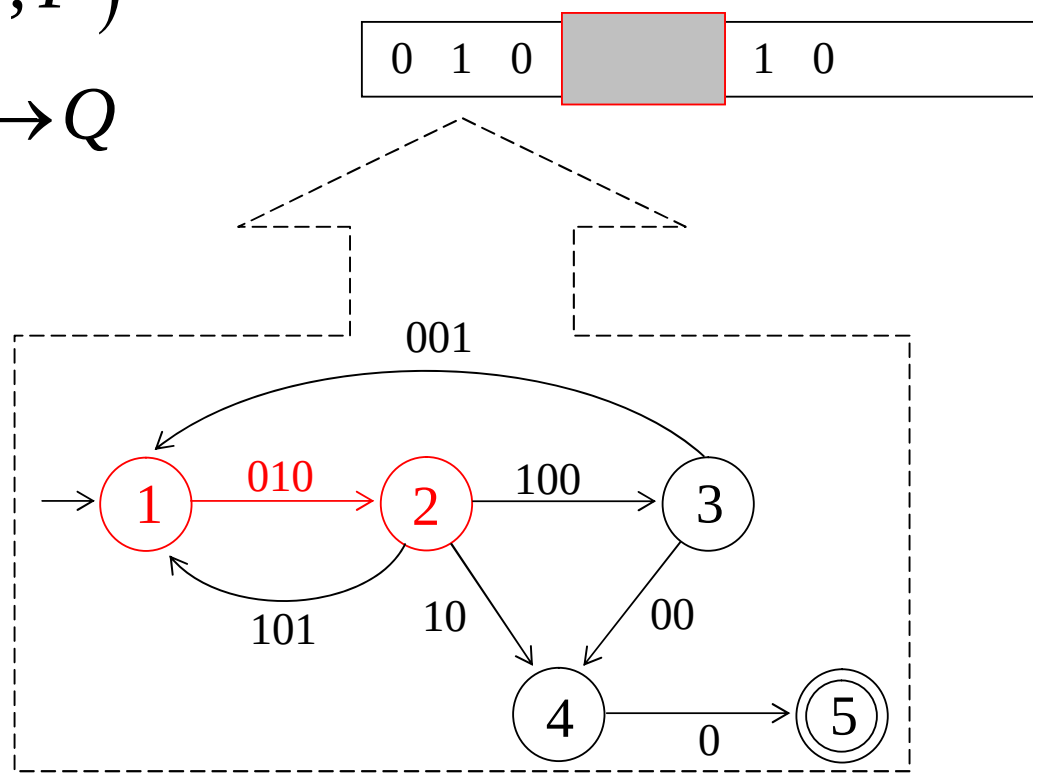
$$\delta : Q \times \Sigma^{\leq k} \xrightarrow{\sim} Q$$



# k-symbols Lookahead DFA

$$D = (Q, \Sigma, \delta, s_0, F)$$

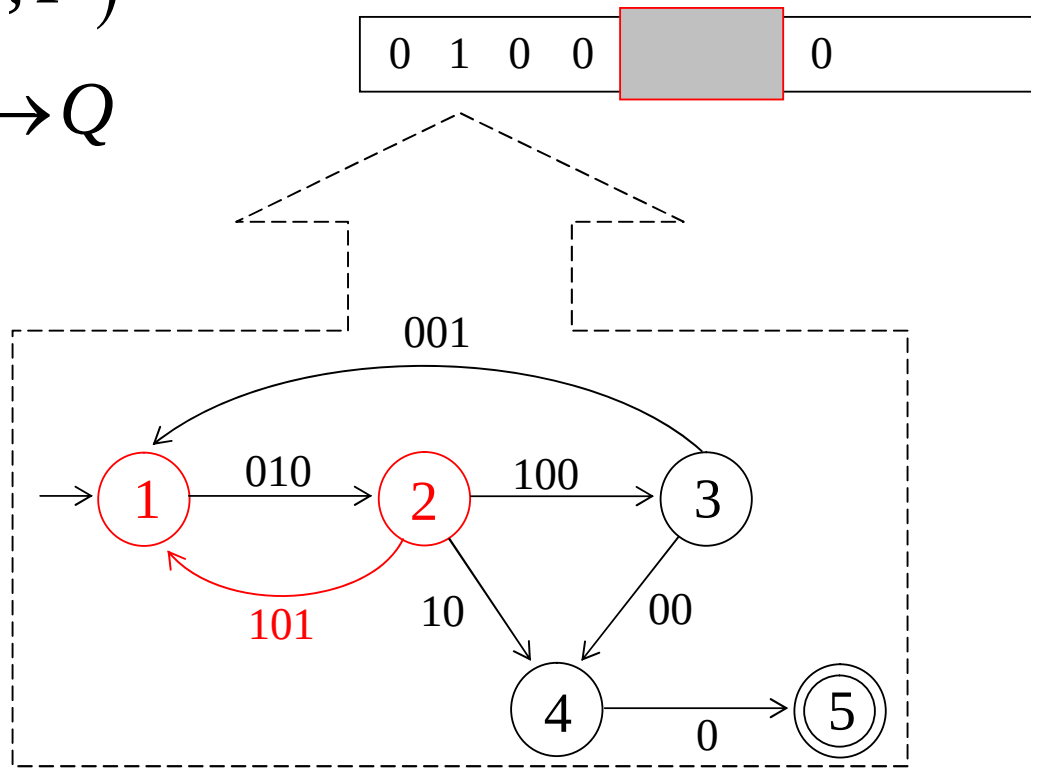
$$\delta : Q \times \Sigma^{\leq k} \xrightarrow{\sim} Q$$



# k-symbols Lookahead DFA

$$D = (Q, \Sigma, \delta, s_0, F)$$

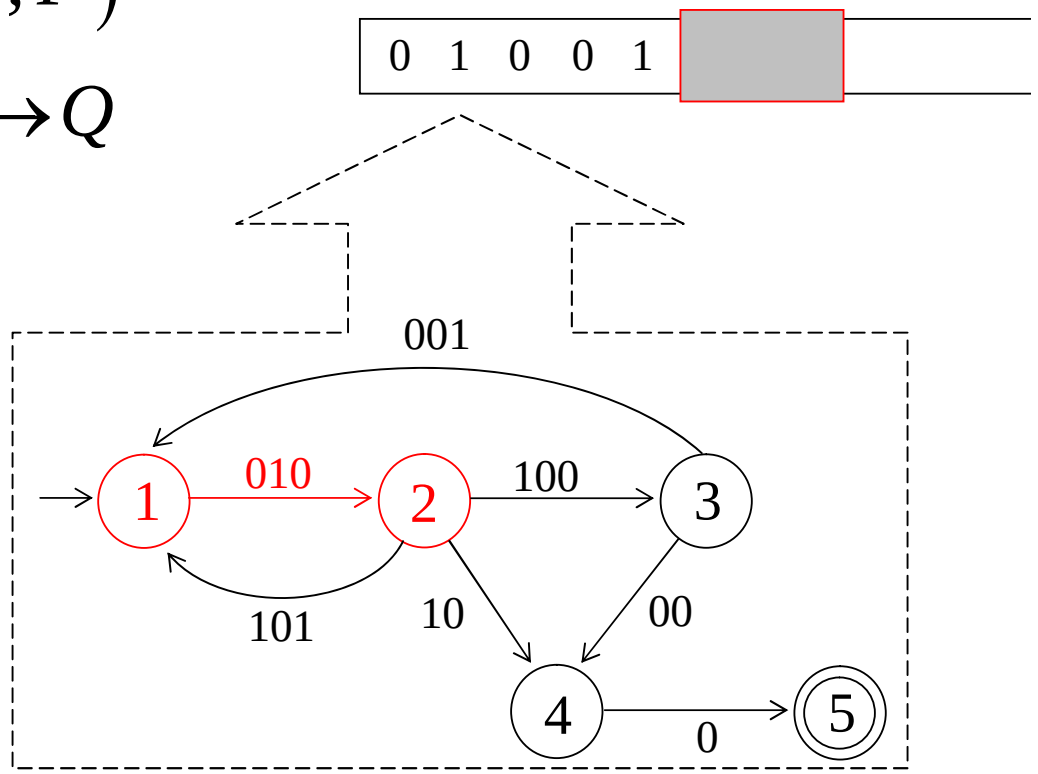
$$\delta : Q \times \Sigma^{\leq k} \xrightarrow{\sim} Q$$



