Computational Linguistics:

Part 1: Finite-State Automata

Pierre Lison

Language Technology Lab
DFKI GmbH, Saarbrücken
http://talkingrobots.dfki.de

Welcome to the course

- Welcome to the **Computational Linguistics** course, Summer Semester 2010 edition!
- Hope you will find it interesting & enjoyable :-)
- As you know, the course timetable is as follows:
  - Lecture every Monday, 14 - 16 hr
  - Exercise session every Thursday, 14 - 16 hr
- The course will be given by five lecturers: Bernd Kiefer, Hans-Ulrich Krieger, Dietrich Klakow, Andreas Eisele and myself (Pierre Lison)
- Successful completion of the course will grant you 6 credit points
- Also keep in mind that this course is one of the *core courses* for the “Computational Linguistics” specialization of the M.Sc. program

Infos & contacts

- The website for the course is up and running:

  http://www.dfki.de/~plison/lectures/compling2010/index.html

- You will find there a detailed schedule of the course, as well as the necessary resources (slides, assignments, solutions)

- A mailing list has also been created. Please subscribe to stay informed about the course details over the semester:

  http://ml.coli.uni-saarland.de/cgi-bin/mailman/listinfo/compling2010

- As I‘m officially responsible for the course, so if you have any general question or request about the course, don‘t hesitate to contact me: plison@dfki.de

- Or drop by at my office (preferably with a short email beforehand):
  - Building D 3.1 (DFKI Altbau), room C 1.11 (floor +1).

Evaluation

- Assignments:
  - Assignments will be given after every lecture, to do before the following exercise session
  - Each completed assignment is to be sent via email to the lecturer. The exact deadline for submitting your assignment will be provided in due time by the corresponding lecturers
  - In order to be able to register for the exam, you need to pass at least 80% of the required assignments
- The final evaluation will consist of a usual 90-minutes written exam
  - A Probeklausur will be offered one week before the exam to give you an idea of what you should expect
- The final grade will be a composite of the assignment grades, and of course the exam (exact percentage still unknown)

Learning materials

- The course content will be mostly provided via PDF slides available on the course website
- Additional readings might also be provided during the lecture
- The two following textbooks are warmly recommended:
  - D. Jurafsky and J. H. Martin: *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, Prentice-Hall, 2009.
  - C. Manning and H. Schütze: *Foundations of Statistical Natural Language Processing*, MIT Press, 1999.
- They are both available at the CoLi library

Goal of the course

- The course focuses on the **core algorithms** used in computational linguistics
  - How they work, on which representations they operate
  - Formal properties, computational complexity
  - Pros and cons of each approach
  - Use in practical applications
- **Learning objectives:**
  - gain a reasonable understanding of how each of these algorithms work
    - Both in theory and in practice (hands-on exercises)
  - And (perhaps more importantly): understand the relative *merits* and *limitations* of these algorithms
    - The goal is to understand which algorithm(s) should be used to solve a particular problem!

Main topics

- In particular, we will review the following topics this semester:
  - Finite-State Automata and their algorithms (Pierre)
  - Chart-based parsing & generation (Bernd)
  - Unification-based parsing & generation (Bernd)
  - Ontologies (Uli)
  - Maximum Entropy Models (Dietrich)
  - Conditional Random Fields (Dietrich)
  - Alignment algorithms (Andreas)
  - Shallow matching algorithms for strings & trees (Pierre)

Course schedule

- **Mon 19.4**: Seminar: Lison - Introduction to the course, Finite-State Automata
- **Thu 22.4**: Exercises: Lison - Finite-State Automata
- **Mon 26.4**: Seminar: Kiefer - Chart-based parsing and generation
- **Thu 29.4**: Exercises: Kiefer - Chart-based parsing and generation
- **Mon 03.5**: Seminar: Krieger - Ontologies
- **Thu 06.5**: Exercises: Krieger - Ontologies
- **Mon 10.05**: Seminar: Kiefer - Unification-based parsing and generation
- *Thu 13.05: PUBLIC HOLIDAYS (Himmelfahrt)*
- **Mon 17.05**: Seminar: Klakow - Maximum Entropy Models
- **Thu 20.05**: Exercises: Kiefer - Unification-based parsing and generation
- *Mon 24.05: PUBLIC HOLIDAYS (Pfingstmontag)*
- **Thu 27.05**: Exercises: Klakow - Maximum Entropy Models
- **Mon 31.05**: Seminar: Klakow - Conditional Random Fields
- *Thu 03.06: PUBLIC HOLIDAYS (Fronleichnam)*
- **Mon 07.06**: Seminar: Eisele - Alignment algorithms
- **Thu 10.06**: Exercises: Klakow - Conditional Random Fields
- **Mon 14.06**: Seminar: Lison: Shallow matching algorithms (strings)
- **Thu 17.06**: Exercises: Eisele - Alignment algorithms
- **Mon 21.06**: Seminar: Lison: Shallow matching algorithms (trees)
- **Thu 24.06**: Exercises: Lison: Shallow matching algorithms
- **Mon 28.06**: Question hour
- **Thu 01.07**: Test exam
- *Mon 05.07: NO SEMINAR*
- **Thu 08.07**: Final exam (90 minutes writing time)

Lecture 1: Finite-State Automata

(based on slides from Annette Frank)

Overview of the lecture

- Background
  - Chomsky hierarchy of languages
  - Basic definitions, generic operations on languages
- Generalities about Finite-State Automata (FSA)
  - Regular languages, regular expressions and FSAs
  - Constructing a FSA from a regular expression
  - Non-deterministic FSAs
- Optimization algorithms for FSAs
  - *Determinization* of a FSA via subset construction
  - *Minimization* of a FSA: equivalence classes, Brzozowski‘s algorithm
- Applications of FSAs & extensions to finite-state transducers
- Conclusions, exercises

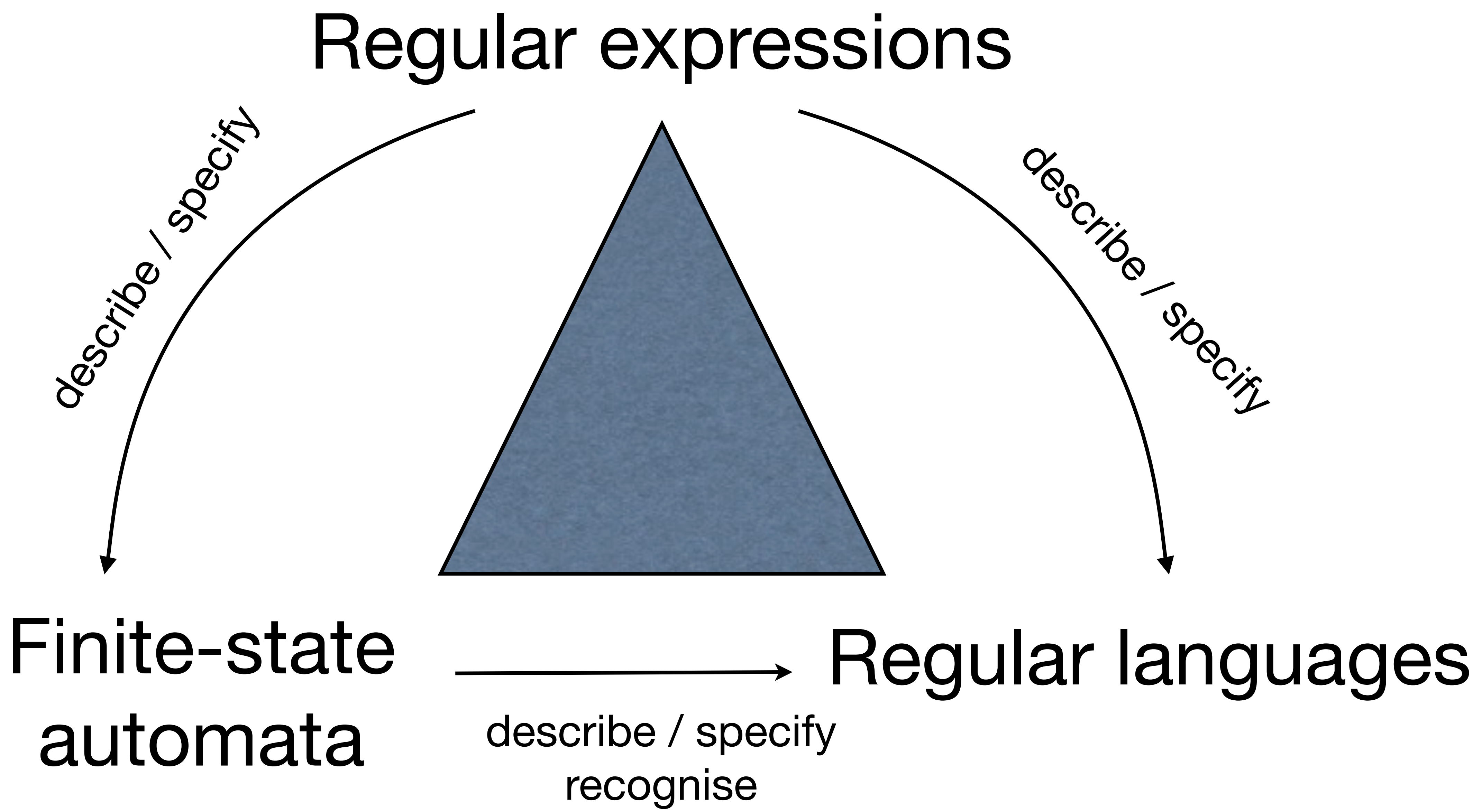
Overview of the lecture

- **Background**
  - **Chomsky hierarchy of languages**
  - **Basic definitions, generic operations on languages**
- Generalities about Finite-State Automata (FSA)
  - Regular languages, regular expressions and FSAs
  - Constructing a FSA from a regular expression
  - Non-deterministic FSAs
- Optimization algorithms for FSAs
  - *Determinization* of a FSA via subset construction
  - *Minimization* of a FSA: equivalence classes, Brzozowski's algorithm
- Applications of FSAs & extensions to finite-state transducers
- Conclusions, exercises

Finite-state automata: what for?

Chomsky Hierarchy of Languages	Hierarchy of Grammars & Automata
Regular languages (type-3)	Regular PS grammar Finite-state automata
Context-free languages (type-2)	Context-free PS grammar Push-down automata
Context-sensitive languages (type-1)	Tree adjoining grammars Linear bounded automata
Unconstrained languages (type-0)	General PS grammars Turing machine

More expressivity
Less computational efficiency



Regular expressions
describe / specify
describe / specify
Finite-state automata
Regular languages
describe / specify
recognise

- **Alphabet** Σ: finite set of symbols
- **String**: sequence $x_1 \ldots x_n$ of symbols x_i taken from the alphabet Σ
- **Language** over Σ: set of strings that can be generated from Σ
  - Sigma star Σ^*: set of all possible strings over the alphabet Σ,
    - For instance, if $\Sigma = \{a,b\}$, then $\Sigma^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$
  - Sigma plus $\Sigma+$ removes the empty element: $\Sigma^+ = \Sigma^* - \{\varepsilon\}$
  - Special language $\varnothing = \{\}$, called the empty language
    - Attention: note the difference with $\{\varepsilon\}$: language with one element, the empty string
- **Formal language**: a subset of Σ^*

Some basic definitions (3)

- A **formal grammar** is a tuple G= < Σ, Φ, S, R >, where
  - Σ is an alphabet of terminal symbols
  - Φ is an alphabet of non-terminal symbols
  - S is the start symbol
  - R is a finite set of rules, with $R \subseteq \Gamma^{+} \times \Gamma^{*}$.
    - Γ is the union of terminal and non-terminal symbols: $\Gamma = \Sigma \cup \Phi$
    - Each rule $\in$ R is of the form $\alpha \rightarrow \beta$

- **Derivation**:
  - Assume a grammar G= $< \Sigma, \Phi, S, R >$ and two arbitrary strings u and $v \in \Gamma^* = (\Sigma \cup \Phi)^*$
  - A direct derivation $u \Rightarrow_G v$ holds iff there exists $s_1, s_2 \in \Gamma^*$ such that $u = (s_1\ \alpha\ s_2)$ and $v = (s_1\ \beta\ s_2)$ and there is a rule $\alpha \rightarrow \beta$ in R
  - A general derivation $u \Rightarrow_{G^*} v$ holds iff either $u = v$ or there exists a string $z \in \Gamma^*$ such that $u \Rightarrow_{G^*} z$ and $z \Rightarrow_G v$
- The **language** L(G) generated by a grammar G is defined as the set of strings $w \subseteq \Sigma^*$ that can be derived from the start symbol S according to the grammar G
  - In other words: $L(G) = \{w : S \Rightarrow_{G^*} w \wedge w \in \Sigma^*\}$

Some basic definitions (5)

- Basic operation on strings: *concatenation* •
  - Assume two strings a and b, defined by $a = x_i \ldots x_m$ and $b = x_{m+1} \ldots x_n$
  - Then the concatenation $a \bullet b = x_i \ldots x_m \, x_{m+1} \ldots x_n$
  - Concatenation is associative, but not commutative
  - ε is the identity element: $a \bullet \varepsilon = \varepsilon \bullet a = a$

Chomsky Hierarchy of grammars

- Classification of languages generated by formal grammars
  - A language is of type *i* (i = 0,1,2,3) iff it is generated by a *type-i* grammar
  - Classification according to increasingly restricted types of production rules:
    - **L-type-0 ⊃ L-type1 ⊃ L-type-2 ⊃ L-type-3**
  - Every grammar generates a unique language, but a language can be generated by several different grammars.
- Two grammars are
  - (Weakly) equivalent if they generate the same string language
  - Strongly equivalentif they generate both the same string language and the same tree language

- **Type - 0 languages: general phrase structure grammars**
  - No restrictions on the form of production rules: arbitrary strings on both the left-hand and right-hand side of rules
  - A grammar G= < Σ, Φ, S, R > generates a language L-type-0 iff:
    - All rules R are of the form $\alpha \rightarrow \beta$, where $\alpha \in \Gamma^{+}$ and $\beta \in \Gamma^{*}$
    - In other words, the LHS must be a nonempty sequence of non-terminal or terminal symbols
    - And RHS a (possibly empty) sequence of non-terminal or terminal symbols
  - Example of type-0 grammar:
    - G = < {S, A, B, C, D, E}, {a}, S, R >, with the following production rules:

S → ACaB	CB → E	aE → Ea
Ca → aaC	aD → Da	AE → ε
CB → DB	AD → AC	

    - Question: what is the language generated by G? $L(G) = \{a^{2^n} | n \geq 1\}$

- **Type-1 languages: context-sensitive grammars**
  - a grammar G= < Σ, Φ, S, R > generates a language L-type-1 iff
    - all rules are of the form $\alpha A \Upsilon \rightarrow \alpha\beta\Upsilon$, where A is a non-terminal ($\in \Phi$) and $\alpha,\beta,\Upsilon \in \Gamma^*$
    - In other words, the LHS is a non-empty sequence of NT and T symbols, with at least one NT symbol
    - The RHS is a non-empty sequence of NT or T symbols
  - Example:
    - G = < {S,B,C}, {a,b,c}, S, R >, with the following production rules:

S → a S B C	a B → a b	
S → a B C	b B → b b	
C B→B C	b C → b c	c C → c c

    - Question: what is the language generated by G? $L(G) = \{a^n b^n c^n | n \geq 1\}$

- **Type-2 languages: context-free grammars**
  - a grammar G= < Σ, Φ, S, R > generates a language L-type-2 iff
    - all rules are of the form A → α, where A is a non-terminal (∈ Φ) and α ∈ Γ*
    - In other words, the LHS is a single NT symbol
    - The RHS is a non-empty sequence of NT or T symbols
  - Example:
    - G = < {S, A}, {a, b}, S, R >, with the following production rules:

S → A S A	A → a
S → b	

    - Question: what is the language generated by G? $L(G) = \{a^n b a^n | n \geq 1\}$

- **Type-3 languages: regular or finite-state grammars**
  - a grammar G= < Σ, Φ, S, R > generates a language L-type-2 iff
    - all rules are of the form A → wB or A →w , where A, B are non-terminals (∈ Φ) and w ∈ Σ*
    - In other words, the LHS is a single NT symbol, and the RHS is a possibly empty sequence of T symbols, optionally followed by a single NT symbol
    - The definition above is right linear. Left linear grammars have rules of the form A → Bw, and function similarly
  - Example:
    - G = < {S, A, B}, {a, b}, S, R >, with the following production rules:

S → a A	B → b B	a → b b B
A → a A	B → b	

    - Question: what is the language generated by G? $L(G) = \{aa^*bbb^*b\}$

Operations on languages

- Typical set-theoretic operations on languages
  - Union: $L_1 \cup L_2 = \{ w : w \in L_1 \text{ or } w \in L_2 \}$
  - Intersection: $L_1 \cap L_2 = \{ w : w \in L_1 \text{ and } w \in L_2 \}$
  - Difference: $L_1 - L_2 = \{ w : w \in L_1 \text{ and } w \notin L_2 \}$
  - Complement of $L \subseteq \Sigma^*$ wrt. Σ^*: $L^{-} = \Sigma^* - L$
- Language-theoretic operations on languages
  - Concatenation: $L_1L_2 = \{w_1w_2 : w_1 \in L_1 \text{ and } w_2 \in L_2\}$
  - Iteration: $L^0 = \{\varepsilon\}$, $L^1 = L$, $L^2 = LL$, ... $L^* = \bigcup_{i \geq 0} L^i$, $L^+ = \bigcup_{i > 0} L^i$
  - Mirror image: $L^{-1} = \{w^{-1} : w \in L\}$
- Union, concatenation and Kleene star are called regular operations
- Regular sets/languages: languages that are defined by the regular operations: concatenation $(\cdot)$, union $(\cup)$ and kleene star $(*)$
- Regular languages are *closed* under *concatenation, union, kleene star, intersection and complementation*

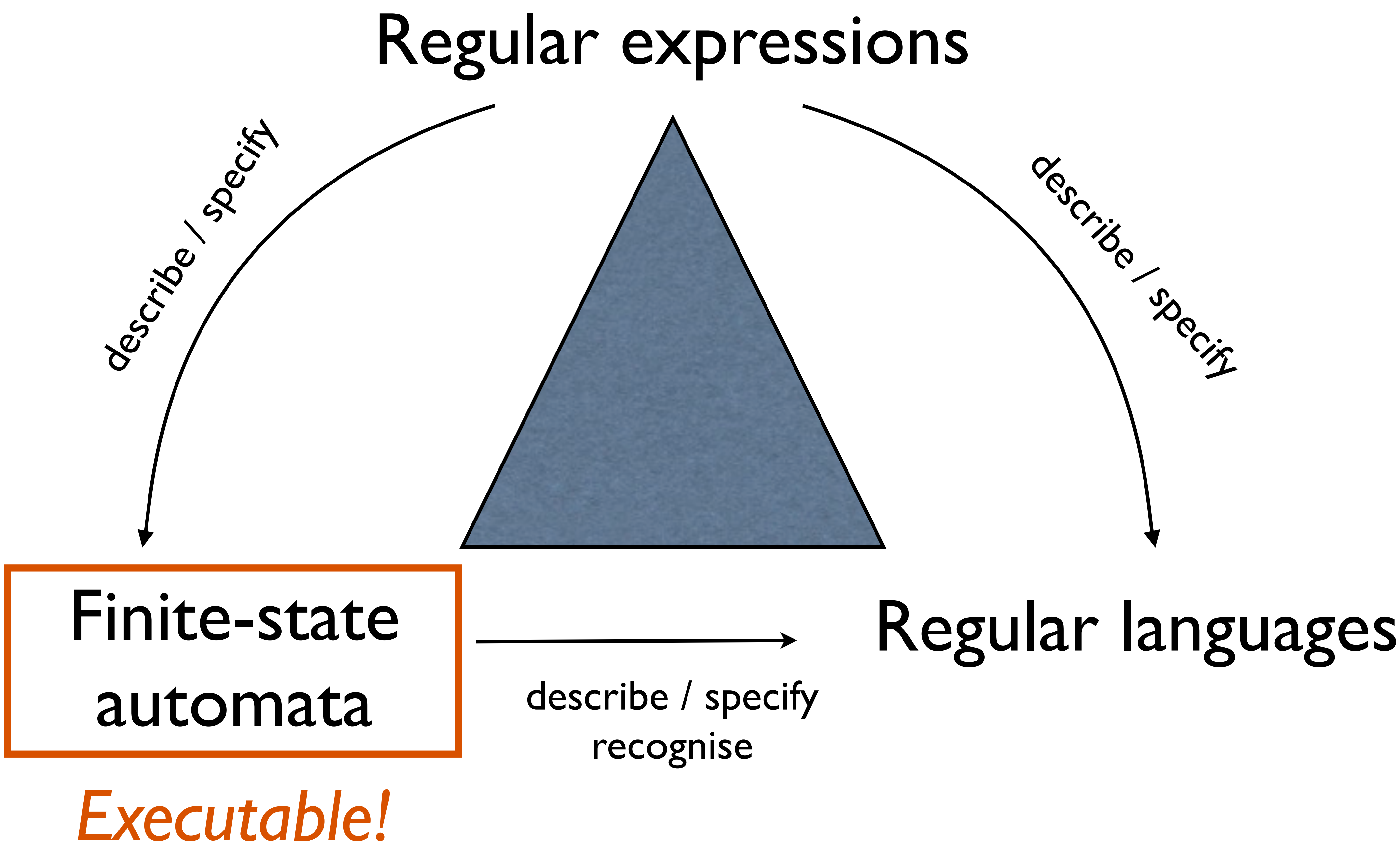
Overview of the lecture

- Background
  - Chomsky hierarchy of languages
  - Basic definitions, generic operations on languages
- **Generalities about Finite-State Automata (FSA)**
  - Regular languages, regular expressions and FSAs
  - Constructing a FSA from a regular expression
  - Non-deterministic FSAs
- Optimization algorithms for FSAs
  - *Determinization* of a FSA via subset construction
  - *Minimization* of a FSA: equivalence classes, Brzozowski's algorithm
- Applications of FSAs & extensions to finite-state transducers
- Conclusions, exercises

Overview of the lecture

- Background
  - Chomsky hierarchy of languages
  - Basic definitions, generic operations on languages
- **Generalities about Finite-State Automata (FSA)**
  - **Regular languages, regular expressions and FSAs**
  - Constructing a FSA from a regular expression
  - Non-deterministic FSAs
- Optimization algorithms for FSAs
  - *Determinization* of a FSA via subset construction
  - *Minimization* of a FSA: equivalence classes, Brzozowski‘s algorithm
- Applications of FSAs & extensions to finite-state transducers
- Conclusions, exercises

FSAs and regular expressions

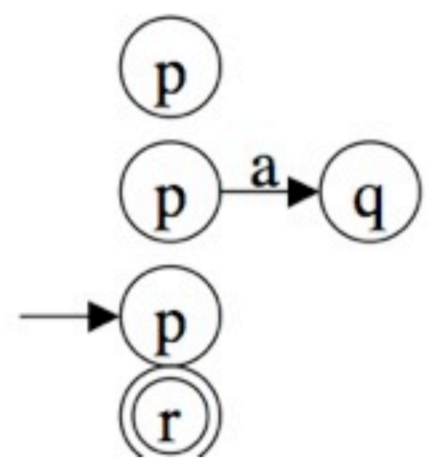


Regular languages & expressions

- Regular sets/languages can be specified/defined by regular expressions
  Given a set of terminal symbols Σ, the following are regular expressions
  - ε is a regular expression
  - For every $a \in \Sigma$, a is a regular expression
  - If R is a regular expression, then R* is a regular expression
  - If Q,R are regular expressions, then QR $(Q \cdot R)$ and $Q \cup R$ are regular expressions
- Every regular expression denotes a regular language
  - $L(\varepsilon) = \{\varepsilon\}$
  - $L(a) = \{a\}$ for all $a \in \Sigma$
  - $L(\alpha\beta) = L(\alpha)L(\beta)$
  - $L(\alpha \cup \beta) = L(\alpha) \cup L(\beta)$
  - $L(\alpha^*) = L(\alpha)^*$

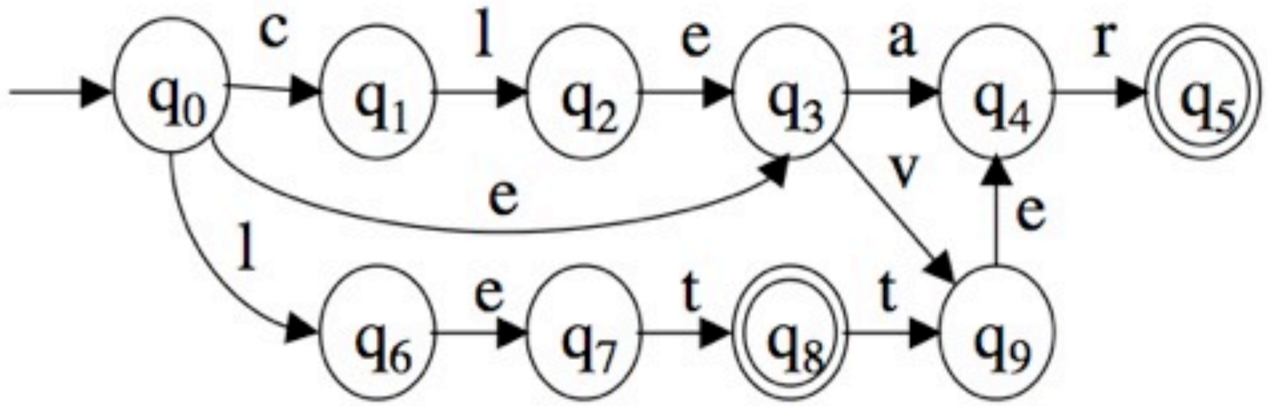
Finite-state automata (FSA)

- Grammars: generate (or recognize) languages
  Automata: recognize (or generate) languages
- Finite-state automata recognize regular languages
- A finite automaton (FA) is a tuple $A = <\Phi, \Sigma, \delta, q_0, F>$
  - Φ a finite non-empty set of states
  - $\mathbf{\Sigma}$ a finite alphabet of input letters
  - δ a transition function $\Phi \times \mathbf{\Sigma} \rightarrow \Phi$
  - $q_0 \in \Phi$ the initial state
  - $F \subseteq \Phi$ the set of final (accepting) states
- Transition grap(ςs (diagrams):
  - states: circles — $p \in \Phi$
  - transitions: directed arcs between circles — $\delta(p, a) = q$
  - initial state — $p = q_0$
  - final state — $r \subseteq F$



FSA transition graphs

- Transition graph



$S = q_0 \quad F = \{q_5, q_8\}$

Transition function $\delta: \Phi \times \Sigma \rightarrow \Phi$

$\delta(q_0,c)=q_1$

$\delta(q_0,e)=q_3$

$\delta(q_0,l)=q_6$

$\delta(q_1,l)=q_2$

$\delta(q_2,e)=q_3$

$\delta(q_3,a)=q_4$

$\delta(q_3,v)=q_9$

$\delta(q_4,r)=q_5$

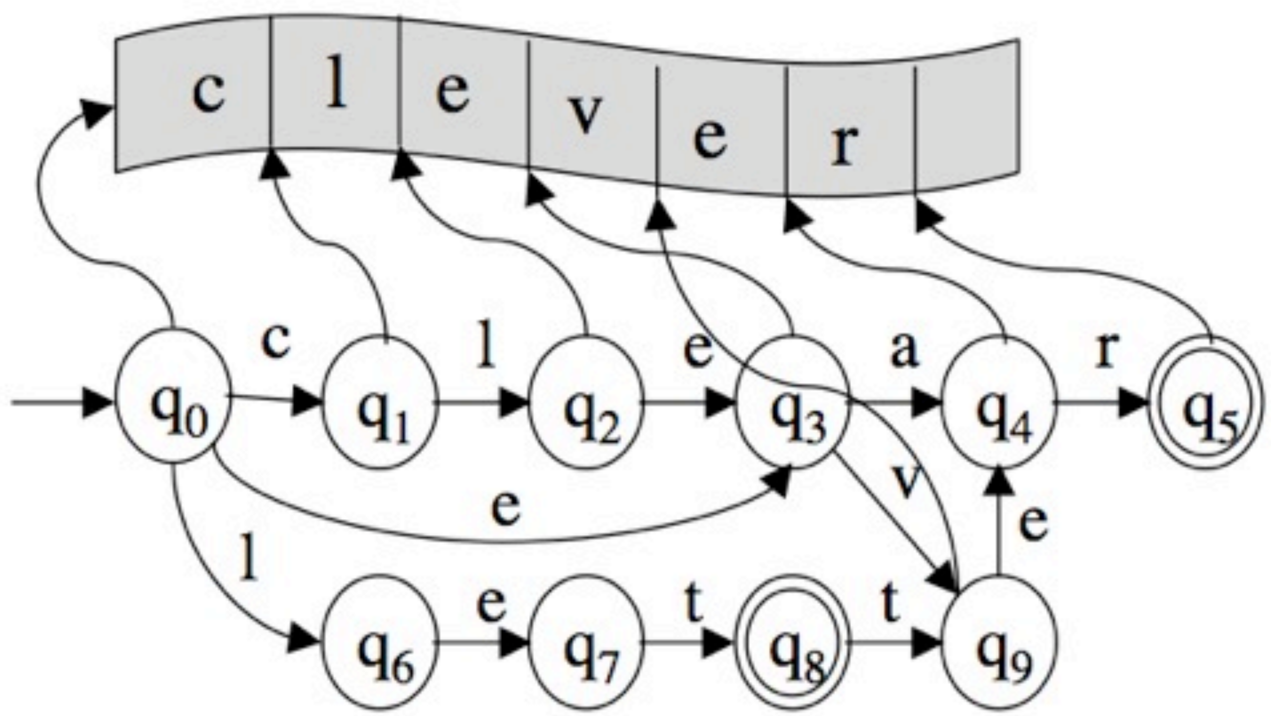
$\delta(q_6,e)=q_7$

$\delta(q_7,t)=q_8$

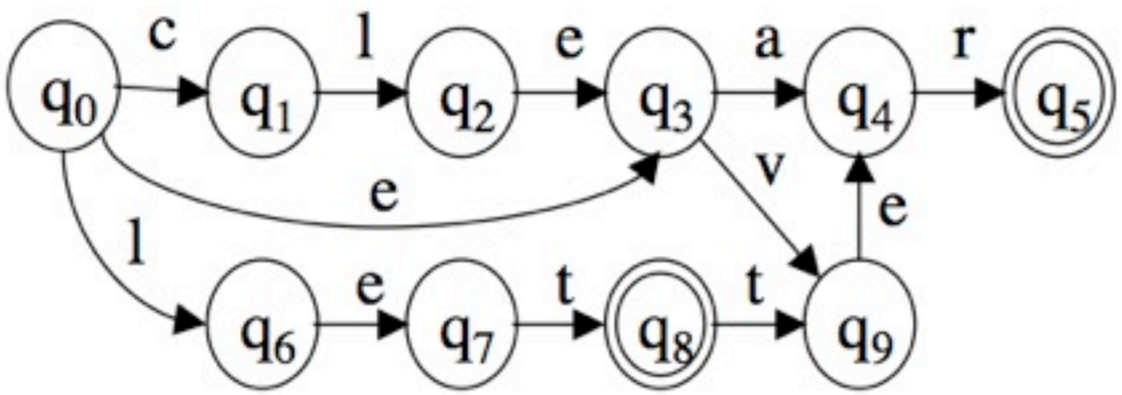
$\delta(q_8,t)=q_9$

$\delta(q_9,e)=q_4$

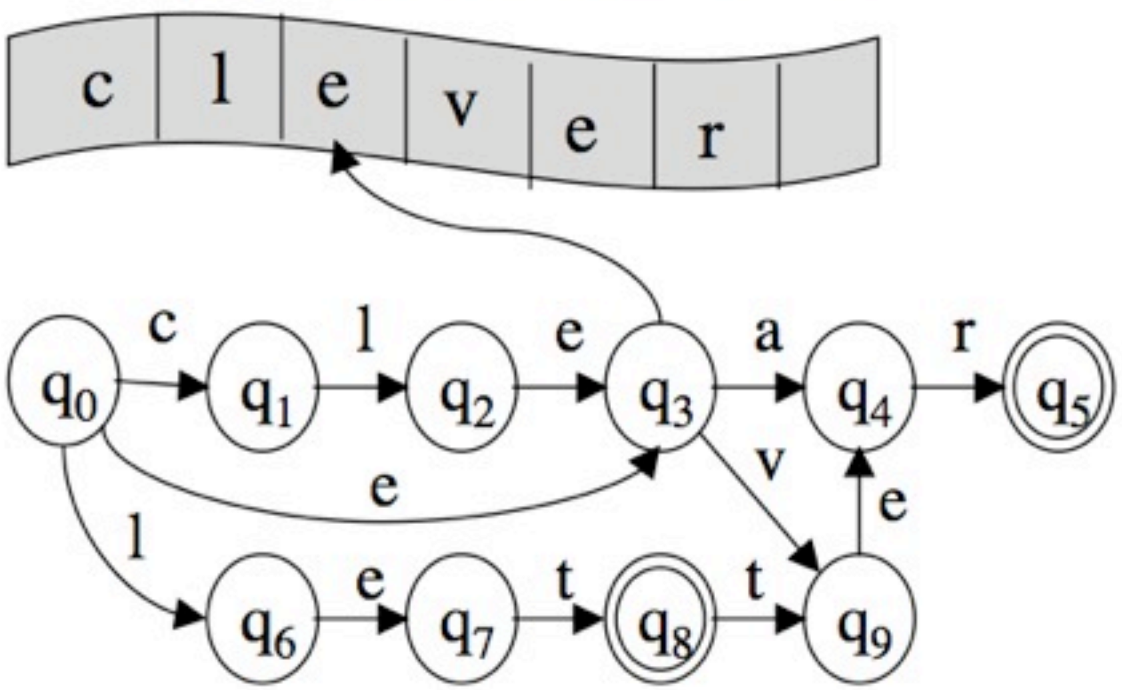
- Traversal of an FSA
  = *Computation with an FSA*



- Transition graph



- Traversal of an FSA
  = *Computation with an FSA*



State diagram

δ	a	c	e	l	r	t	v
q_0	0	q_1	q_3	q_6	0	0	0
q_1	0	0	0	q_2	0	0	0
q_2	0	0	q_3	0	0	0	0
q_3	q_4	0	0	0	0	0	q_9
q_4	0	0	0	0	q_5	0	0
q_5	0	0	0	0	0	0	0
q_6	0	0	q_7	0	0	0	0
q_7	0	0	0	0	0	q_8	0
q_8	0	0	0	0	0	q_9	0
q_9	0	0	q_4	0	0	0	0

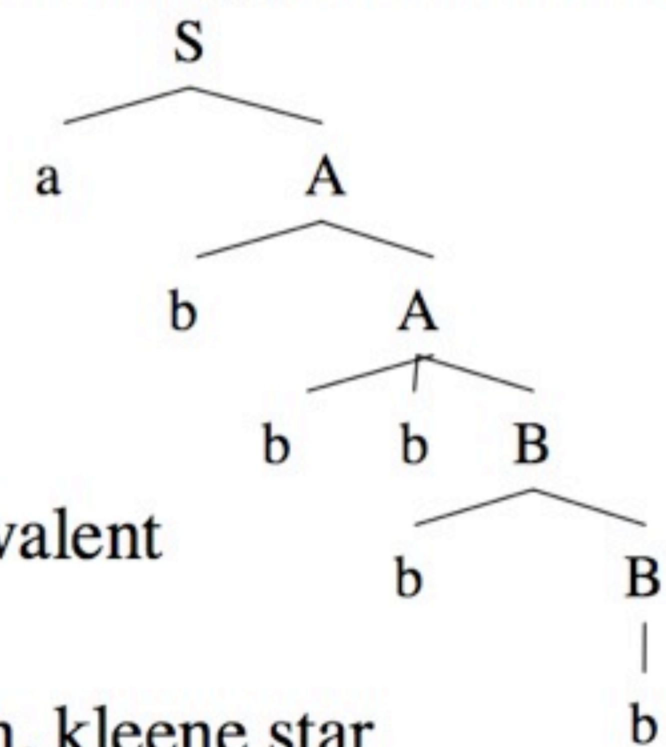
FSAs can be used for
- acceptance (recognition)
- generation