# k-symbols Lookahead DFA

$$D = (Q, \Sigma, \delta, s_0, F)$$

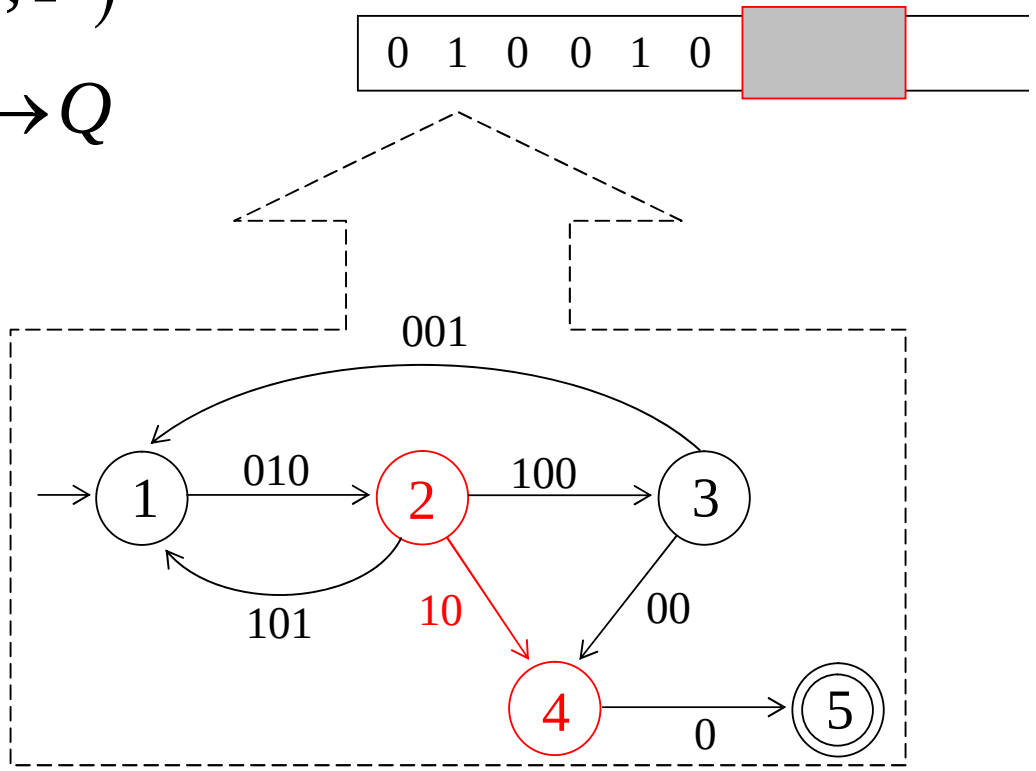
$$\delta : Q \times \Sigma^{\leq k} \xrightarrow{\sim} Q$$



# k-symbols Lookahead DFA

$$D = (Q, \Sigma, \delta, s_0, F)$$

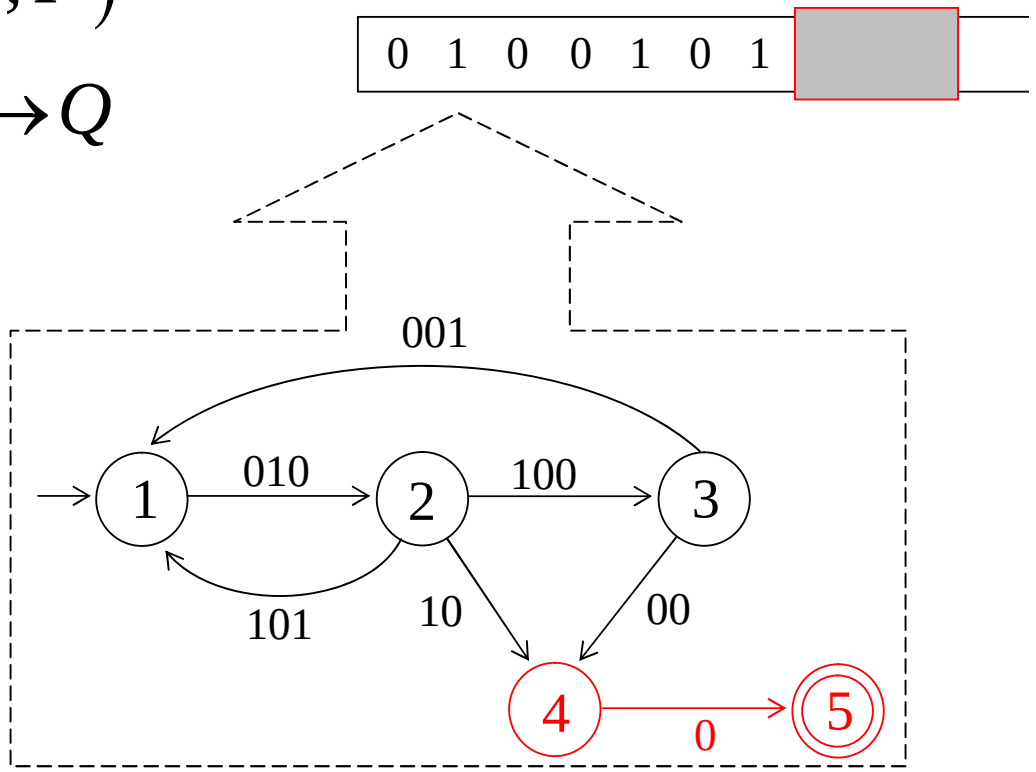
$$\delta : Q \times \Sigma^{\leq k} \xrightarrow{\sim} Q$$



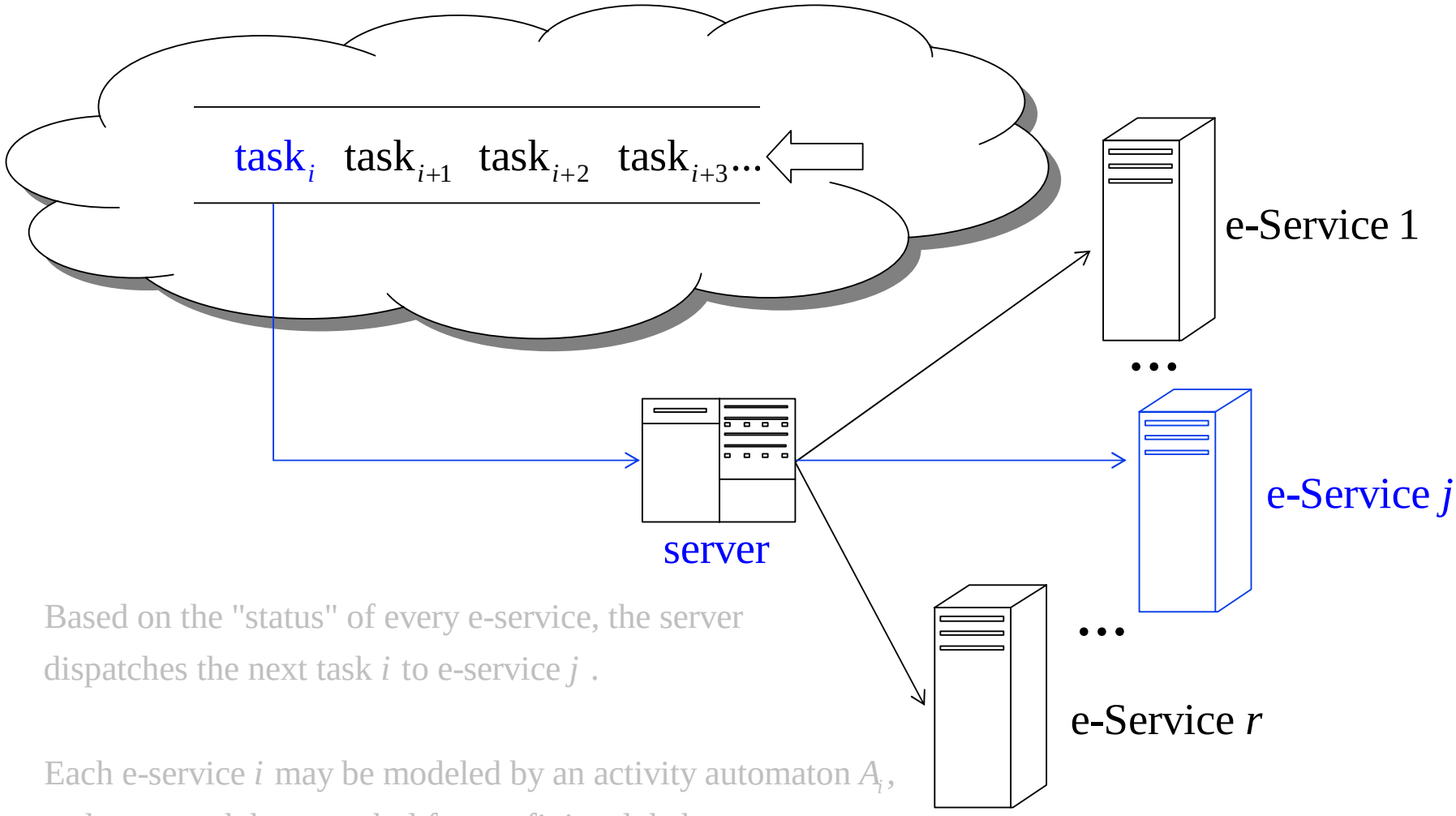
# k-symbols Lookahead DFA

$$D = (Q, \Sigma, \delta, s_0, F)$$

$$\delta : Q \times \Sigma^{\leq k} \xrightarrow{\sim} Q$$



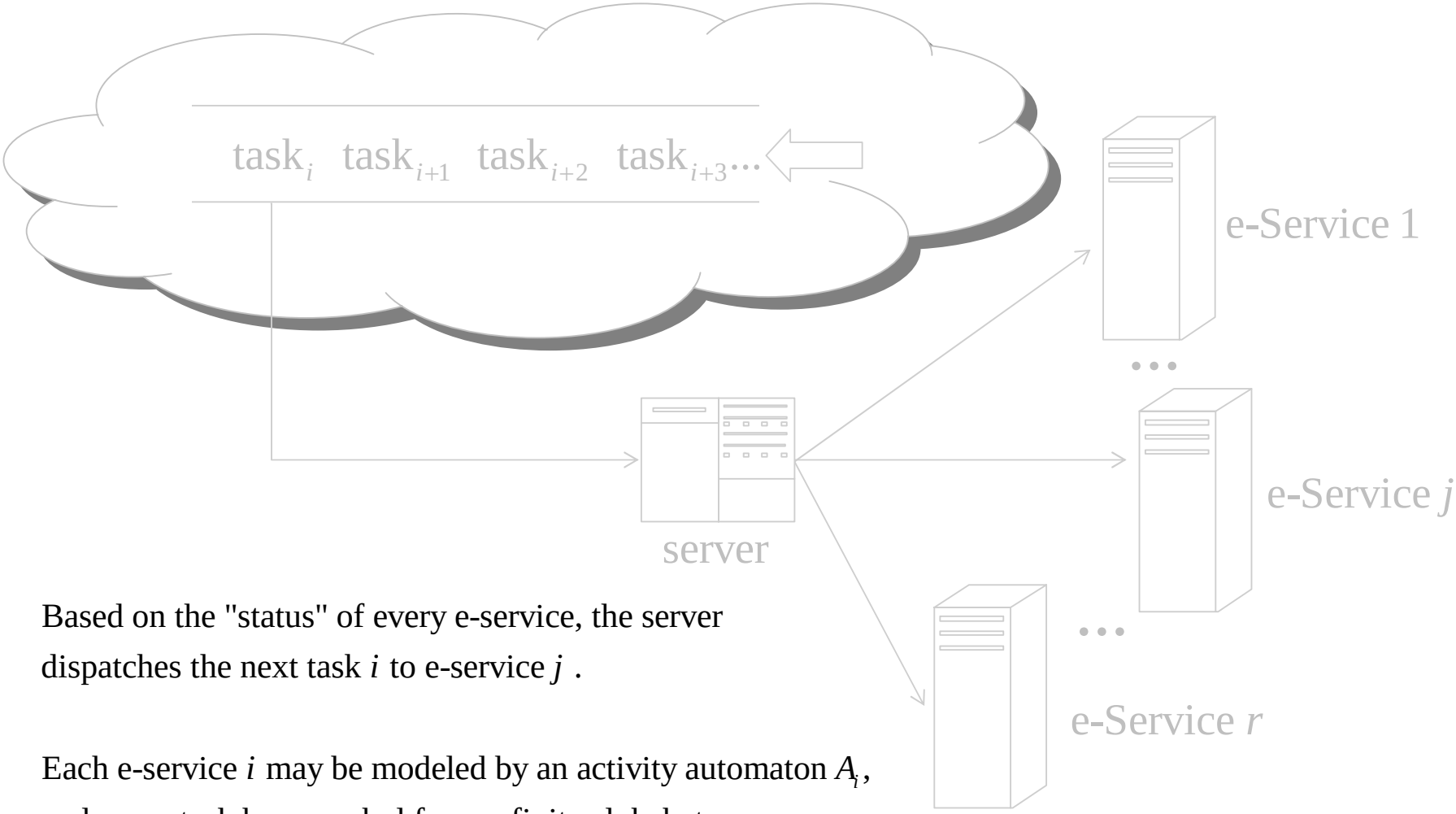
# E-Services Composition and Delegation




Based on the "status" of every e-service, the server dispatches the next task $i$ to e-service $j$ .

Each e-service $i$ may be modeled by an activity automaton $A_i$, and every task by a symbol from a finite alphabet.

# E-Services Composition and Delegation




Based on the "status" of every e-service, the server dispatches the next task $i$ to e-service $j$ .

Each e-service $i$ may be modeled by an activity automaton $A_i$, and every task by a symbol from a finite alphabet.

# Composition and Delegation Models

$A$: represents valid sequences of atomic tasks (e.g. a web session)

$A_i$: activity automaton for a specific e-service $i$

shuffle product

$\langle \overrightarrow{A}; A_1, ..., A_r \rangle$ is composable if $L(A) \subseteq L(A_1) \# L(A_2) \# ... \# L(A_r)$

or interleaving

(e.g. *xaayby* $\in$ *aab* # *xyy* )

This is a necessary condition to ensure that the web application is feasible. Question: is it sufficient?

# Composition and Delegation Models

$A$: represents valid sequences of atomic tasks (e.g. a web session)

$A_i$: activity automaton for a specific e-service $i$

$\langle A; A_1, ..., A_r \rangle$ is composable if

$$L(A) \subseteq L(A_1) \# L(A_2) \# ... \# L(A_r)$$

(e.g. $x\underline{a}\underline{a}y\underline{b}y \in aab \# xyy$ )

This is a necessary condition to ensure that the web application is feasible. Question: is it sufficient?

# Composition and Delegation

$A$ : $q_1$

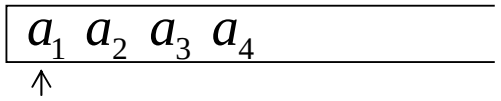
$A_1$ : $p_1$

$A_2$ : $p_2$

...