Traversal & acceptance

- *Traversal* of a (deterministic) FSA
  - FSA *traversal* is defined by states and transitions of A, relative to an input string $w \in \Sigma^*$
  - A *configuration* of A is defined by the current state and the unread part of the input string: (q, w_i), with $q \in \Phi$, w_i suffix of w
  - A *transition*: a binary relation between configurations
    $(q,w_i) \vdash_A (q',w_{i+1})$ iff $w_i = zw_{i+1}$ for $z \in \Sigma$ and $\delta(q,z) = q'$
    (q,w_i) *yields* (q',w_{i+1}) in a single transition step
  - Reflexive, transitive closure of $\vdash_A$: $(q, w_i) \vdash^*_A (q', w_j)$
    (q, w_i) *yields* (q', w_j) in zero or a finite number of steps
- *Acceptance*
  - Decide whether an input string *w* is in the language L(A) defined by FSA A
  - An FSA A *accepts* a string w iff $(q_0,w) \vdash^*_A (q_f, \varepsilon)$, with q_0 initial state, $q_f \subseteq F$
  - The *language L(A) accepted by FSA A* is the *set of all strings accepted by A*
    I.e., $w \in L(A)$ iff there is some $q_f \subseteq F_A$ such that $(q_0,w) \vdash^*_A (q_f, \varepsilon)$

Regular grammars & FSAs

- A grammar $G = <\Sigma, \Phi, S, R>$ is called right linear (or regular) iff all rules R are of the form $A \rightarrow w$ or $A \rightarrow wB$, *where* $A,B \in \Phi$ and $w \in \Sigma^*$
  - $\Sigma=\{a, b\}$, $\Phi=\{S,A,B\}$, $R=\{S \rightarrow aA, A \rightarrow aA, A \rightarrow bbB, B \rightarrow bB, B \rightarrow b\}$
    $S \Rightarrow aA \Rightarrow aaA \Rightarrow aabbB \Rightarrow aabbbB \Rightarrow aabbbb$
  - The NT symbol corresponds to a state in an FSA: the future of the derivation only depends on the identity of this state or symbol and the remaining production rules.
  - *Correspondence of type-3 grammar rules with transitions in a (non-deterministic) FSA:*
    - $A \rightarrow w\,B \quad \equiv \quad \delta(A,w)=B$
    - $A \rightarrow w \quad \equiv \quad \delta(A,w)=q,\ q \in F$
  - Conversely, we can construct an FSA from the rules of a type-3 language



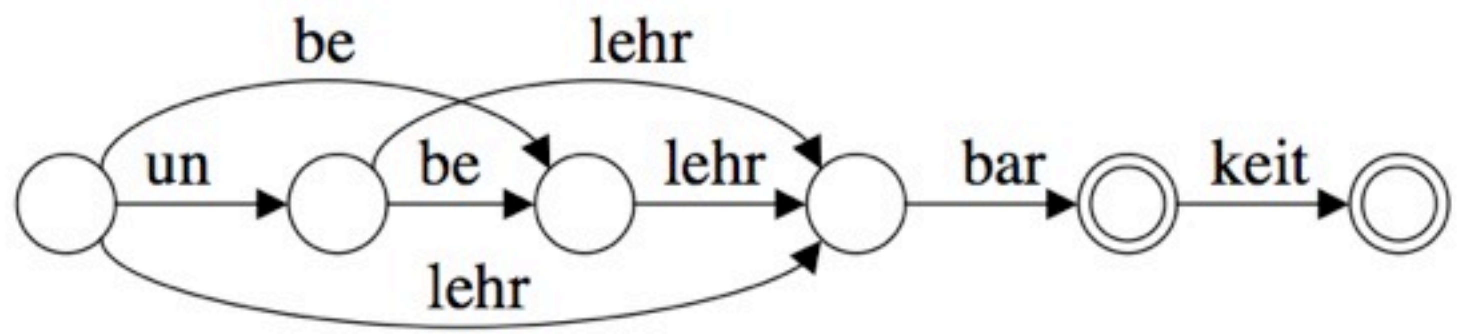
- Regular grammars and FSAs can be shown to be equivalent
- Regular grammars generate regular languages
- Regular languages are defined by concatenation, union, kleene star

Deterministic finite-state automata

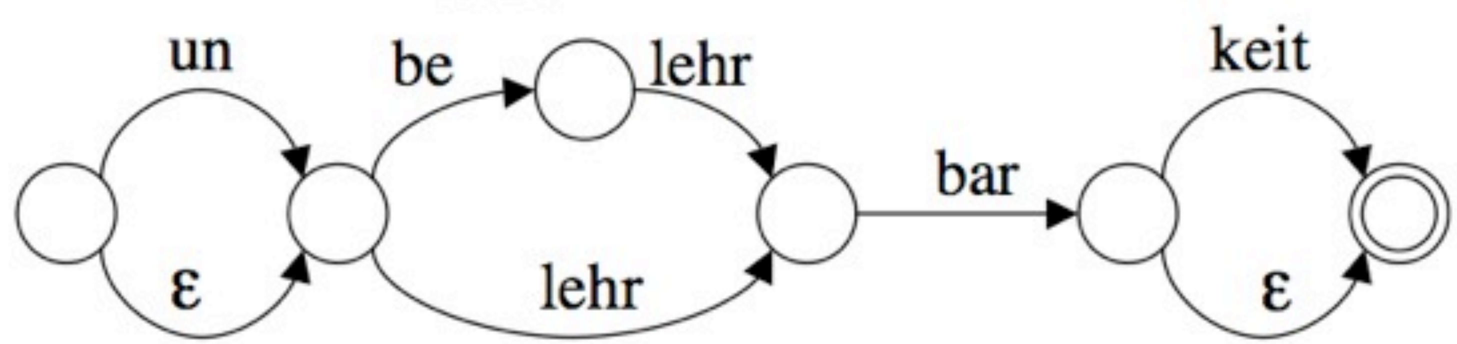
- Deterministic finite-state automata (DFSA)
  - at *each state*, there is *at most one transition* that can be taken to read the next input symbol
  - the next state (transition) is *fully determined by current configuration*
  - δ is functional (and there are no ε-transitions)
- Determinism is a useful property for an FSA to have!
  - Acceptance or rejection of an input can be computed in *linear time 0(n)* for inputs of length n
  - Especially important for processing of LARGE documents
- Appropriate problem classes for FSAs
  - Recognition and acceptance of *regular languages*, in particular *string manipulation*, regular phonological and morphological processes
  - Approximations of non-regular languages in morphology, shallow finite-state parsing, …

Multiple equivalent FSAs

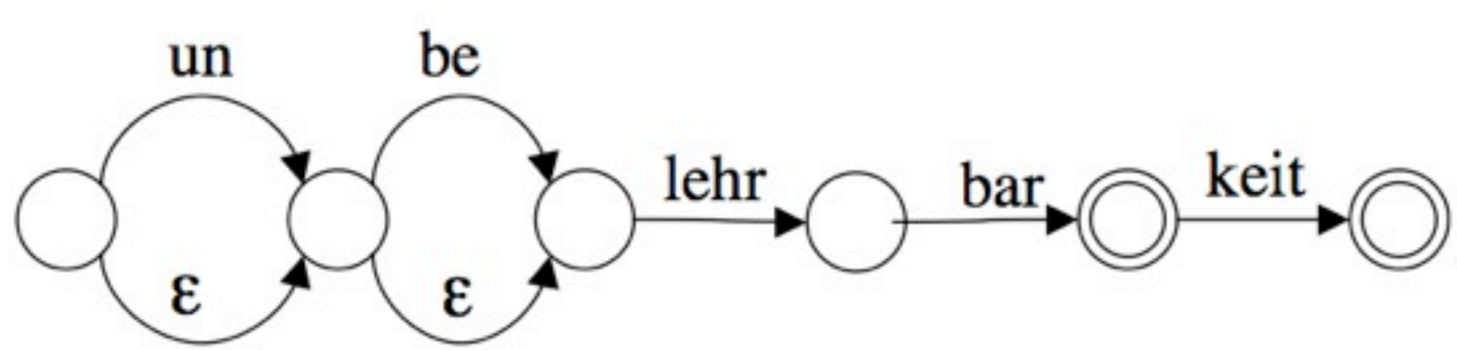
- FSA for the language L_{lehr} = { *lehrbar*, *lehrbarkeit*, *belehrbar*, *belehrbarkeit*, *unbelehrbar*, *unbelehrbarkeit*, *unlehrbar*, *unlehrbarkeit* }
- DFSA for L_{lehr}



- Regular expression and FSA for L_{lehr} : (un | ε) (be lehr | lehr) bar (keit | ε) (non-deterministic)



- Equivalent FSA (non-deterministic)

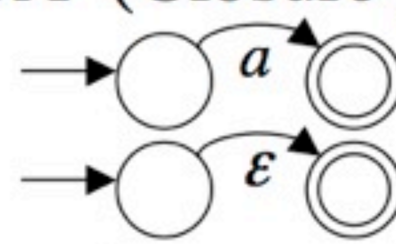


Overview of the lecture

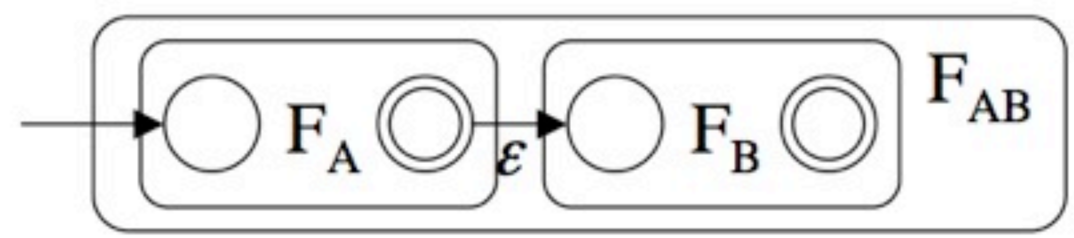
- Background
  - Chomsky hierarchy of languages
  - Basic definitions, generic operations on languages
- **Generalities about Finite-State Automata (FSA)**
  - Regular languages, regular expressions and FSAs
  - **Constructing a FSA from a regular expression**
  - Non-deterministic FSAs
- Optimization algorithms for FSAs
  - *Determinization* of a FSA via subset construction
  - *Minimization* of a FSA: equivalence classes, Brzozowski‘s algorithm
- Applications of FSAs & extensions to finite-state transducers
- Conclusions, exercises

Defining FSAs through regexps

- FSAs for even mildly complex regular languages are best constructed from regular expressions!
- Every regular expression denotes a regular language
  - $L(\varepsilon) = \{\varepsilon\}$
  - $L(a) = \{a\}$ for all $a \in \Sigma$
  - $L(\alpha\beta) = L(\alpha)L(\beta)$
  - $L(\alpha \cup \beta) = L(\alpha) \cup L(\beta)$
  - $L(\alpha^*) = L(\alpha)^*$
- Every regular expression translates to a FSA (Closure properties)
  - An FSA for a (with $L(a) = \{a\}$), $a \in \Sigma$:
  - An FSA for ε (with $L(\varepsilon) = \{\varepsilon\}$), $\varepsilon \in \Sigma$:

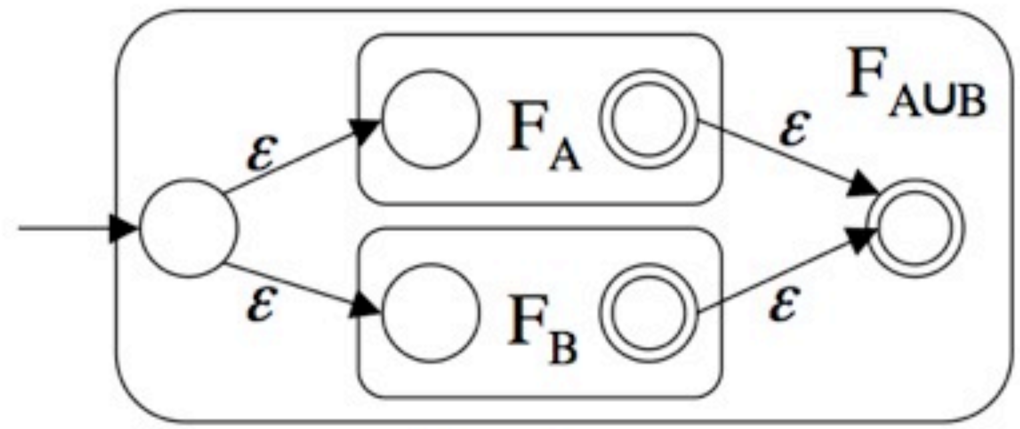


  - Concatenation of two FSAs F_A and F_B:
    - $S_{AB} = S_A$ (S initial state)
    - $F_{AB} = F_B$ (F set of final states)
    - $\delta_{AB} = \delta_A \cup \delta_B \cup \{\delta(<q_i,\varepsilon>,q_j) \mid q_i \in F_A, q_j = S_B\}$

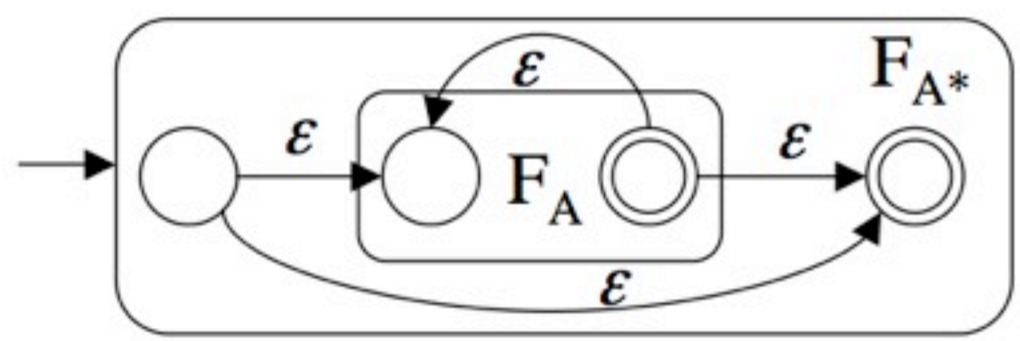


Defining FSAs through regexps

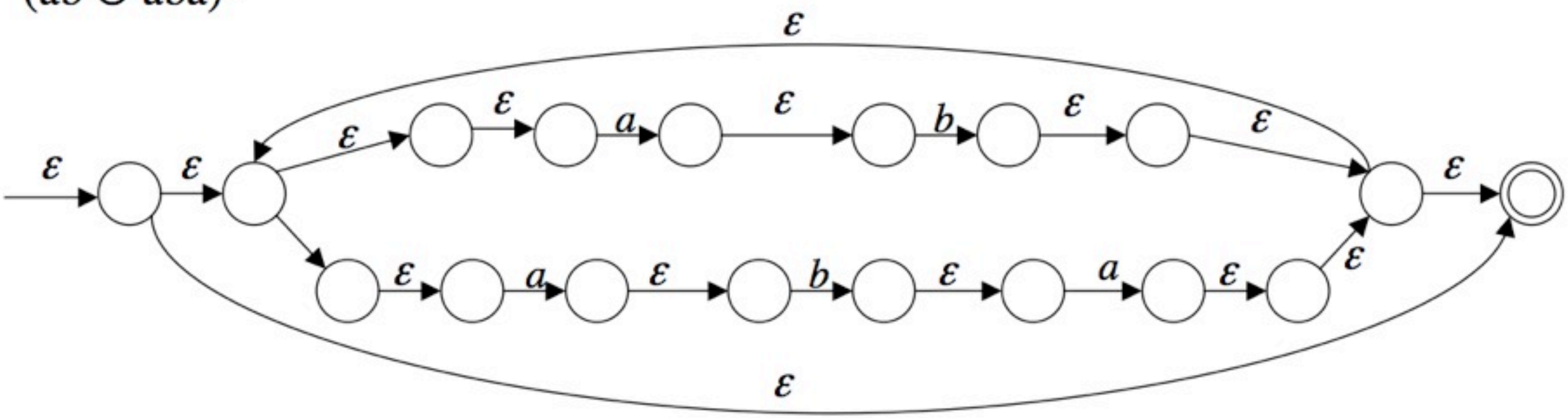
- union of two FSAs F_A and F_B:
  - $S_{AB} = s_0$ (new state)
  - $F_{AB} = \{ s_j \}$ (new state)
  - $\delta_{AB} = \delta_A \cup \delta_B$
    $\cup \{\delta(<q_0,\varepsilon>,q_z) \mid q_0 = S_{AB}, (q_z = S_A \text{ or } q_z = S_B)\}$
    $\cup \{\delta(<q_z,\varepsilon>,q_j) \mid (q_z \in F_A \text{ or } q_z \in F_B), q_i \in F_{AB}\}$



- Kleene Star over an FSA F_A :
  - $S_{A*} = s_0$ (new state)
  - $F_{A*} = \{ q_j \}$ (new state)
  - $\delta_{AB} = \delta_A \cup$
    $\cup \{\delta(<q_j,\varepsilon>,q_z) \mid q_j \in F_A, q_z = S_A)\}$
    $\cup \{\delta(<q_0,\varepsilon>,q_z) \mid q_0 = S_{A*}, (q_z = S_A \text{ or } q_z = F_{A*})\}$
    $\cup \{\delta(<q_z,\varepsilon>,q_j) \mid q_z \in F_A , q_j \in F_{A*}\}$



(ab ∪ aba)*



- *ε-transition:* move to δ(q, ε) without reading an input symbol
- FSA construction from regular expressions yields a non-deterministic FSA (NFSA)
  - Choice of next state is *only partially determined* by the current configuration, i.e., we cannot always predict which state will be the next state in the traversal