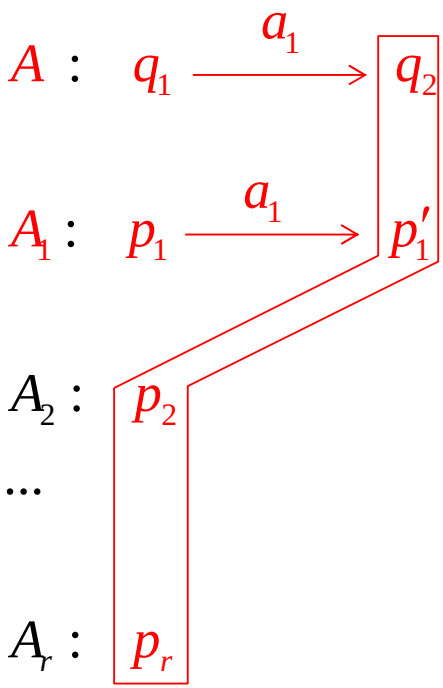
$A_r$ : $p_r$

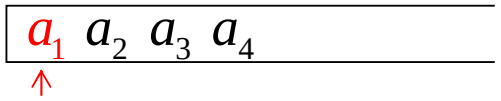
$(q_1, p_1, p_2, \ldots, p_r)$




$A \cap (A_1 \# \ldots \# A_r)$

# Composability and Delegation

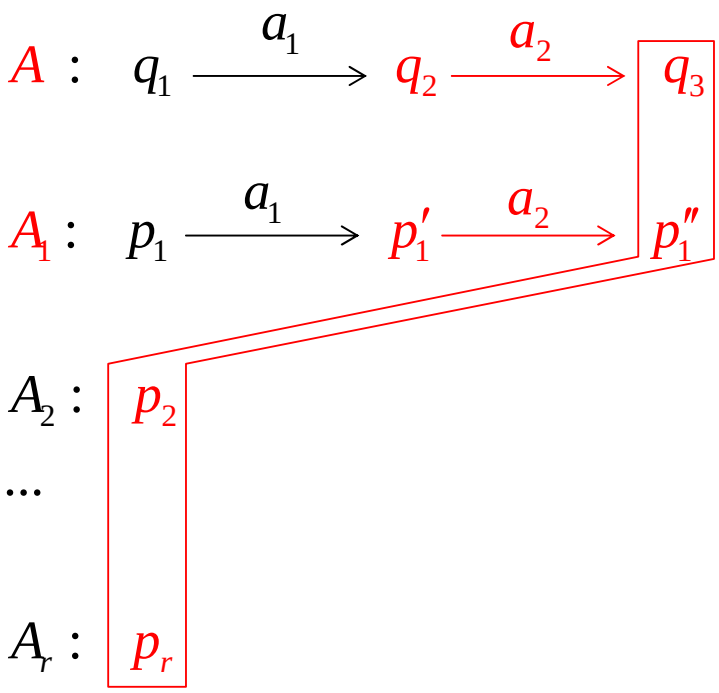


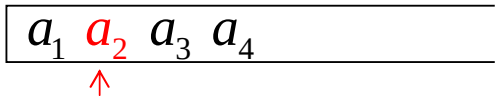




$A \cap (A_1 \# \ldots \# A_r)$

$(q_1, p_1, p_2, \ldots, p_r) \xrightarrow{a_1} (q_2, p_1', p_2, \ldots, p_r)$

# Composability and Delegation

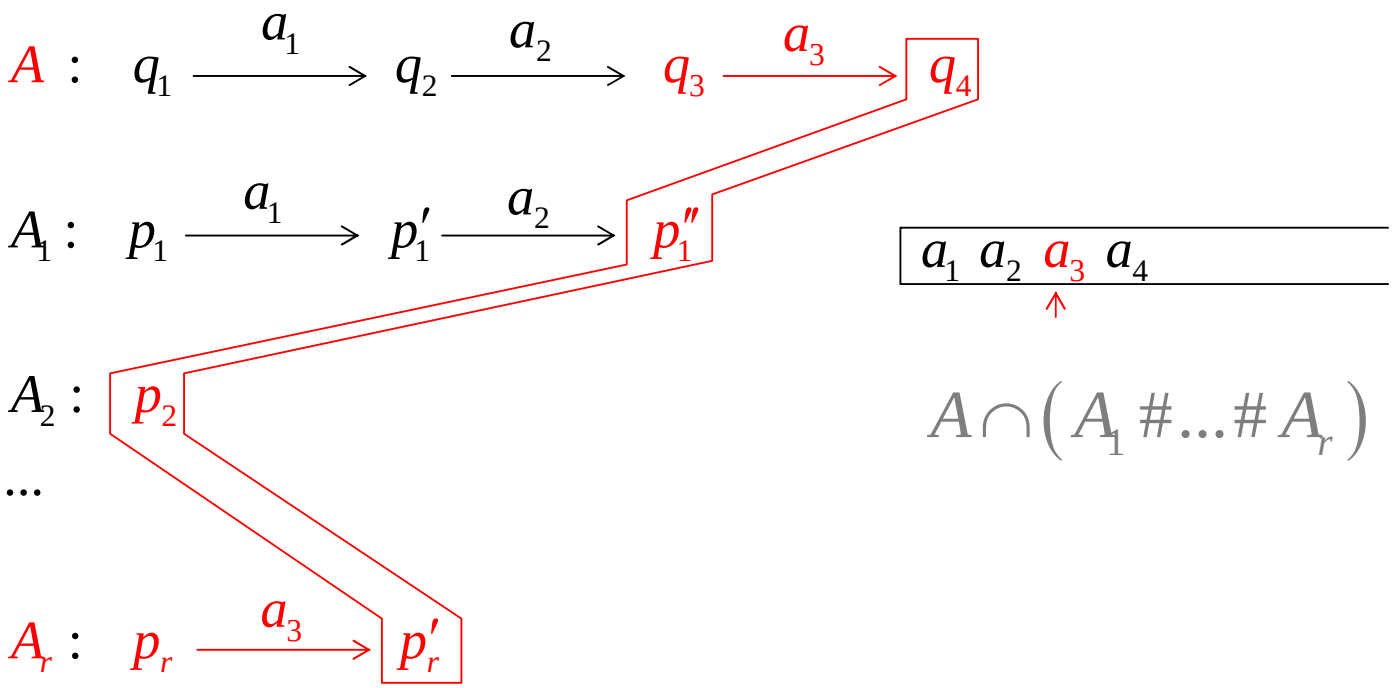



$$A \cap (A_1 \# \ldots \# A_r)$$

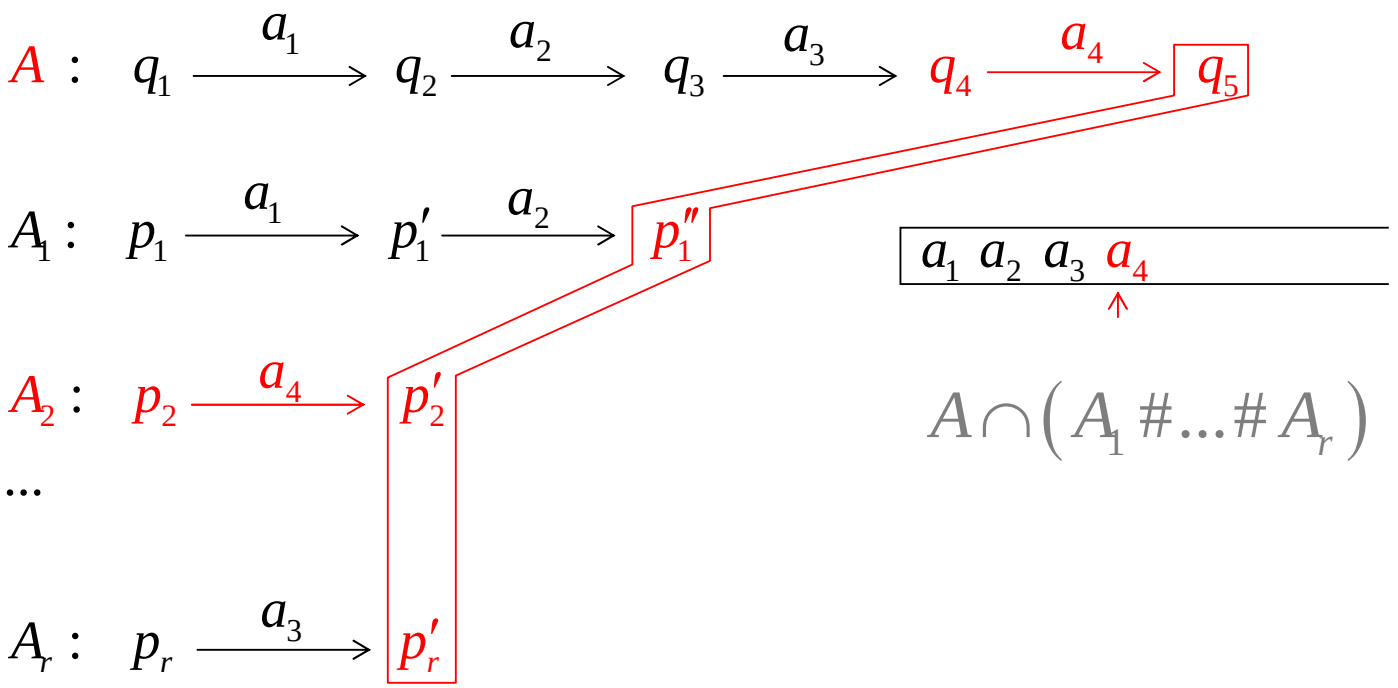
$$(q_1, p_1, p_2, \ldots, p_r) \xrightarrow{a_1} (q_2, p_1', p_2, \ldots, p_r) \xrightarrow{a_2} (q_3, p_1'', p_2, \ldots, p_r)$$

# Composability and Delegation




$$(q_1, p_1, p_2, \ldots, p_r) \xrightarrow{a_1} (q_2, p_1', p_2, \ldots, p_r) \xrightarrow{a_2} (q_3, p_1'', p_2, \ldots, p_r) \xrightarrow{a_3}$$
$$\xrightarrow{a_3} (q_4, p_1'', p_2, \ldots, p_r')$$

# Nondeterministic Delegator




$$(q_1, p_1, p_2, \ldots, p_r) \xrightarrow{a_1} (q_2, p_1', p_2, \ldots, p_r) \xrightarrow{a_2} (q_3, p_1'', p_2, \ldots, p_r) \xrightarrow{a_3}$$
$$\xrightarrow{a_3} (q_4, p_1'', p_2, \ldots, p_r') \xrightarrow{a_4} (q_5, p_1'', p_2', \ldots, p_r')$$

# k-Lookahead Delegation Model

$M = (Q, \Sigma, \delta, s_0, F)$ : the composite NFA $A \cap (A_1 \# ... \# A_r)$

How can we use M deterministically ?

a $k$ – delegator for $M$ is an equivalent $k$-symbol lookahead DFA $(Q, \Sigma, \delta', s_0, F)$, with $\delta' : Q \times \Sigma^{\leq k} \xrightarrow{\sim} Q$ verifying:

$$\forall\, (q, a_1 ... a_i) \in Q \times \Sigma^{\leq k} : \quad \delta'(q, a_1 ... a_i) \in \delta(q, a_1),\ i \leq k$$

# k-Lookahead Delegation Model

$\delta(Q, a_1 \dots a_r)$

$= \delta(\delta(Q, a_1 \dots a_{r-1}), a_r)$

augment using k symbol inputs

$M = (Q, \Sigma, \delta, s_0, F)$ : the composite NFA $A \cap (A_1 \# \dots \# A_r)$

How can we use M deterministically ?

a $k-$delegator for $M$ is an equivalent $k$-symbol lookahead DFA $(Q, \Sigma, \delta', s_0, F)$, with $\delta' : Q \times \Sigma^{\leq k} \xrightarrow{\sim} Q$ verifying:

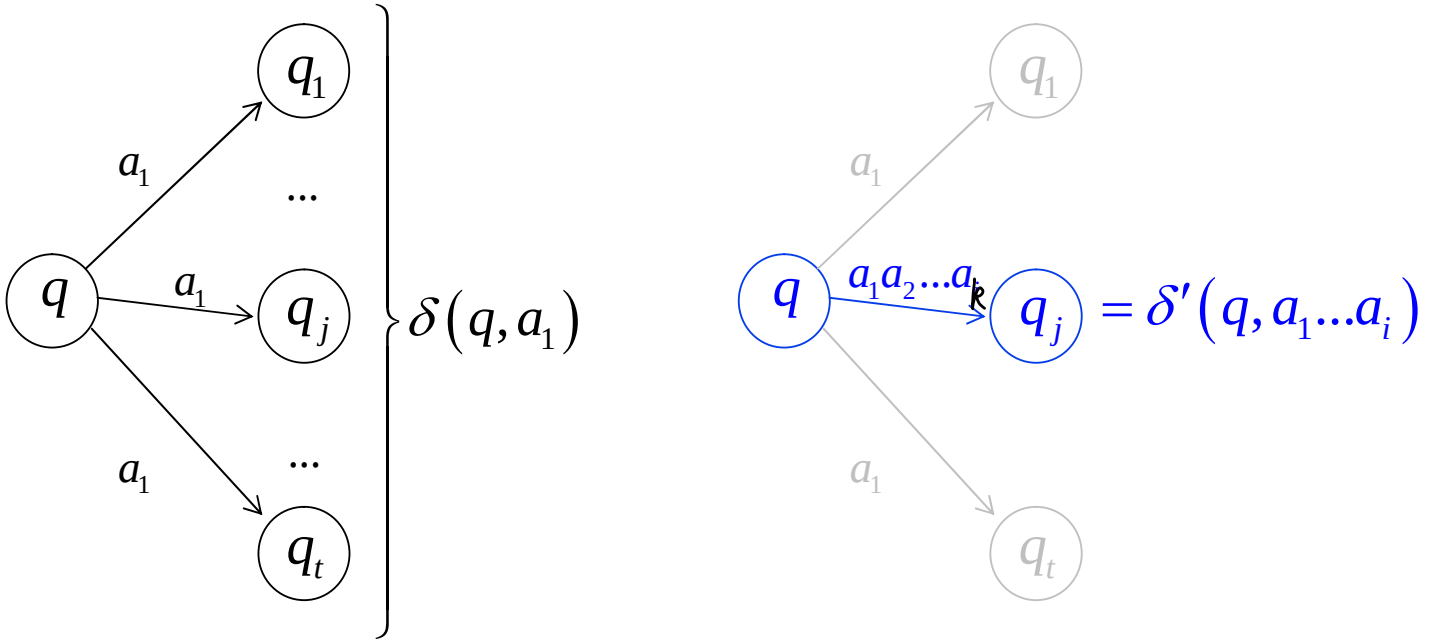
$$\forall\, (q, a_1 \dots a_i) \in Q \times \Sigma^{\leq k} : \quad \delta'(q, a_1 \dots a_i) \in \delta(q, a_1),\ i \leq k$$

# k-Lookahead Delegation Model

Interpretation:

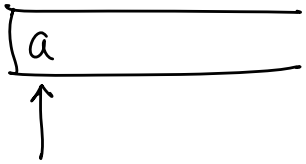
an NFA $M$

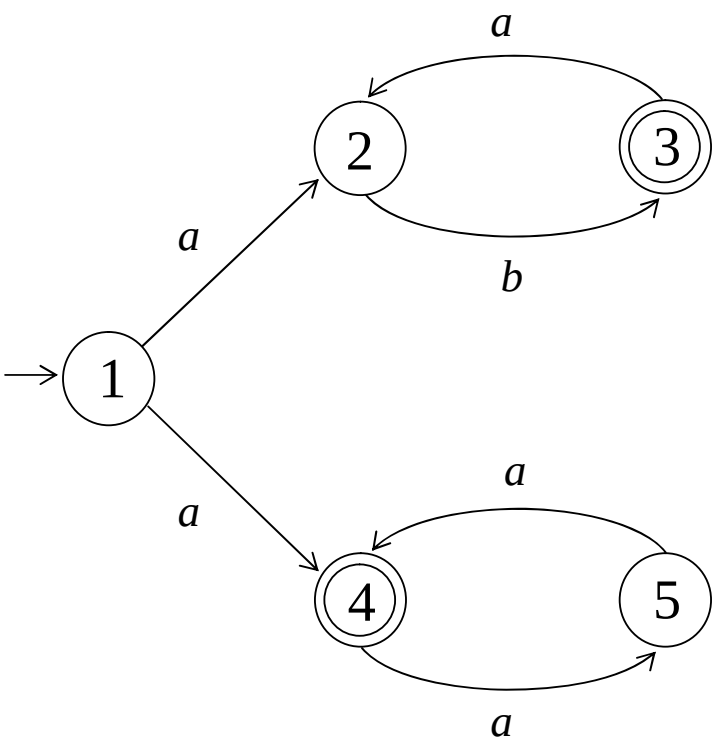
a $k$ – delegator for $M$




$$\forall\, (q, a_1 ... a_i) \in Q \times \Sigma^{\leq k} : \quad \delta'(q, a_1 ... a_i) \in \delta(q, a_1)$$

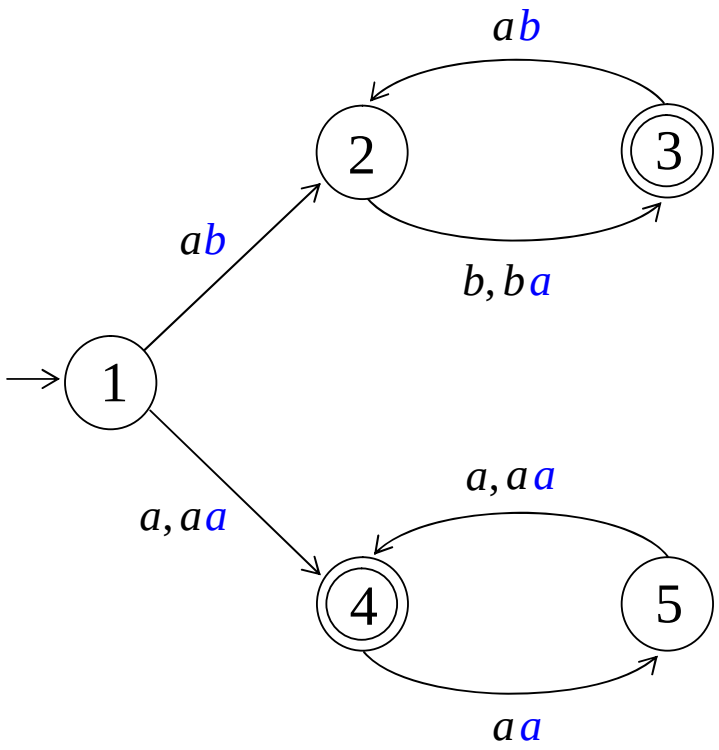
# Example: an unambiguous NFA




$$a\left(b(ab)^{*}+(aa)^{*}\right)$$

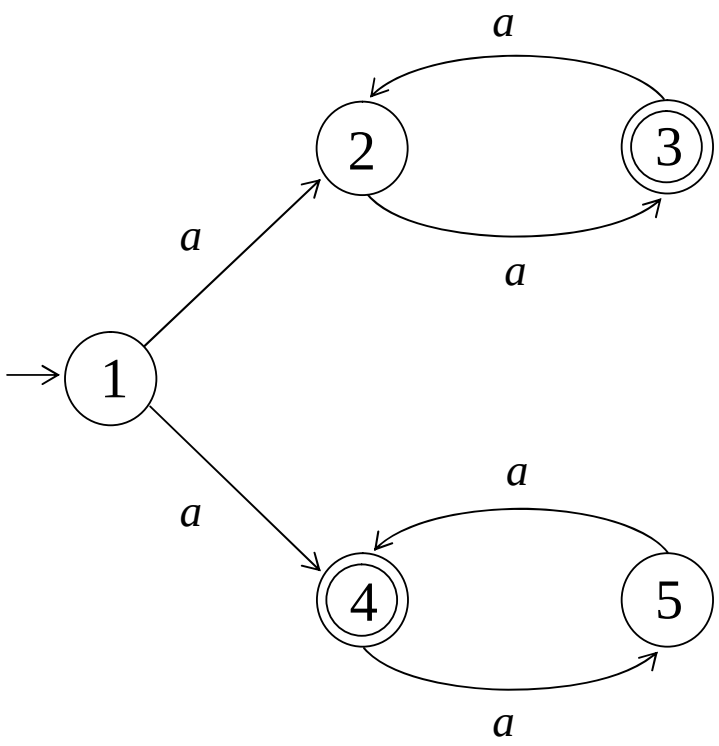
# Example: a 2-delegator




$$a\left(b(ab)^* + (aa)^*\right)$$

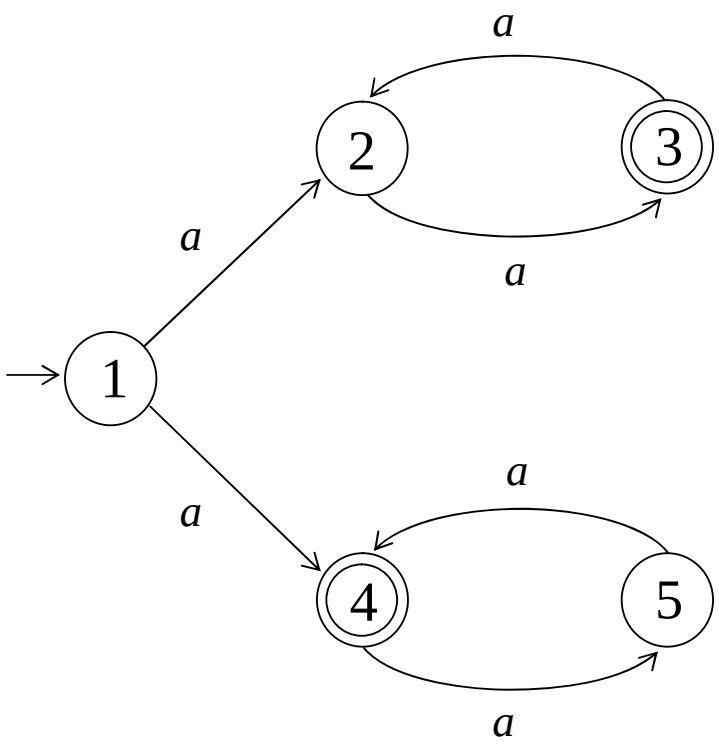
This unambiguous NFA has a 2-delegator.