Overview of the lecture

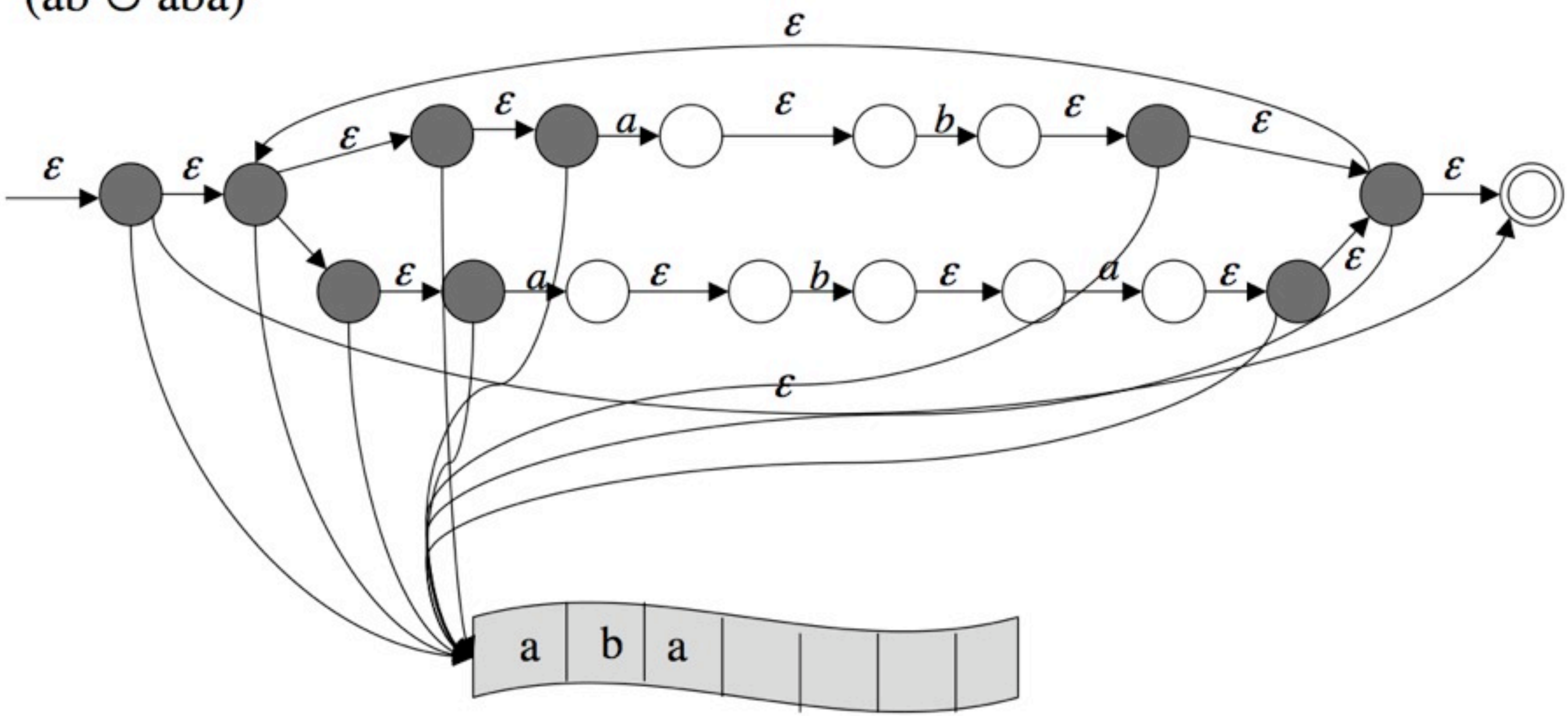
- Background
  - Chomsky hierarchy of languages
  - Basic definitions, generic operations on languages
- **Generalities about Finite-State Automata (FSA)**
  - Regular languages, regular expressions and FSAs
  - Constructing a FSA from a regular expression
  - **Non-deterministic FSAs**
- Optimization algorithms for FSAs
  - *Determinization* of a FSA via subset construction
  - *Minimization* of a FSA: equivalence classes, Brzozowski‘s algorithm
- Applications of FSAs & extensions to finite-state transducers
- Conclusions, exercises

Non-deterministic FSA

- *Non-determinism*
  - Introduced by ε-transitions and/or
  - Transition being a *relation* Δ over $\Phi \times \Sigma^* \times \Phi$, i.e. a set of triples $<q_{source}, z, q_{target}>$
    Equivalently: Transition function δ maps to a *set of states*: $\delta: \Phi \times \Sigma \rightarrow \wp(\Phi)$
- A non-deterministic FSA (NFSA) is a tuple $A = <\Phi, \Sigma, \delta, q_0, F>$
  - Φ a finite non-empty set of states
  - $\boldsymbol{\Sigma}$ a finite alphabet of input letters
  - δ a transition function $\Phi \times \boldsymbol{\Sigma}^* \rightarrow \wp(\Phi)$ (or a finite relation over $\Phi \times \Sigma^* \times \Phi$)
  - $q_0 \in \Phi$ the initial state
  - $F \subseteq \Phi$ the set of final (accepting) states
- Adapted definitions for *transitions and acceptance* of a string by a NFSA
  - $(q,w) \vdash_A (q',w_{i+1})$ iff $w_i = zw_{i+1}$ for $z \in \Sigma^*$ and $q' \in \delta(q,z)$
  - An NDFA (w/o ε) *accepts* a string w iff there is *some traversal* such that $(q_0,w) \vdash^*_A (q', \varepsilon)$ and $q' \subseteq F$.
  - A string w is *rejected* by NDFA A iff A does not accept w,
    i.e. *all configurations* of A for string w are rejecting configurations!

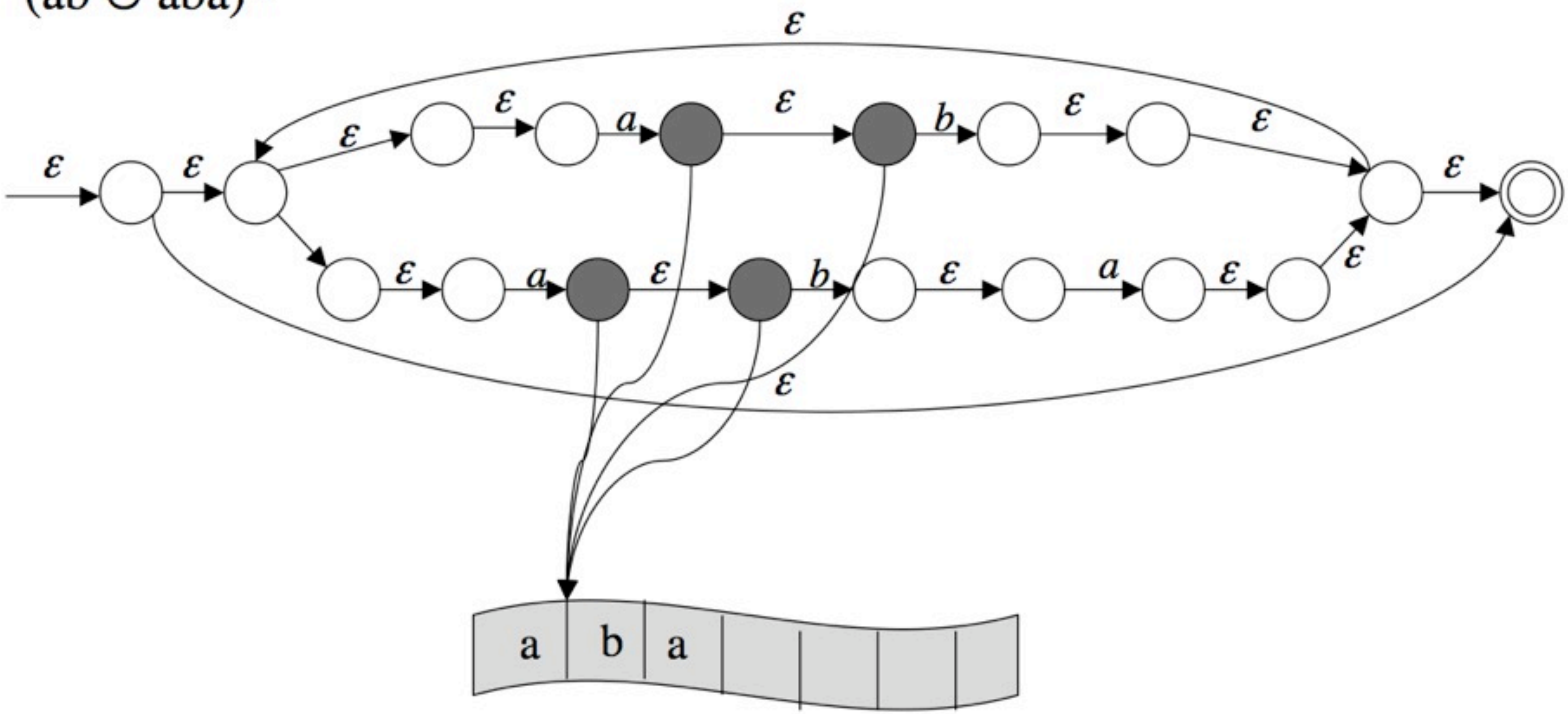
Non-determinism in FSAs

(ab ∪ aba)*

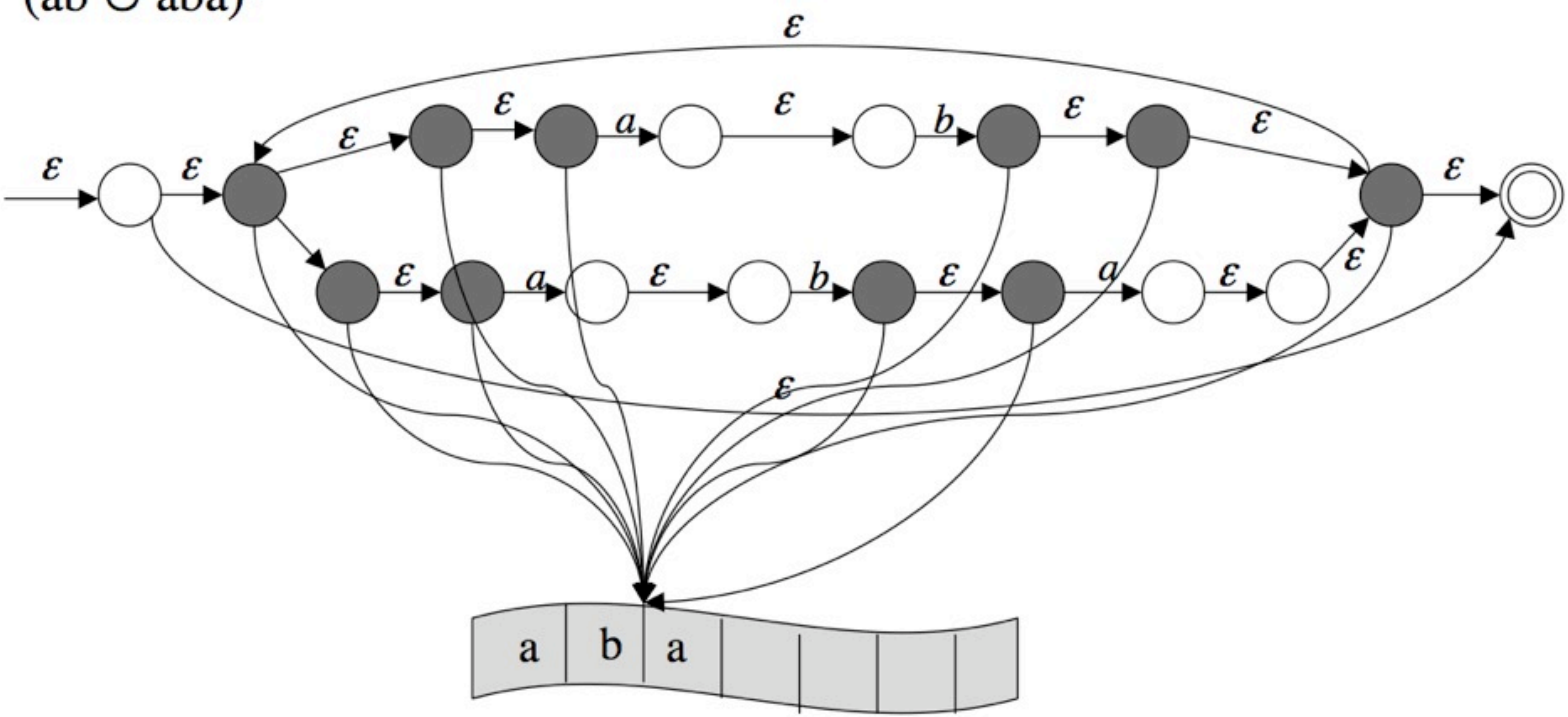


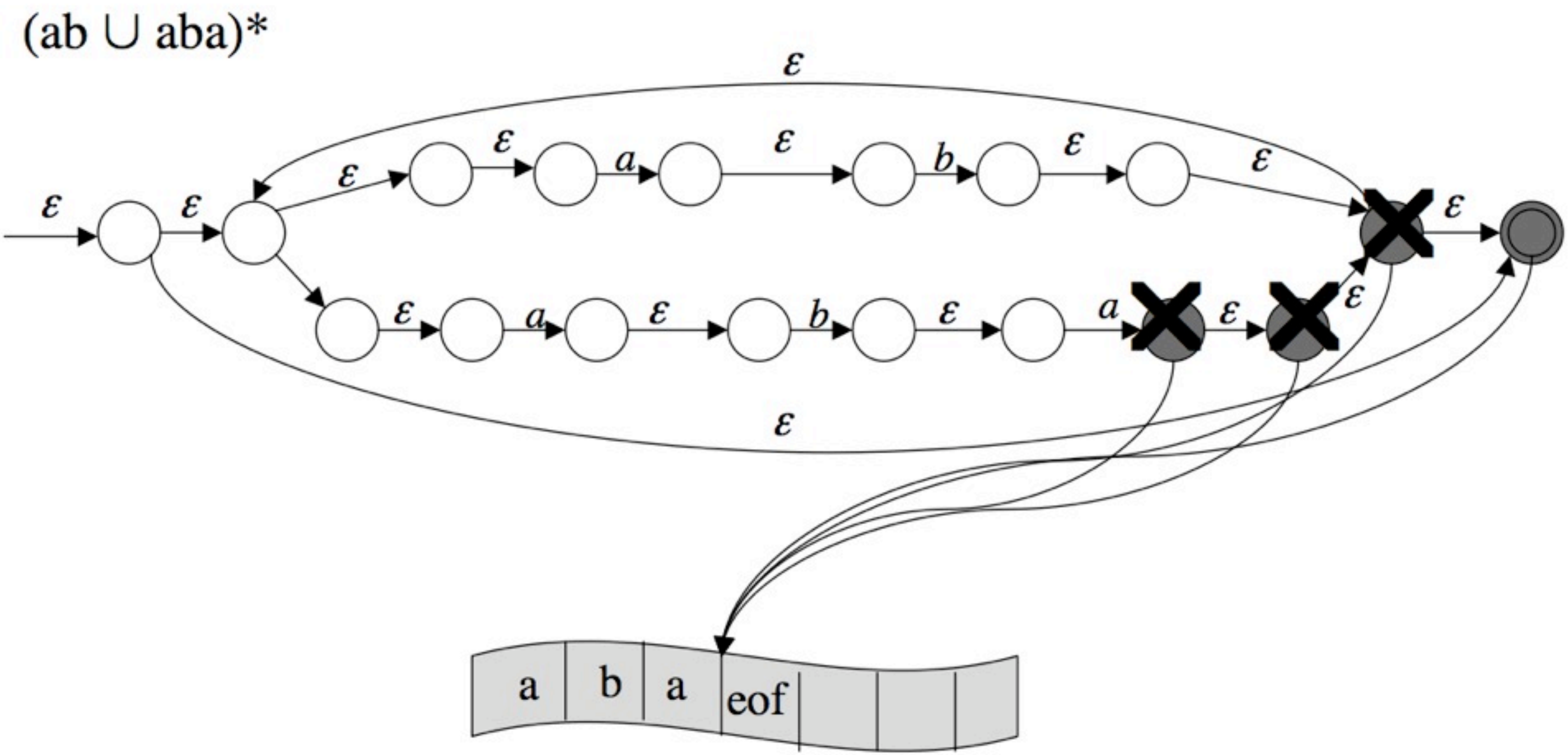
Non-determinism in FSAs

(ab ∪ aba)*

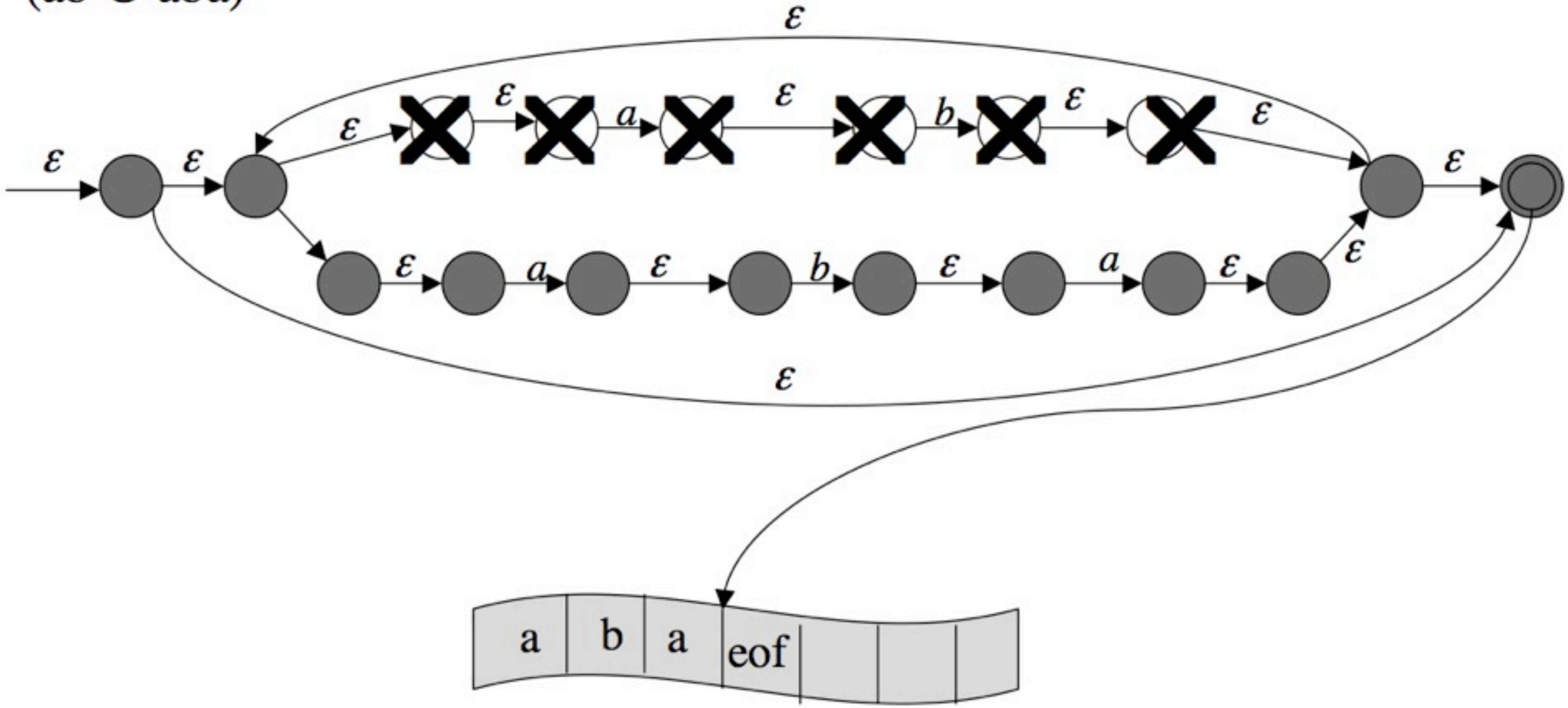


(ab ∪ aba)*




(ab ∪ aba)*
ε
ε
ε
ε
a
ε
b
ε
ε
ε
ε
ε
a
ε
b
ε
a
ε
ε
ε
a
b
a
eof

(ab ∪ aba)*



Equivalence of DFSAs and NFSAs

- Despite non-determinism, NFSAs are not more powerful than DFSAs: they accept the same class of languages: regular languages
- For every non-deterministic FSA there is deterministic FSA that accepts the same language (and vice versa)
  - The corresponding DFSA has in general more states, in which it models the sets of possible states the NFSA could be in in a given traversal
- There is an algorithm (via subset construction) that allows conversion of an NFSA to an equivalent DFSA

Efficiency considerations: an FSA is most efficient and compact iff

- It is a DFSA (efficiency) → Determinization of NFSA
- It is minimal (compact encoding) → Minimization of FSAs

Overview of the lecture

- Background
  - Chomsky hierarchy of languages
  - Basic definitions, generic operations on languages
- Generalities about Finite-State Automata (FSA)
  - Regular languages, regular expressions and FSAs
  - Constructing a FSA from a regular expression
  - Non-deterministic FSAs
- **Optimization algorithms for FSAs**
  - *Determinization* of a FSA via subset construction
  - *Minimization* of a FSA: equivalence classes, Brzozowski's algorithm
- Applications of FSAs & extensions to finite-state transducers
- Conclusions, exercises

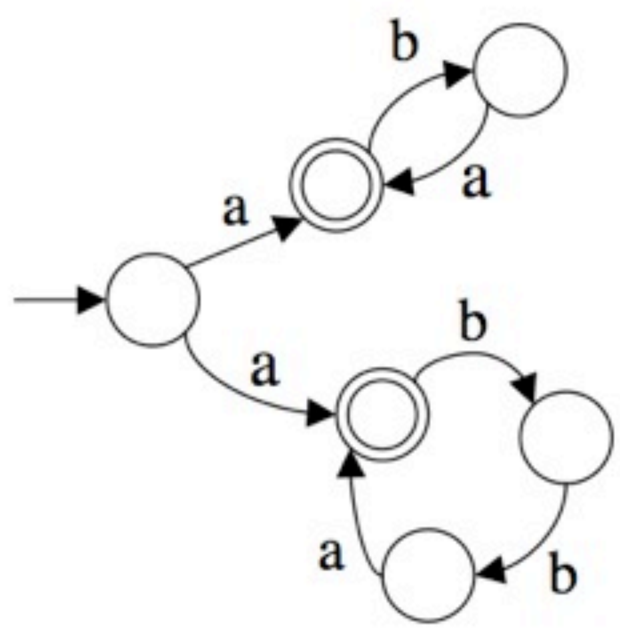
Overview of the lecture

- Background
  - Chomsky hierarchy of languages
  - Basic definitions, generic operations on languages
- Generalities about Finite-State Automata (FSA)
  - Regular languages, regular expressions and FSAs
  - Constructing a FSA from a regular expression
  - Non-deterministic FSAs
- **Optimization algorithms for FSAs**
  - ***Determinization* of a FSA via subset construction**
  - *Minimization* of a FSA: equivalence classes, Brzozowski‘s algorithm
- Applications of FSAs & extensions to finite-state transducers
- Conclusions, exercises

Determin. by subset construction

NFSA $A=<\Phi,\Sigma,\delta,q_0,F>$

$A'=<\Phi',\Sigma,\delta',q_0',F'>$

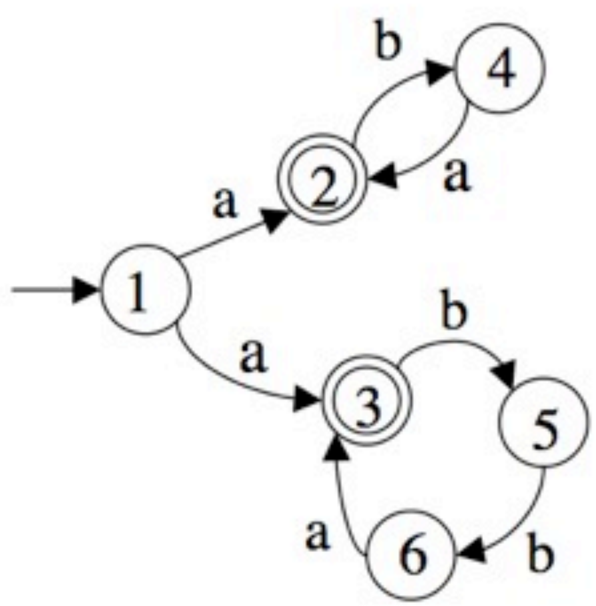


$L(A)= a(ba)^* \cup a(bba)^*$

Subset construction:

Compute δ' from δ
for all subsets $S \subseteq \Phi$ and $a \in \Sigma$ s.th.
$\delta'(S,a) = \{ s' \mid \exists s \in S \text{ s.th. } (s,a,s') \in \delta \}$

NFSA $A=<\Phi,\Sigma,\delta,q_0,F>$ $A'=<\Phi',\Sigma,\delta',q_0',F'>$



$L(A)= a(ba)^* \cup a(bba)^*$

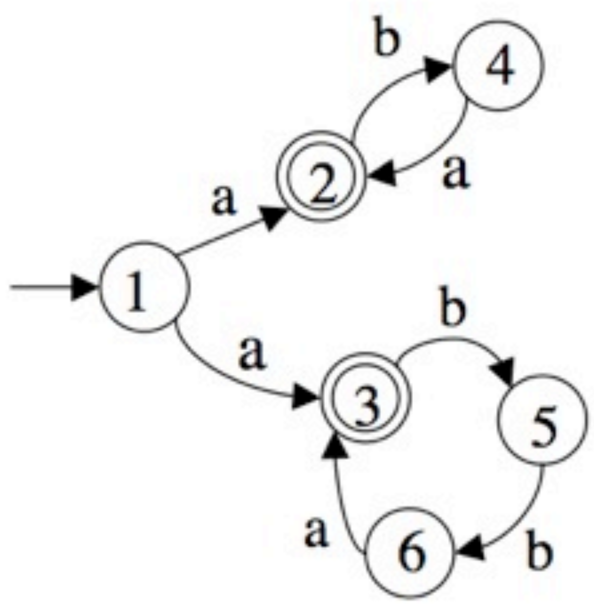
$\Phi'= \{ B \mid B \subseteq \{1,2,3,4,5,6\}$
$q_0'=\{1\}$,
$\delta'(\{1\},a)=\{2,3\}$,
$\delta'(\{1\},b)=\varnothing$,
$\delta'(\{2,3\},a)= \varnothing$,
$\delta'(\{2,3\},b)= \{4,5\}$,
$\delta'(\{4,5\},a)= \{2\}$,
$\delta'(\{4,5\},b)= \{6\}$,
$\delta'(\{2\},a)= \varnothing$,
$\delta'(\{2\},b)= \{4\}$,
$\delta'(\{6\},a)= \{3\}$,
$\delta'(\{6\},b)= \varnothing$,

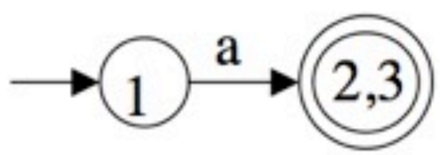
$\delta'(\{4\},a)= \{2\}$,
$\delta'(\{4\},b)= \varnothing$,
$\delta'(\{3\},a)= \varnothing$,
$\delta'(\{3\},b)= \{5\}$,
$\delta'(\{5\},a)= \varnothing$,
$\delta'(\{5\},b)= \{6\}$

$F'= \{\{2,3\},\{2\},\{3\}\}$

Determin. by subset construction

NFSA $A = \langle \Phi, \Sigma, \delta, q_0, F \rangle$ $\quad$ $A' = \langle \Phi', \Sigma, \delta', q_0', F' \rangle$





$\Phi' = \{ B \mid B \subseteq \{1,2,3,4,5,6\}$

$q_0' = \{1\}$,

$\delta'(\{1\},a) = \{2,3\}$,

$\delta'(\{1\},b) = \emptyset$,

$\delta'(\{2,3\},a) = \emptyset$,

$\delta'(\{2,3\},b) = \{4,5\}$,

$\delta'(\{4,5\},a) = \{2\}$,

$\delta'(\{4,5\},b) = \{6\}$,

$\delta'(\{2\},a) = \emptyset$,

$\delta'(\{2\},b) = \{4\}$,

$\delta'(\{6\},a) = \{3\}$,

$\delta'(\{6\},b) = \emptyset$,

$\delta'(\{4\},a) = \{2\}$,

$\delta'(\{4\},b) = \emptyset$,

$\delta'(\{3\},a) = \emptyset$,

$\delta'(\{3\},b) = \{5\}$,

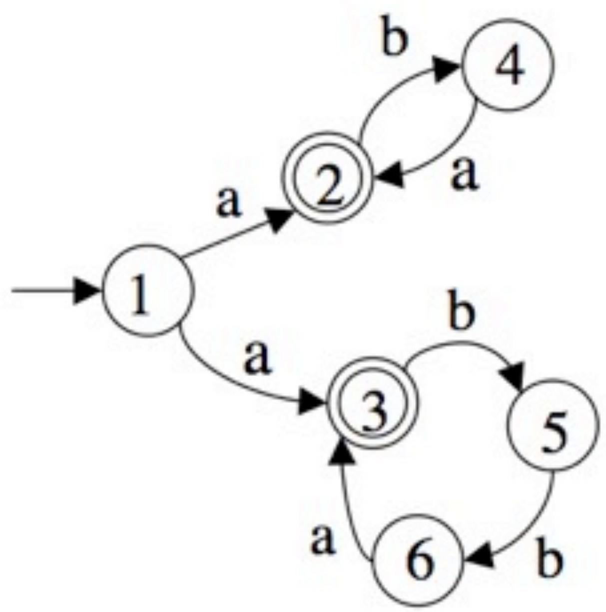
$\delta'(\{5\},a) = \emptyset$,

$\delta'(\{5\},b) = \{6\}$

$F' = \{\{2,3\},\{2\},\{3\}\}$

Determin. by subset construction

NFSA $A=\langle\Phi,\Sigma,\delta,q_0,F\rangle$ $A'=\langle\Phi',\Sigma,\delta',q_0',F'\rangle$



$\Phi' = \{ B \mid B \subseteq \{1,2,3,4,5,6\}$

$q_0' = \{1\}$,

$\delta'(\{1\},a)=\{2,3\}$,

$\delta'(\{1\},b)=\varnothing$,

$\delta'(\{2,3\},a)=\varnothing$,

$\delta'(\{2,3\},b)=\{4,5\}$,

$\delta'(\{4,5\},a)=\{2\}$,

$\delta'(\{4,5\},b)=\{6\}$,

$\delta'(\{2\},a)=\varnothing$,

$\delta'(\{2\},b)=\{4\}$,

$\delta'(\{6\},a)=\{3\}$,

$\delta'(\{6\},b)=\varnothing$,

$\delta'(\{4\},a)=\{2\}$,

$\delta'(\{4\},b)=\varnothing$,

$\delta'(\{3\},a)=\varnothing$,

$\delta'(\{3\},b)=\{5\}$,

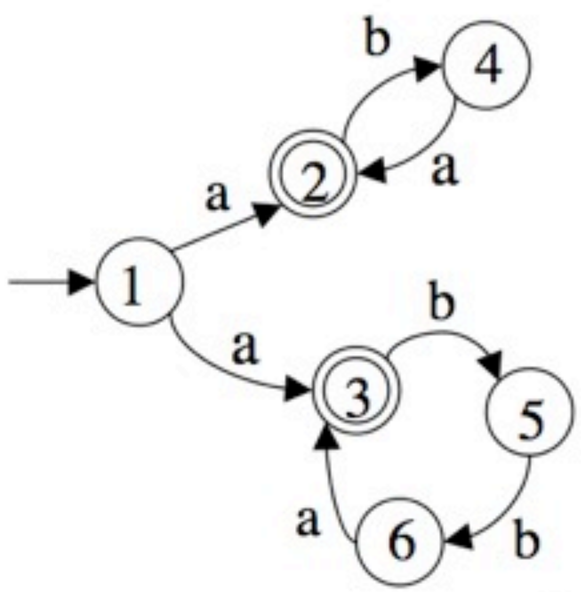
$\delta'(\{5\},a)=\varnothing$,

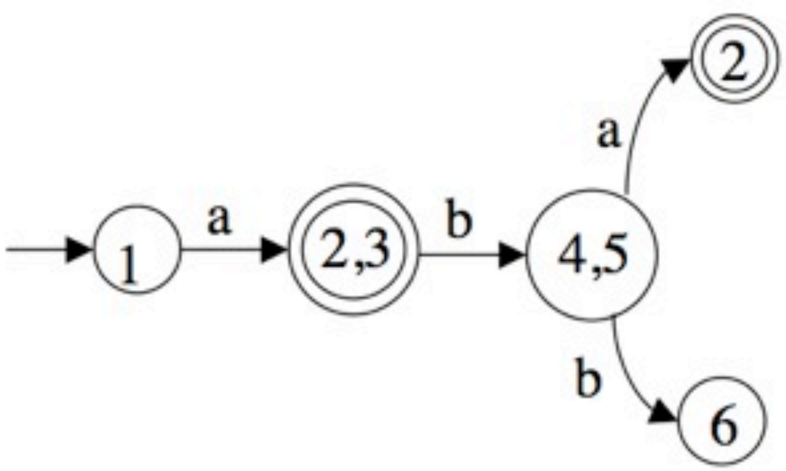
$\delta'(\{5\},b)=\{6\}$

$F' = \{\{2,3\},\{2\},\{3\}\}$

Determin by subset construction

NFSA $A = \langle \Phi, \Sigma, \delta, q_0, F \rangle$ $\quad$ $A' = \langle \Phi', \Sigma, \delta', q_0', F' \rangle$