# Example: an unambiguous NFA




$\left(a^{+}\right)$

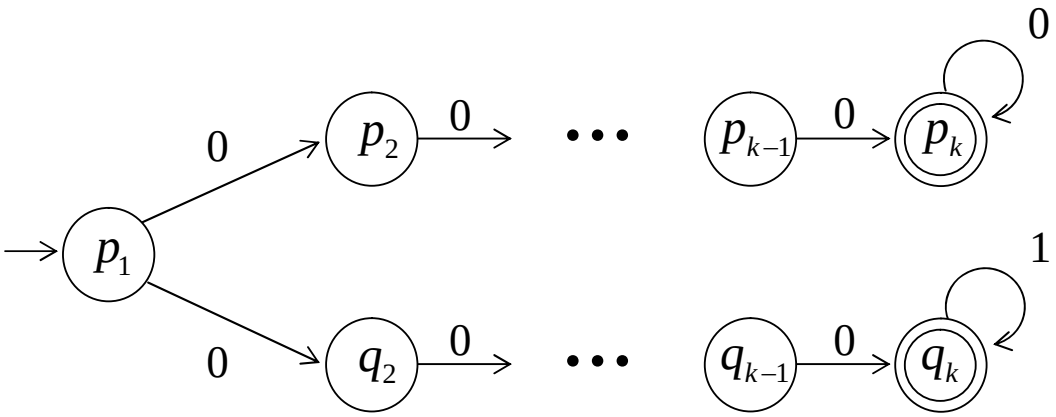
# Example: no delegators




$\left(a^{+}\right)$

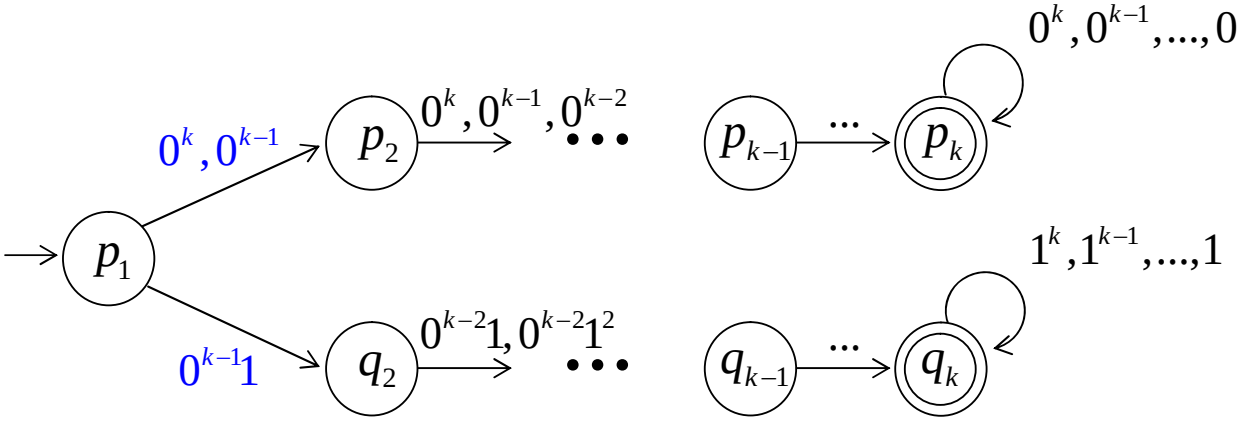
This unambiguous NFA has no $k$-delegator, $\forall k > 0$.

# Example: an ambiguous NFA




$$0^{k-1}\left(0^* + 1^*\right)$$

# Example: a k-delegator, no (<k)-delegator

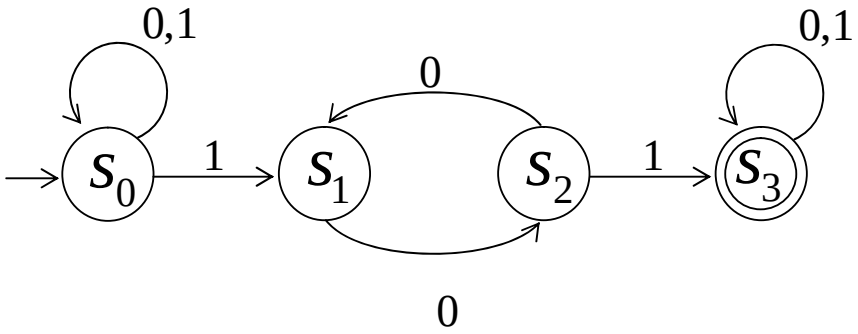



$$0^{k-1}\left(0^* + 1^*\right)$$

This NFA has a $k$-delegator but no $(k-1)$- delegators.

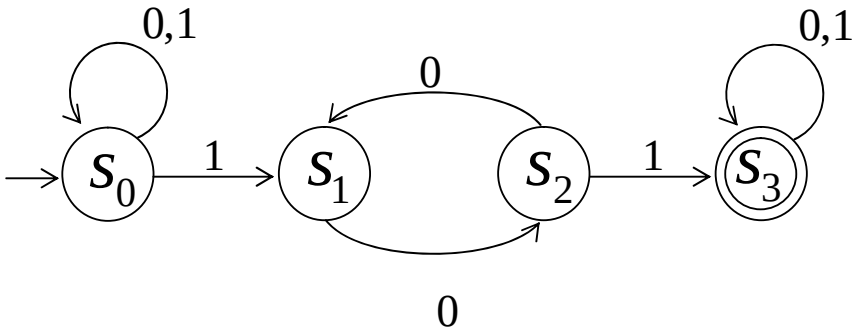
Note: $k$-delegation $\Rightarrow (k+1)$-delegation

# Example: an ambiguous NFA




$$(0+1)^* 1\ 0(00)^* 1\ (0+1)^*$$

# Example: no delegators




$$(0+1)^* 1\, 0 (00)^* 1 (0+1)^*$$

This NFA has no $k$-delegator, for any $k > 0$.

idea: take $\underline{10^{k-1}}0^k1$ and $\underline{10^{k-1}}0^{k+1}101$

# Why Study NFA Delegation

$O(2^k \cdot n)$

- E-Service Composition – *given some specifications and a set of e-services, implement an application which follows the specifications.*

- Deterministic simulation of NFA: alternative to classical subset construction to simulate an NFA by a DFA. Linear blow-up vs. exponential blow-up.

- A theoretical metric for nondeterminism: *if an NFA M has a 3-delegator and M' does not have a k-delegator for, say k < 10, then M' is "more nondeterministic" than M.*

# Language vs. Machine Property

1'st Question:

Is delegation a language property or a machine property?

(there exist infinitely many NFA for a given regular language)

# Delegation as a Language Property

$L$ - regular language

1. $L$ is weakly delegable if

$$\left(\forall M_{NFA \text{ for } L}\right)\left(\exists k_{k>0}\right)\left(M \text{ has a } k-\text{delegator}\right)$$

1. $L$ is strongly delegable if

$$\left(\exists k_{k>0}\right)\left(\forall M_{NFA \text{ for } L}\right)\left(M \text{ has a } k-\text{delegator}\right)$$

# Delegation as a Language Property

*Theorem.*

$$L \text{ is strongly deleg.} \Leftrightarrow L \text{ is weakly deleg.} \Leftrightarrow L \text{ is finite}$$

$$\Leftarrow \qquad\qquad \Rightarrow$$

*Consequence.*

It is more interesting to see delegability as a machine property.

# Complexity of Finding a k-Delegator

Problems:

$(P_1)$ $k$ is an integer not part of the input.