$\Phi' = \{ B \mid B \subseteq \{1,2,3,4,5,6\}$

$q_0' = \{1\}$,

$\delta'(\{1\},a)=\{2,3\}$,

$\delta'(\{1\},b)=\varnothing$,

$\delta'(\{2,3\},a)= \varnothing$,

$\delta'(\{2,3\},b)= \{4,5\}$,

$\delta'(\{4,5\},a)= \{2\}$,

$\delta'(\{4,5\},b)= \{6\}$,

$\delta'(\{2\},a)= \varnothing$,

$\delta'(\{2\},b)= \{4\}$,

$\delta'(\{6\},a)= \{3\}$,

$\delta'(\{6\},b)= \varnothing$,

$\delta'(\{4\},a)= \{2\}$,

$\delta'(\{4\},b)= \varnothing$,

$\delta'(\{3\},a)= \varnothing$,

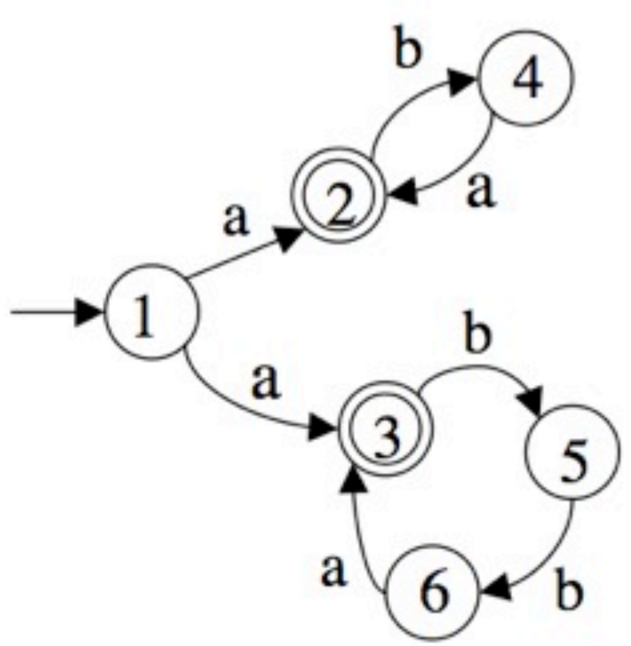
$\delta'(\{3\},b)= \{5\}$,

$\delta'(\{5\},a)= \varnothing$,

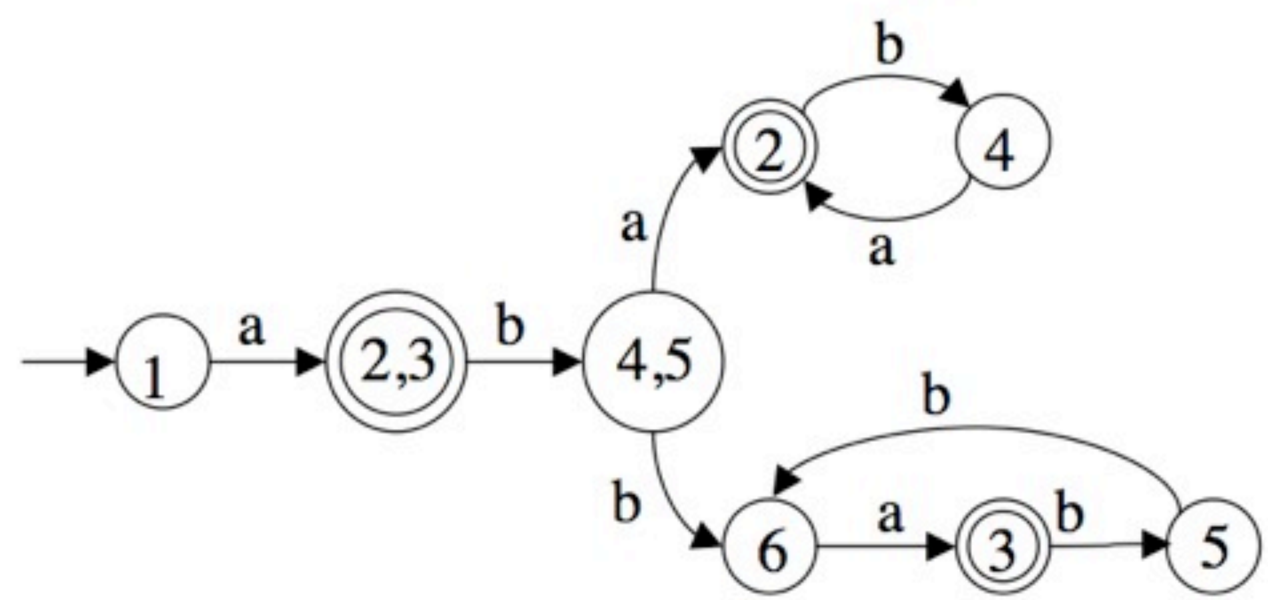
$\delta'(\{5\},b)= \{6\}$

$F' = \{\{2,3\},\{2\},\{3\}\}$

NFSA $A=\langle\Phi,\Sigma,\delta,q_0,F\rangle$



DFSA $A'=\langle\Phi',\Sigma,\delta',q_0',F'\rangle$



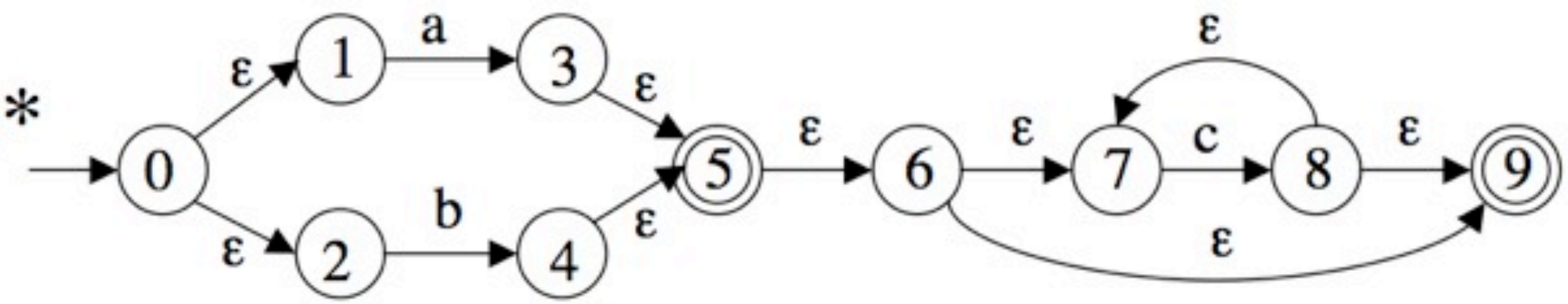
$$\boxed{L(A) = L(A') = a(ba)^* \cup a(bba)^*}$$

ε-transitions and ε-closure

- Subset construction must account for ε-transitions
- ε-closure
  - The ε-closure of some state q consists of q as well as all states that can be reached from q through a sequence of ε-transitions
    - $q \in \varepsilon\text{-closure}(q)$
    - If $r \in \varepsilon\text{-closure}(q)$ and $(r, \varepsilon, q') \in \delta$, then $q' \in \varepsilon\text{-closure}(q)$,
  - ε-closure defined on sets of states
    - $\varepsilon\text{-closure}(R) = \bigcup_{q \in R} \varepsilon\text{-closure}(q)$ (with $R \subset \Phi$)
- Subset construction for ε-NFSAs
  - Compute δ' from δ for all subsets $S \subseteq \Phi$ and $a \in \Sigma$ s.th.
    $\delta'(S,a) = \{ s'' \mid \exists s \in S \text{ s.th. } (s,a,s') \in \delta \text{ and } s'' \in \varepsilon\text{-closure}(s') \}$

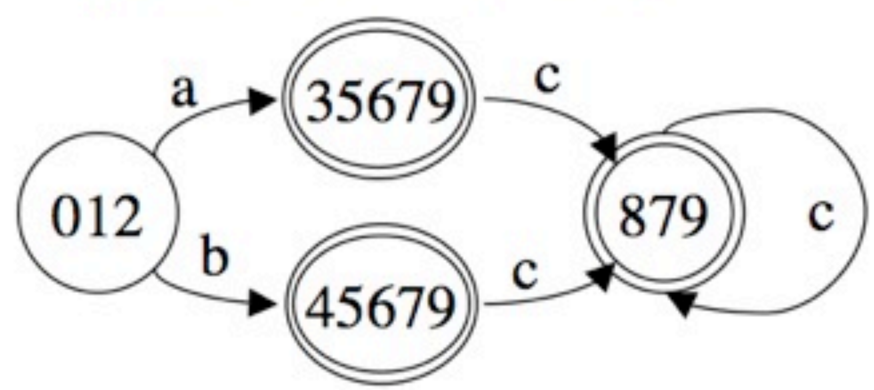
Example

- ε-NFSA for (a|b)c*

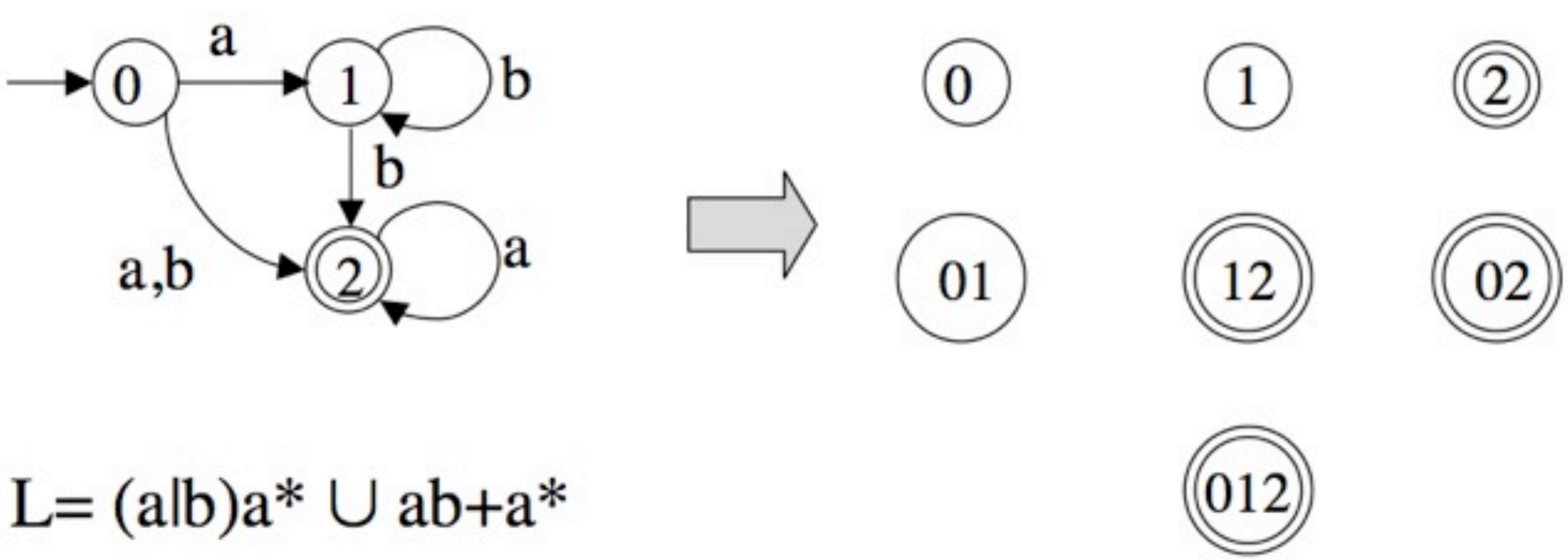


ε-closure for all s∈Φ:
ε-closure(0)={0,1,2},
ε-closure(1)={1},
ε-closure(2)={2},
ε-closure(3)={3,5,6,7,9},
ε-closure(4)={4,5,6,7,9},
ε-closure(5)={5,6,7,9},
ε-closure(6)={6,7,9},
ε-closure(7)={7},
ε-closure(8)={8,7,9},
ε-closure(9)={9}

Transition function over subsets
δ'({0},ε)= {0,1,2},
δ'({0,1,2},a)={3,5,6,7,9},
δ'({0,1,2},b)= {4,5,6,7,9},
δ'({3,5,6,7,9},c)= {8,7,9},
δ'({4,5,6,7,9},c)= {8,7,9},
δ'({8,7,9},c)= {8,7,9}

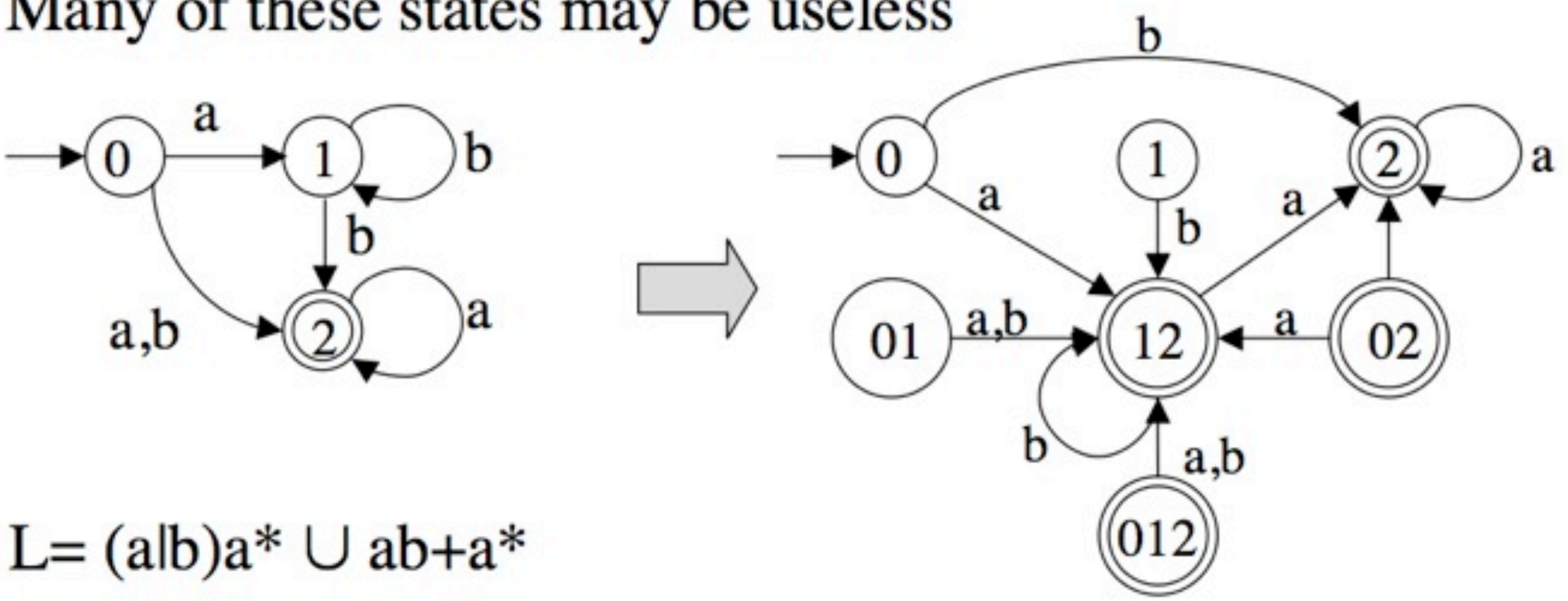


- Construction of DFSA A'=<Φ',Σ, δ',q_0',F'> from NFSA A=<Φ,Σ, δ, q_0,F>
  - Φ'={B| B⊆Φ}, if unconstrained can be $2^{|\Phi|}$
    with |Φ| = 33 this could lead to an FSA with 2^{33} states
    (exceeds the range of integers in most programming languages)
  - Many of these states may be useless



L= (a|b)a* ∪ ab+a*

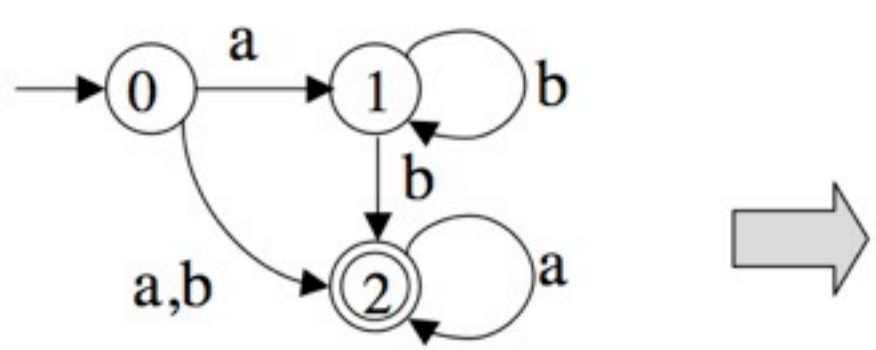
- Construction of DFSA $A' = \langle \Phi', \Sigma, \delta', q_0, F' \rangle$ from NFSA $A = \langle \Phi, \Sigma, \delta, q_0, F \rangle$
  - $\Phi' = \{B \mid B \subseteq \Phi\}$, if unconstrained can be $2^{|\Phi|}$
    with $|\Phi| = 33$ this could lead to an FSA with 2^{33} states
    (exceeds the range of integers in many programming languages)
  - Many of these states may be useless

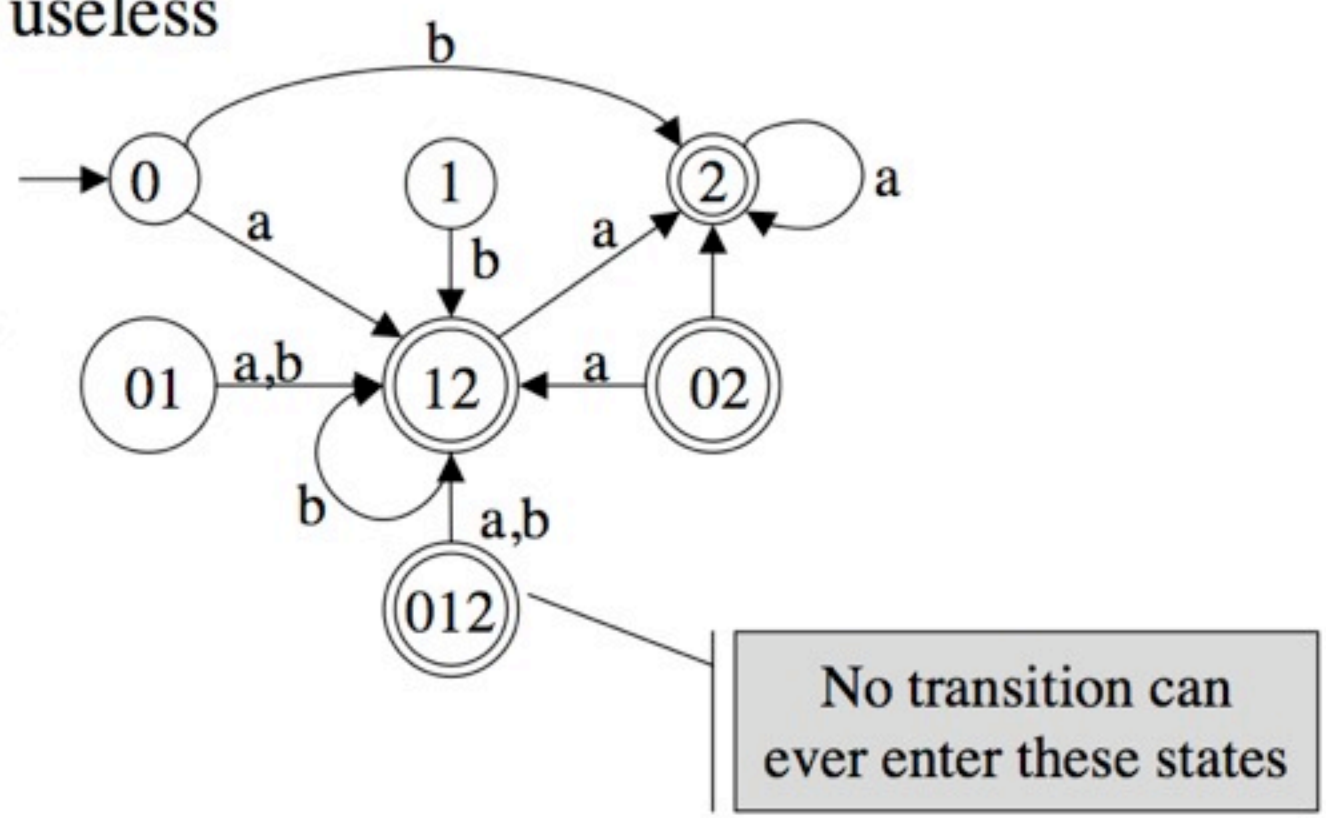


L= (a|b)a* ∪ ab+a*

Algorithm for subset construction

- Construction of DFSA $A' = \langle \Phi', \Sigma, \delta', q_0, F' \rangle$ from NFSA $A = \langle \Phi, \Sigma, \delta, q_0, F \rangle$
  - $\Phi' = \{B \mid B \subseteq \Phi\}$, if unconstrained can be $2^{|\Phi|}$
    with $|\Phi| = 33$ this could lead to an FSA with 2^{33} states
    (exceeds the range of integers in many programming languages)
  - Many of these states may be useless