*input* : an NFA $M$

*output* : *YES* if $M$ has a $k-$delegator, *NO* otherwise

$(P_2)$

*input* : an NFA $M$ and an integer $k$ (k in unary)

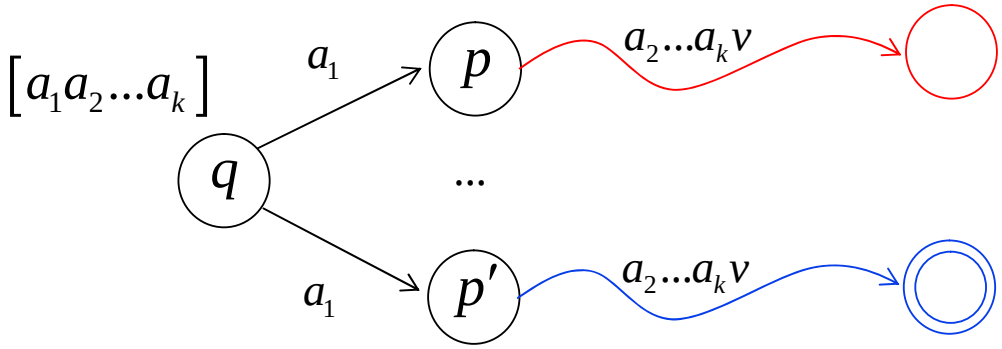
*output* : *YES* if $M$ has a $k-$delegator, *NO* otherwise

$(P_3)$

*input* : an NFA $M$ does there exist a k s.t. M has a k-delegator?

*output* : *YES* if $M$ has a delegator, *NO* otherwise

# State Blindness



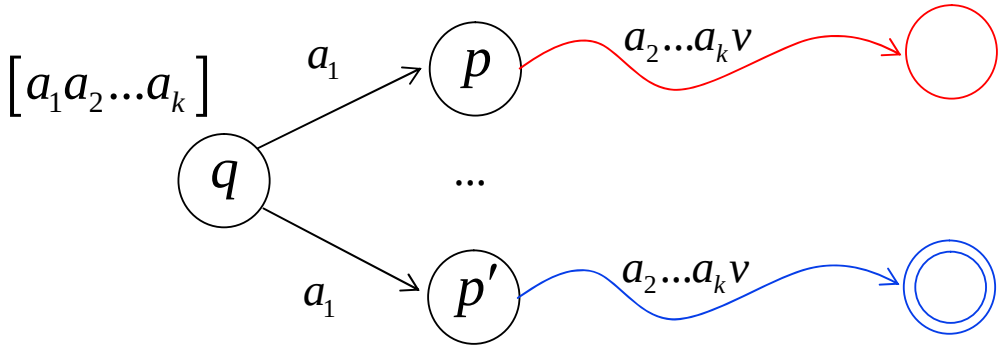

$q$ is $a_1 a_2 ... a_k$-blind if

$$\forall p \in \delta(q, a_1) \quad \exists p' \in \delta(q, a_1), v \in \Sigma^* :$$

$$\delta(p, a_2 ... a_k v) \notin F, \ \delta(p', a_2 ... a_k v) \in F$$

a state is $k$-blind if there exists $a_1 a_2 ... a_k$ such that it is $a_1 a_2 ... a_k$-blind

# State Blindness




$q$ is $a_1 a_2 ... a_k$-blind if

$$\forall p \in \delta(q, a_1) \quad \exists p' \in \delta(q, a_1), v \in \Sigma^* :$$

$$\delta(p, a_2 ... a_k v) \cap F = \phi, \ \delta(p', a_2 ... a_k v) \cap F \neq \phi$$

a state is $k$-blind if there exists $a_1 a_2 ... a_k$ such that it is $a_1 a_2 ... a_k$-blind

# State-blindness is computable

Recall that a string w is q-blind if it is not possible in state q to find a deterministic choice given w as the look-ahead.

Define $B_q = \{ w \mid w \text{ is q-blind} \}$

Theorem: Let M be a DFA with n states, and over an alphabet of size m, and let q be a state of M. The language $B_q$ is regular, and there is a DFA with at most $(4^n + 1)^m$ that accepts $B_q$.

# A Characterization for Unambiguous NFA

*Theorem.*

An unambiguous NFA has a $k$-delegator iff none of its states are $k$-blind.

Consequently, an unambiguous NFA has a delegator iff all its states have finite blindness.

# Problem 1 for unambiguous NFA's

Theorem: Let k be a fixed integer. There is a polynomial time algorithm that given an unambiguous NFA M determines if M has a k-delegator.

Proof (sketch) The problem can be reduced to the problem of containment problem for unambiguous NFA's.

From a result of Hunt and Stearns, this problem is known to be solvable in polynomial time.

# Delegation for Unambiguous NFA

*Theorem*(*s*).

| | $(P_1)$ | $(P_2)$ | $(P_3)$ |
|---|---|---|---|
| unambiguous NFA | P | co-NP | PSPACE |

The proof of co-NP and PSPACE upper-bounds for P2 and P3 are similar to that of Problem 1.

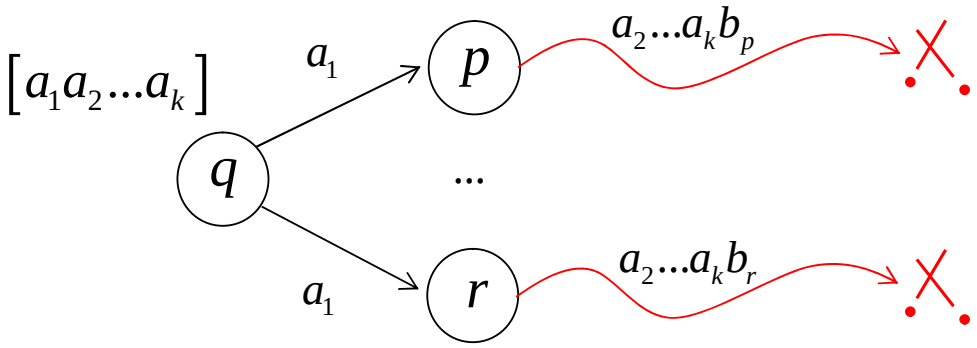
# Arbitrary NFA

All 3 problems are significantly harder for general NFA:

- a delegator for a (trim) unambiguous NFA must use all its states: local properties (blindness) have a global impact;

- containment and equivalence problems are decidable in polynomial time for unamb. NFA.

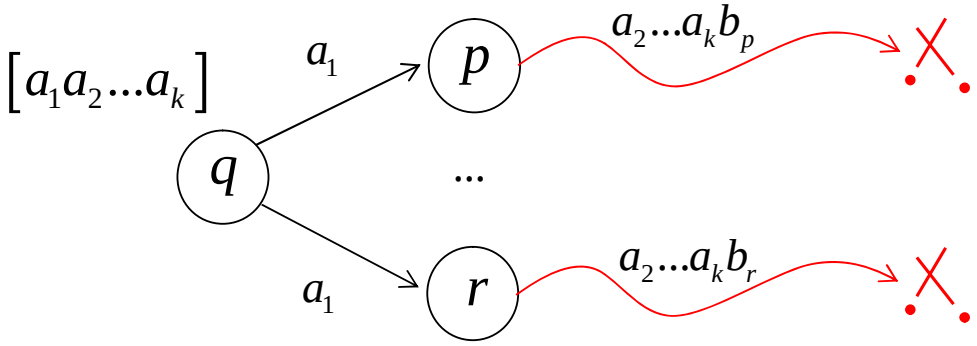
# Arbitrary NFA: Forbidden Words




$q$ is $a_1a_2...a_k$-forbidden if one of the following two conditions is satisfied recursively (intuitive def.) :

1. $q$ is $a_1a_2...a_k$-blind;
2. for every state $p \in \delta(q, a_1)$ there exists $b_p \in \sum$ such that $p$ is $a_2...a_kb_p$-forbidden.

# Arbitrary NFA: Forbidden Words




$q$ is $a_1a_2...a_k$-forbidden if one of the following two conditions is satisfied recursively (intuitive def.) :

1. $q$ is $a_1a_2...a_k$-blind;
2. for every state $p \in \delta(q, a_1)$ there exists $b_p \in \sum$ such that $p$ is $a_2...a_kb_p$-forbidden.

# Arbitrary NFA: Delegation Characterization

Notation: $F_q$ is the set of all forbidden words for $q$.

*Theorem.*

An NFA $M = (Q, \Sigma, \delta, q_0, F)$ has a $k$-delegator iff

$$F_{q_0} \cap pref_k(L_M) = \phi.$$

Consequently, an NFA has a delegator iff $F_{q_0}$ is finite.

# Delegation for Unambiguous NFA

*Theorem*(*s*).

| | $(P_1)$ | $(P_2)$ | $(P_3)$ |
|---|---|---|---|
| general NFA | PSPACE-complete | PSPACE-hard | ? |

# Wrap-up

- NFA delegation is a finite-state model used in web service applications, task scheduling, NFA simulation, measure of nondeterminism, etc.

- NFA delegation is a machine property and its computational complexity is machine dependent:

| | $(P_1)$ | $(P_2)$ | $(P_3)$ |
|---|---|---|---|
| unambiguous | P | co-NP | PSPACE |
| general | PSPACE-complete | PSPACE-hard | ? |

don't have lowerbounds

(previous work: for $k = 1$, $P_1$ in the general case : EXPTIME)

# Direction for future work

Main open problem:

- Investigate complexity matters for other families of NFA. For example, study NFA that are a shuffle product of DFA.
- Is delegation decidable for arbitrary NFA? Study the nature of $F_{q_0}$ .

Other Questions:

- The complexity result for Problem 1 (general case) was proven for a 4-letter alphabet. Can the PSPACE-completeness proof be extended to smaller alphabets?
- Is problem 2 complete for co-NP for unambiguous NFA? Is problem 3 complete for PSPACE for unambiguous NFA?

# References

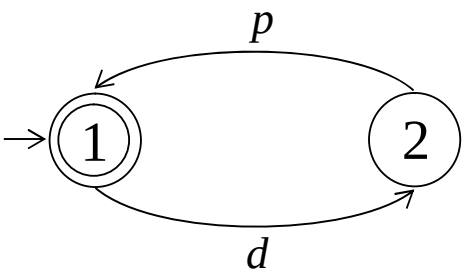

**Service-oriented computing:**

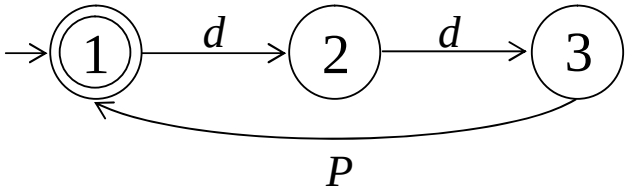
- R. Hull and J. Su. *Tools for Design of Composite Web Services*. In SIGMOD 2004, pp. 958-961 (2004)
- M. Mecella and G. D. Giacomo. *Service Composition: Technologies, Methods and Tools for Synthesis and Orchestration of Composite Services and Processes*. Tutorial in ICSOC 2004.
- D. Berardi, D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Mecella. *Automatic Composition of e-Services that Export their Behaviour*. In ICSOC 2003, LNCS 2910, pp. 43-58, 2003.

**Finite state models for e-services:**

- C. E. Gerede, O. H. Ibarra, B. Ravikumar, and J. Su. *On-line and Ad-hoc Minimum Cost Delegation in e-Service Composition*. In IEEE SCC, pp. 103-112, 2005.
- Z. Dang, O. H. Ibarra, and J. Su. *Composability of Infinite-State Activity Automata*. In ISAAC 2004, LNCS 3341, pp. 377-388.
- C. E. Gerede, R. Hull, O. H. Ibarra, and J. Su. *Automated Composition of e-Services: Lookaheads*. In ISOC 2004, pp. 252-262.

# Design Methodology - Example



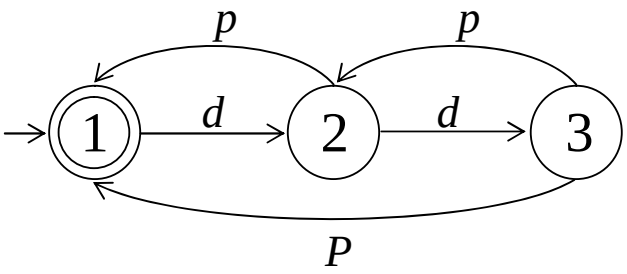




$eS_1$: downloads and processes files sequentially

$eS_2$: downloads 2 files at a time and processes them in parallel

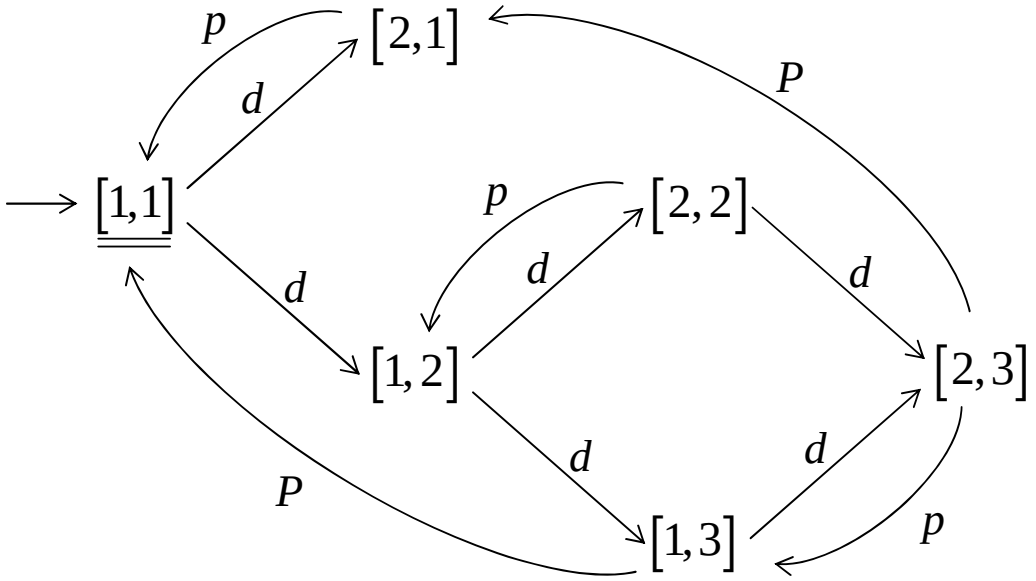
---

S - application specification: download and process, having no more than 2 files in the system at any time



# Design Methodology – Step 1

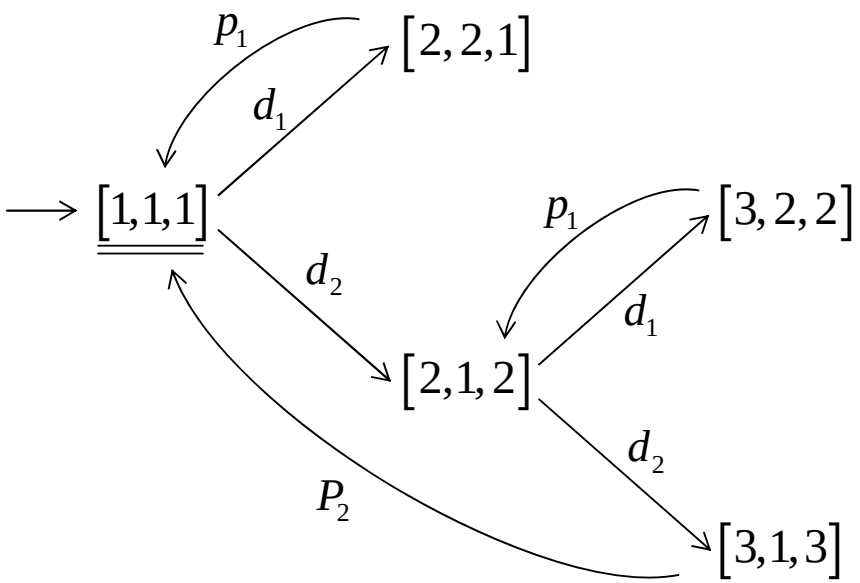
construct $eS_1 \# eS_2$ :




it represents the shuffle behavior of $eS_1$ and $eS_2$

# Design Methodology – Step 2

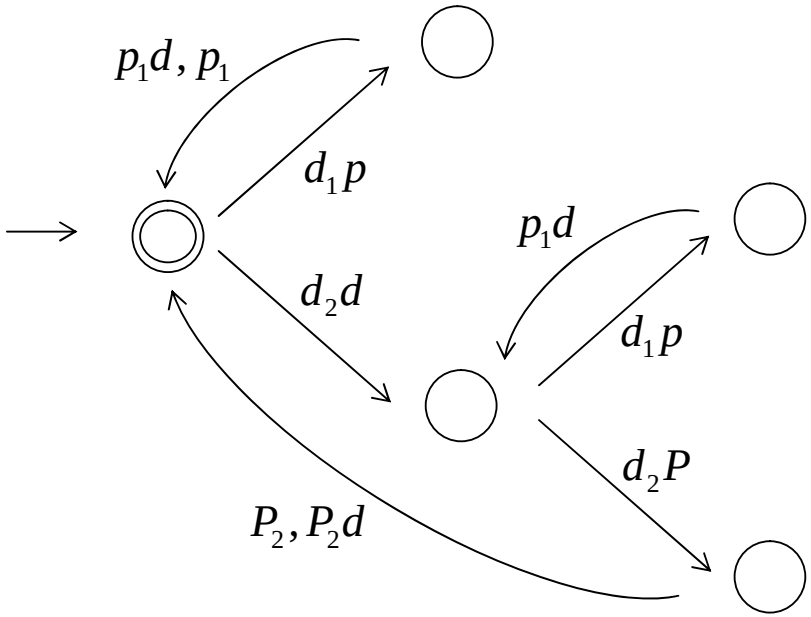
construct $S \cap eS_1 \# eS_2$ :

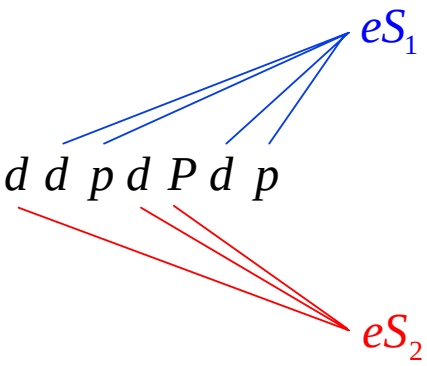



it represents the nondeterministic behavior of the application ($S$)

# Design Methodology – Step 3

find a delegator :




it represents the design of $S$