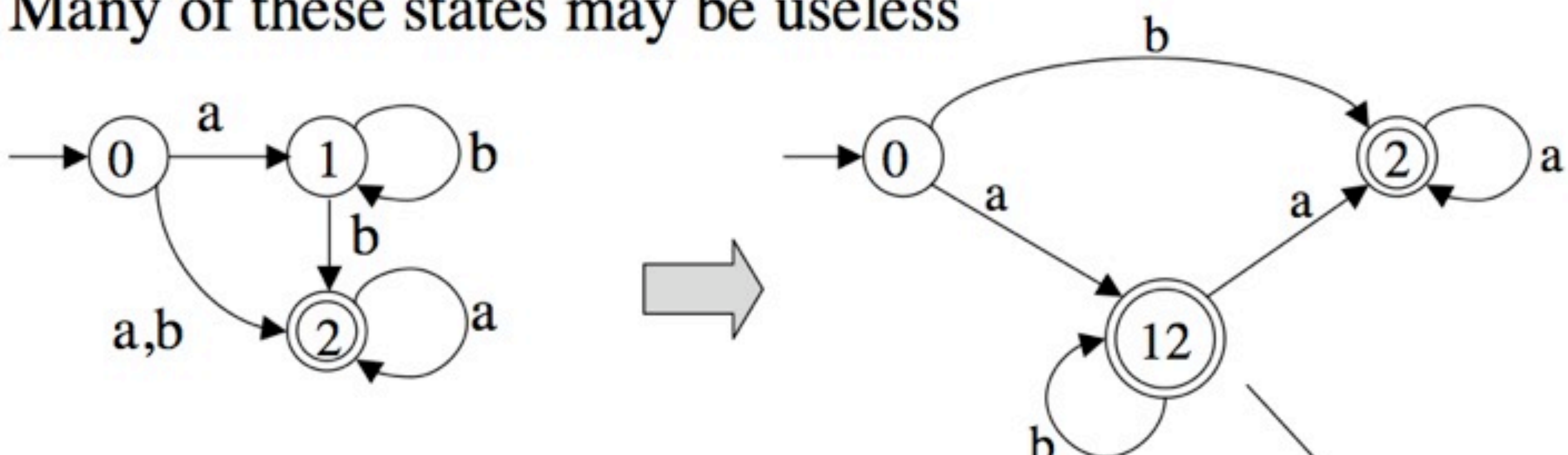




L= (a|b)a* ∪ ab+a*

Algorithm for subset construction

- Construction of DFSA $A' = \langle \Phi', \Sigma, \delta', q_0, F' \rangle$ from NFSA $A = \langle \Phi, \Sigma, \delta, q_0, F \rangle$
  - $\Phi' = \{B \mid B \subseteq \Phi\}$, if unconstrained can be $2^{|\Phi|}$
    with $|\Phi| = 33$ this could lead to an FSA with 2^{33} states
    (exceeds the range of integers in many programming languages)
  - Many of these states may be useless

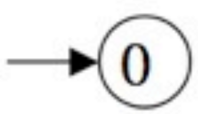


$L = (a|b)a^* \cup ab^+a^*$

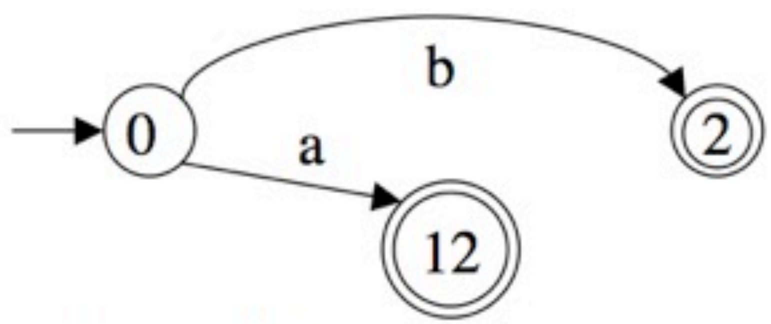
Only consider states that can be traversed starting from q_0

Algorithm for subset construction

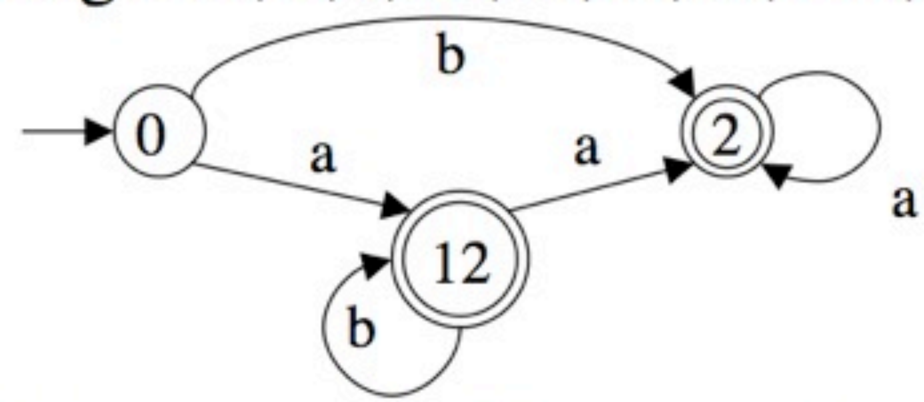
- Basic idea: we only need to consider states $B \subseteq \Phi$ that can ever be traversed by a string $w \in \Sigma^*$, starting from q_0'
- I.e., those $B \subseteq \Phi$ for which $B = \delta'(q_0, w)$, for some $w \in \Sigma^*$, with δ' the recursively constructed transition function for the target DFSA A'
- Consider all strings $w \in \Sigma^*$ in order of their length: ε, a,b, aa,ab,ba,bb, aaa,...



l=0 (ε)



l=1 (a,b)

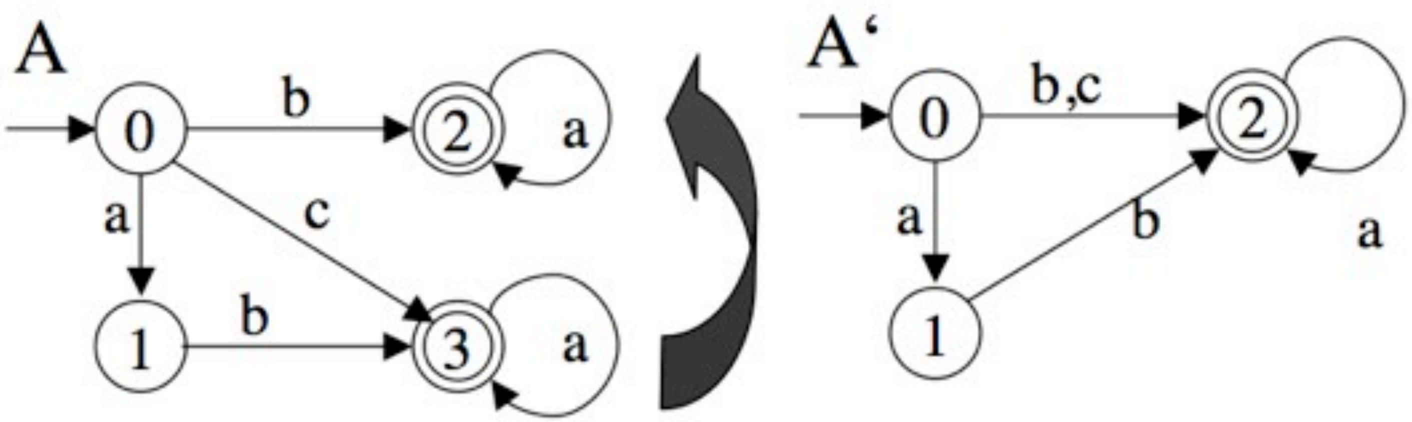


l=2,3,4, … (aa, ab, ba, bb, aaa, aab, aba, …)

- Construction by increasing lengths of strings
- For each $a \in \Sigma$, construct transitions to known or new states according to δ
- New target states (A') are placed in a queue (FIFO)
- Termination: no states left on queue

- Background
  - Chomsky hierarchy of languages
  - Basic definitions, generic operations on languages
- Generalities about Finite-State Automata (FSA)
  - Regular languages, regular expressions and FSAs
  - Constructing a FSA from a regular expression
  - Non-deterministic FSAs
- **Optimization algorithms for FSAs**
  - *Determinization* of a FSA via subset construction
  - ***Minimization* of a FSA: equivalence classes,** Brzozowski's algorithm
- Applications of FSAs & extensions to finite-state transducers
- Conclusions, exercises

- Can we transform a large automaton into a smaller one (provided a smaller one exists)?
- If A is a DFSA, is there an algorithm for constructing an equivalent minimal automaton A_{min} from A?



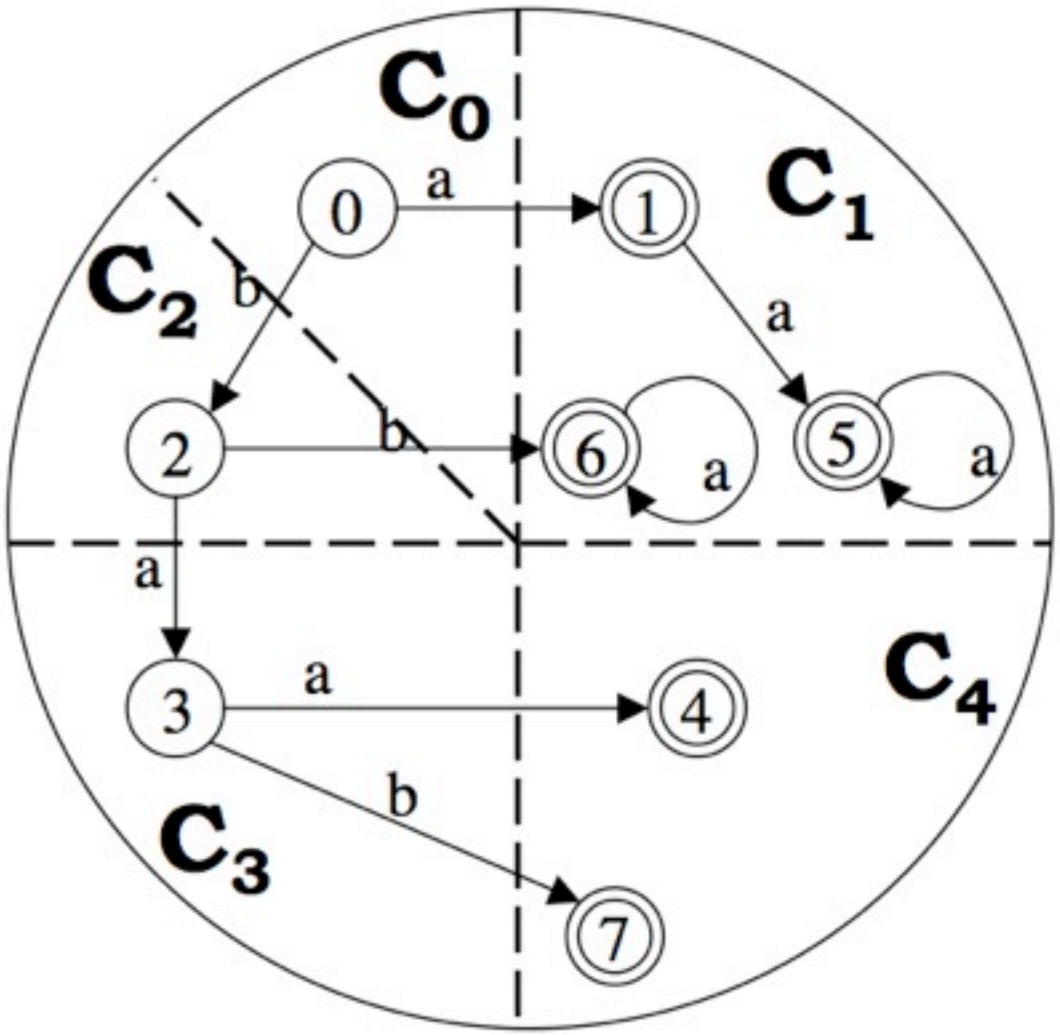
A is *equivalent* to A'
i.e., L(A) = L(A')

A' is *smaller* than A
i.e., $|\Phi| > |\Phi'|$

- A can be transformed to A':
  - States 2 and 3 in A "*do the same job*": once A is in state 2 or 3, it *accepts the same suffix string*. Such states are called *equivalent*.
  - Thus, we can eliminate state 3 without changing the language of A, by *redirecting* all arcs leading to 3 to 2, instead.

- A DFSA can be minimized if there are *pairs of states* $q,q' \in \Phi$ that are *equivalent*
- Two states q,q' are *equivalent* iff they accept the *same right language*.

- Right language of a state:
  - For $A=<\Phi,\Sigma,\delta,q_0,F>$ a DFSA, *the right language $L^{\rightarrow}(q)$ of a state $q \in \Phi$* is the set of all strings accepted by A starting in state q: $L^{\rightarrow}(q) = \{w \in \Sigma^* \mid \delta^*(q,w) \in F\}$
  - Note: $L^{\rightarrow}(q_0) = L(A)$
- State equivalence:
  - For $A=<\Phi,\Sigma,\delta,q_0,F>$ a DFSA, if $q,q' \in \Phi$, *q and q' are equivalent* ($q \equiv q'$) iff $L^{\rightarrow}(q) = L^{\rightarrow}(q')$
  - $\equiv$ is an equivalence relation (i.e., reflexive, transitive and symmetric)
  - $\equiv$ *partitions* the set of states Φ into a number of *disjoint sets Q_1 .. Q_n of equivalence classes* s.th. $\bigcup_{i=1..m} Q_i = \Phi$ and $q \equiv q'$ for all $q,q' \in Q_i$



All classes C_i consist of *equivalent states* $q_{j=i..n}$ that accept *identical right languages* $L^{\rightarrow}(q_j)$

Whenever two states q,q' belong to different classes, $L^{\rightarrow}(q) \neq L^{\rightarrow}(q')$

Equivalence classes on state set defined by ≡

Minimization:
elimination of equivalent states

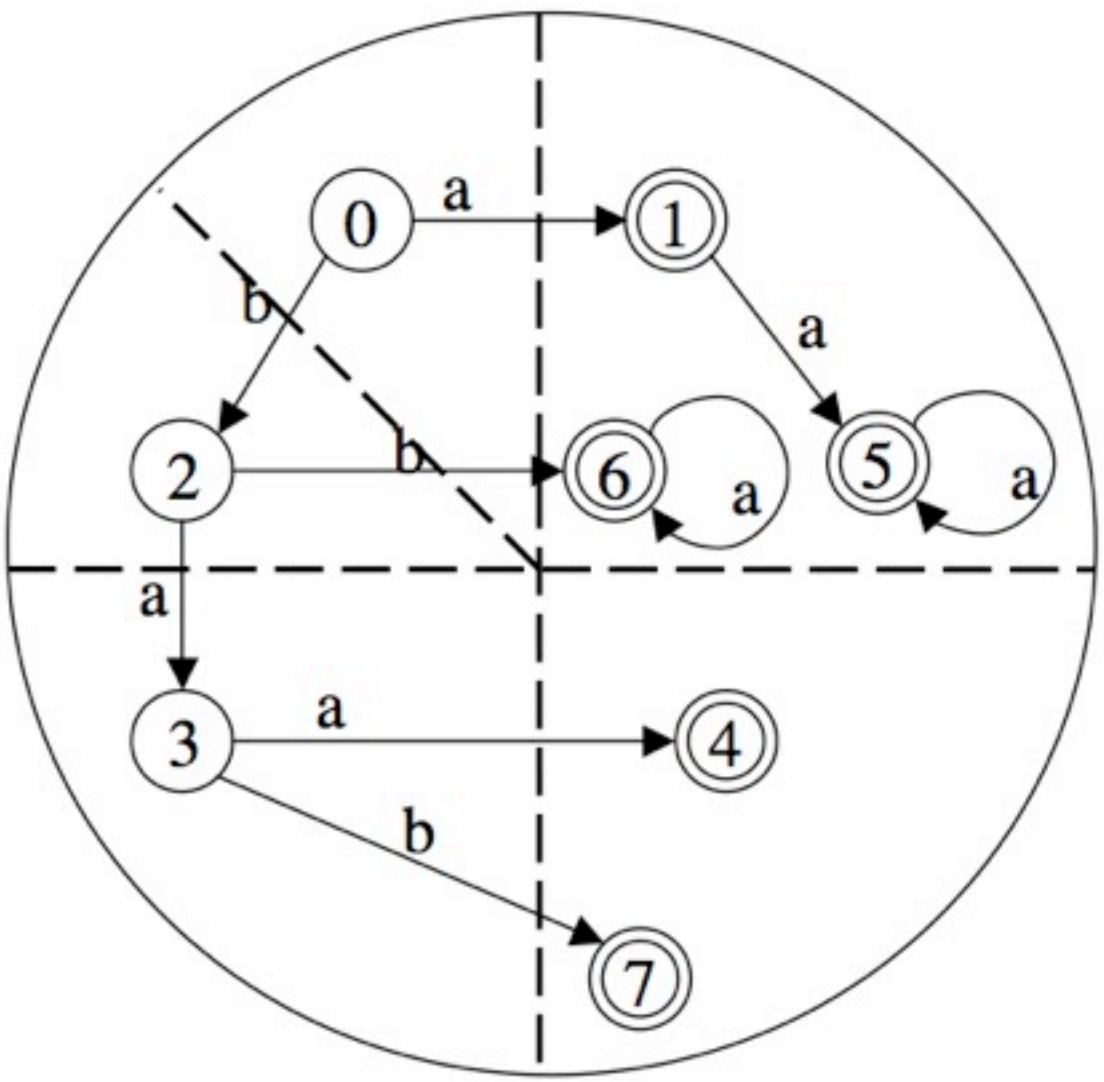
Minimization of a DFSA

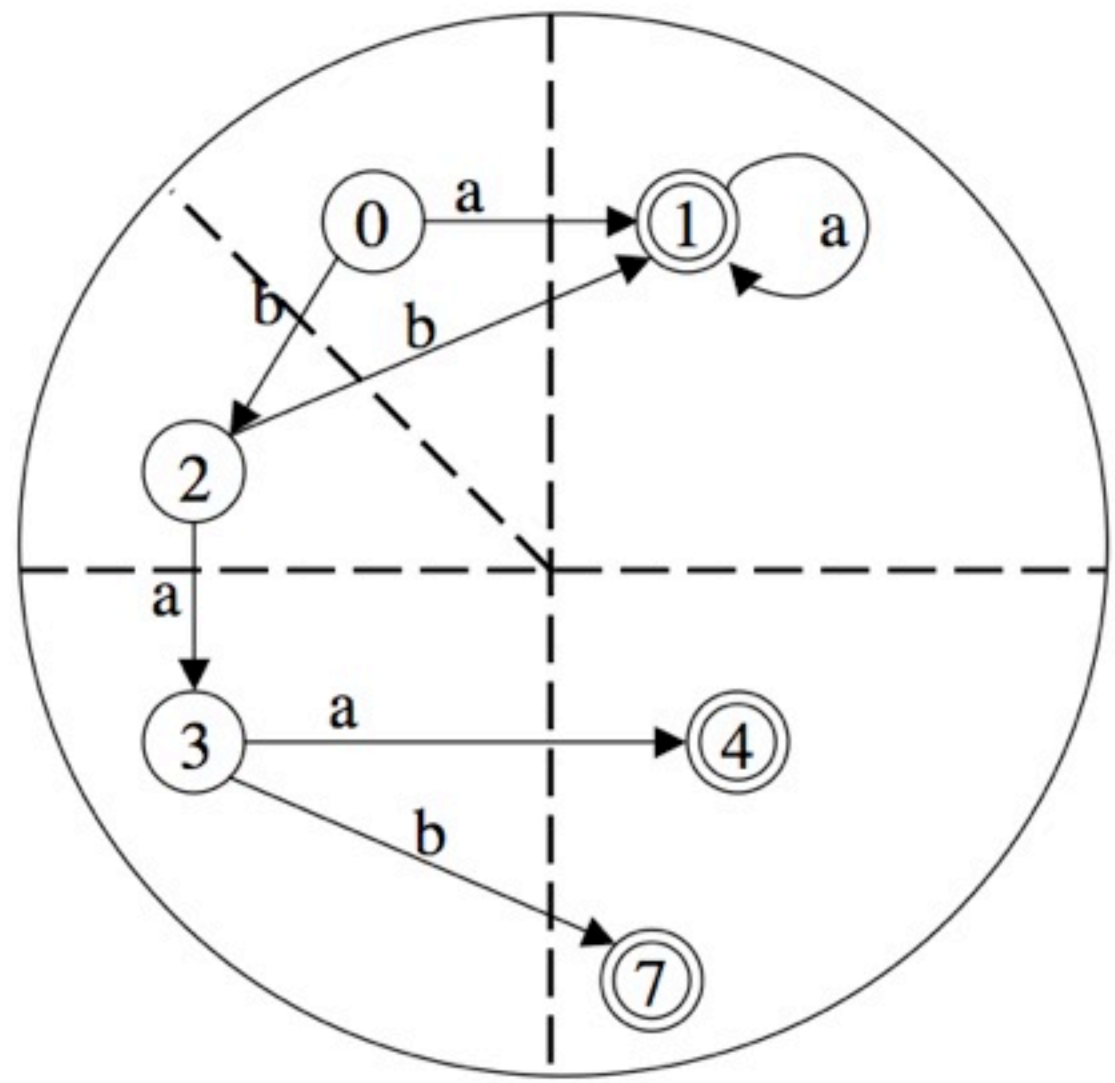
A DFSA $A=<\Phi,\Sigma, \delta, q_0,F>$ that contains *equivalent states q, q'*
can be transformed to a smaller, equivalent DFSA $A'=<\Phi',\Sigma, \delta', q_0,F'>$ where

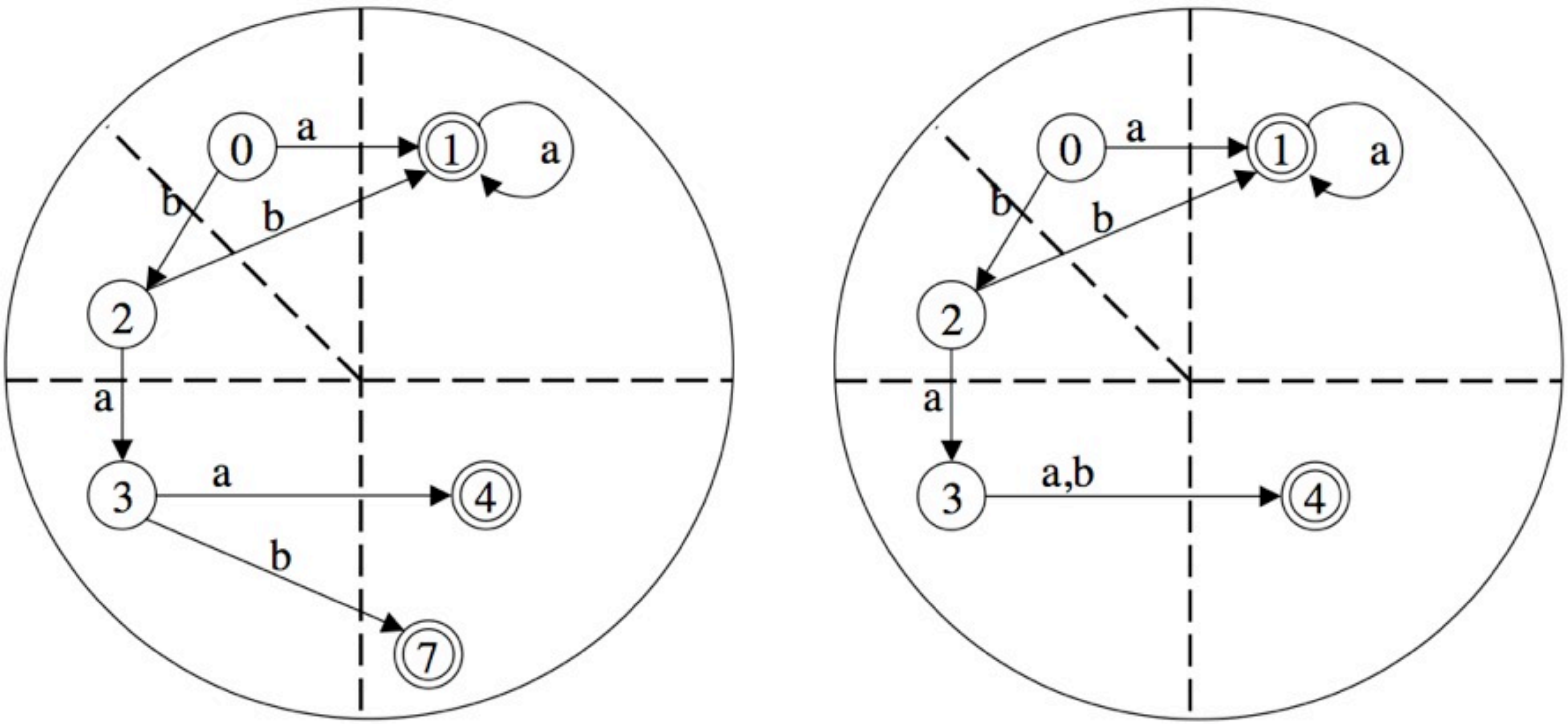
- $\Phi' = \Phi\backslash\{q'\}$, $F'=F\backslash\{q'\}$,
- δ' is like δ with all transitions to q' redirected to q: $\delta'(s,a) = q$ if $\delta(s,a) = q'$; $\delta'(s,a) = \delta(s,a)$ otherwise

- Two-step algorithm
  - Determine all pairs of equivalent states q,q'
  - Apply DFSA reduction until no such pair q,q' is left in the automaton
- *Minimality*
  - The resulting FSA is the smallest DFSA (in size of Φ) that accepts L(A): we never merge different equivalence classes, so we obtain one state per class.
    - We cannot do any further reduction and still recognize L(A).
    - As long as we have >1 state per class, we can do further reduction steps.
- A DFSA $A=<\Phi,\Sigma, \delta, q_0,F>$ is *minimal* iff there is no pair of distinct but equivalent states $\in\Phi$, i.e. $\forall\, q, q'\in\Phi : q \equiv q' \Leftrightarrow q = q'$

Example



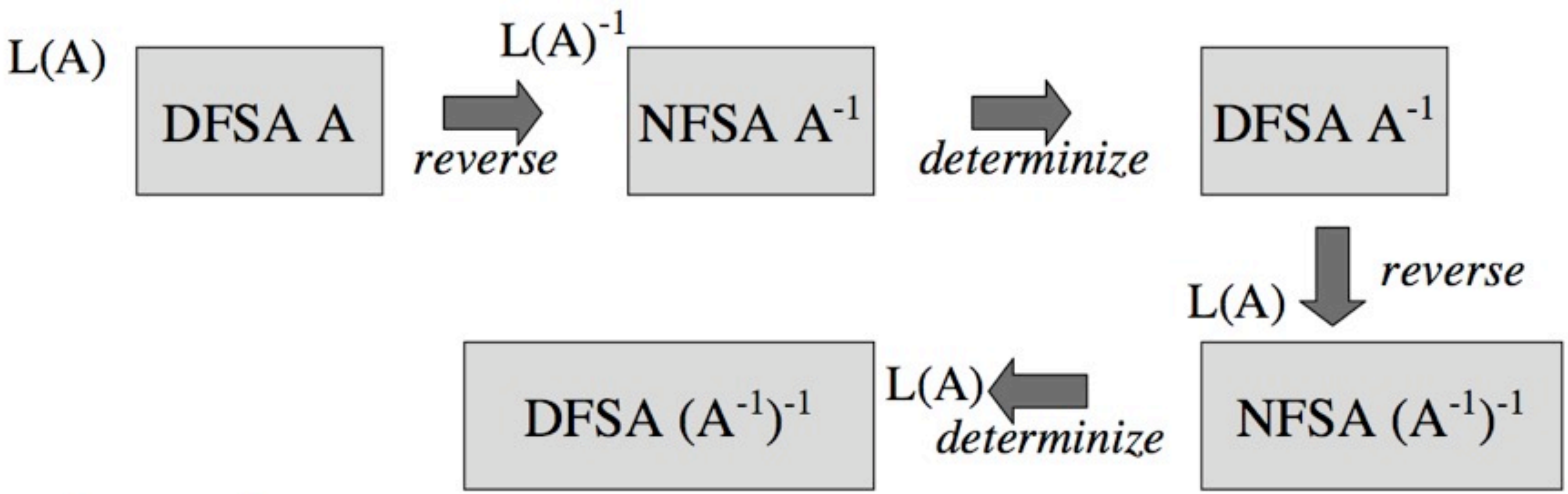



0
a
1
a
b
b
2
a
3
a
4
b
7
0
a
1
a
b
b
2
a
3
a,b
4

Overview of the lecture

- Background
  - Chomsky hierarchy of languages
  - Basic definitions, generic operations on languages
- Generalities about Finite-State Automata (FSA)
  - Regular languages, regular expressions and FSAs
  - Constructing a FSA from a regular expression
  - Non-deterministic FSAs
- **Optimization algorithms for FSAs**
  - *Determinization* of a FSA via subset construction
  - ***Minimization* of a FSA:** equivalence classes, **Brzozowski's algorithm**
- Applications of FSAs & extensions to finite-state transducers
- Conclusions, exercises

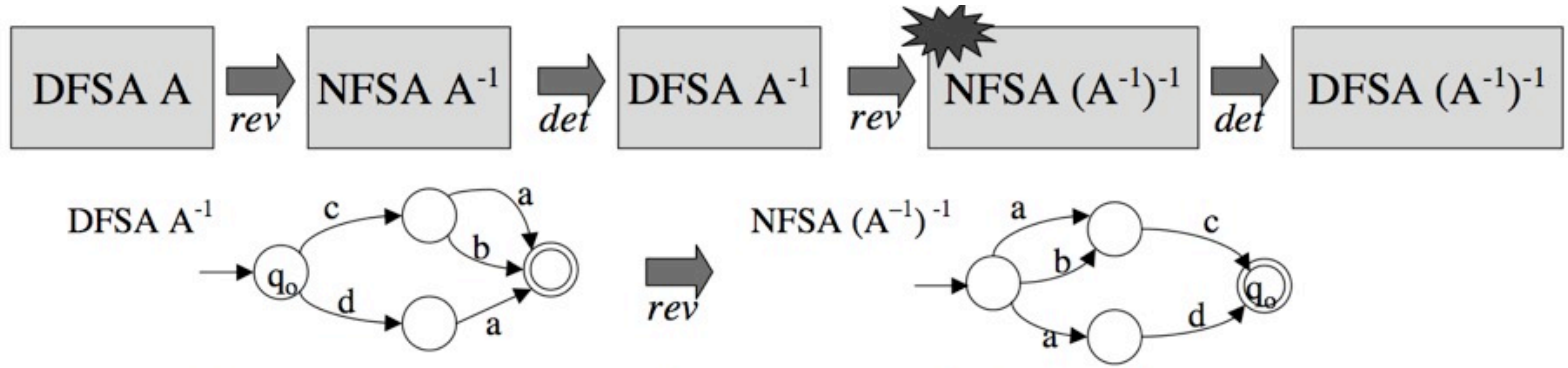
Minimization by reversal and determinization



Reversal

- Final states of A^{-} : set of initial states of A
- Initial state of A^{-} : F of A
- $\delta^{-}(q,a) = \{p \in \Phi \mid \delta(p,a)=q \}$
- $L(A^{-1}) = L(A)^{-1}$

Why does it yield a minimal DFSA?



Consider the *right languages of states q, q‘ in NFSA* $(A^{-1})^{-1}$:

- *If* for all distinct states q, q‘ $L^{\rightarrow}(q) \neq L^{\rightarrow}(q')$, i.e. $L^{\rightarrow}(q) \cap L^{\rightarrow}(q') = \emptyset$, it holds that each pair of states q,q’ recognize *different right languages*, and thus, that the NFSA $(A^{-1})^{-1}$ satisfies the *minimality condition* for a DFSA.
- If there were states q,q’ in NFSA $(A^{-1})^{-1}$ s.th. $L^{\rightarrow}(q) \cap L^{\rightarrow}(q') \neq \emptyset$, there would be some string w that leads to two distinct states in DFSA A^{-1}. This contradicts the *determinicity* criterion of a DFSA.
- Determinization of NFSA $(A^{-1})^{-1}$ does not destroy the property of minimality

Overview of the lecture

- Background
    - Chomsky hierarchy of languages
    - Basic definitions, generic operations on languages
- Generalities about Finite-State Automata (FSA)
    - Regular languages, regular expressions and FSAs
    - Constructing a FSA from a regular expression
    - Non-deterministic FSAs
- Optimization algorithms for FSAs
    - *Determinization* of a FSA via subset construction
    - *Minimization* of a FSA: equivalence classes, Brzozowski‘s algorithm
- **Applications of FSAs & extensions to finite-state transducers**
- Conclusions, exercises

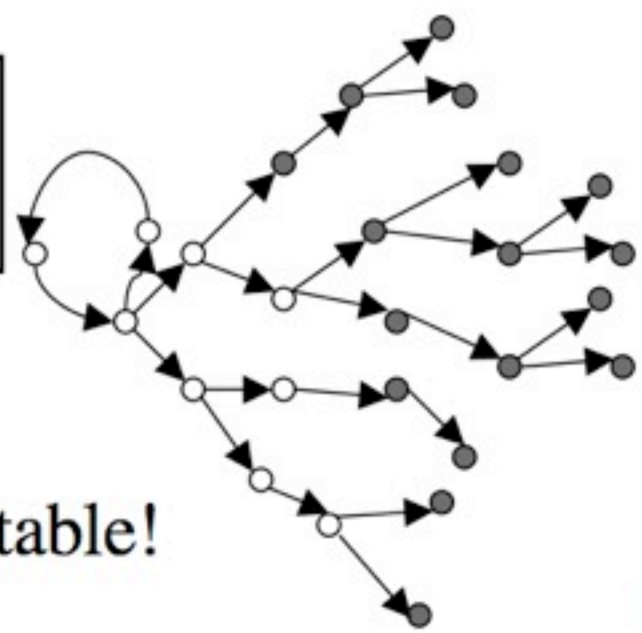
Applications: string matching

- Exact, full string matching
  - Lexicon lookup: search for given word/string in a lexicon
  - Compile lexicon entries to FSA by union
  - Test input words for acceptance in lexicon-FSA

Word list

compile to FSA

transition table!



recognition/application/lookup of input word *w* in/to FSA $A_{lexicon}$:

$(q_0,w) \vdash^*_{Alexicon} (q_f, \varepsilon)$ is true, with q_0 initial state and $q_f \subseteq F$

traversal and recognition (acceptance)

Applications: string matching

- Substring matching
  - Identify stop words in stream of text
  - Stem recognition: *small*, *smaller*, *smallest*
- Make use of full power of finite-state operations!
  - Regular expression with any-symbols for text search
    - ?* small(ε | er | est) ?*
    - ?* (a | the | …) ?*
  - Compilation to NFSA, convert to DFSA
  - Application by *composition* of FST with full text
    - $\mathrm{FSA}_{\text{text stream}}$ .o. $\mathrm{FST}_{\text{small}}$: if defined, search term is substring of text

Applications: replacement

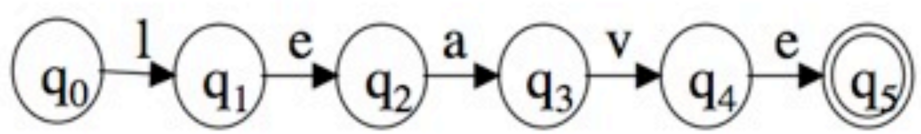
- (Sub)string replacement
  - Delete stop words in text
  - Stemming: reduce/replace inflected forms to stems: *smallest* → *small*
  - Morphology: map inflected forms to lemmas (and PoS-tags): *good*, *better*, *best* → good+Adj
  - Tokenization: insert token boundaries
  - …

  ⇒ Finite-state transducers (FST)

From automata to transducers

Automata

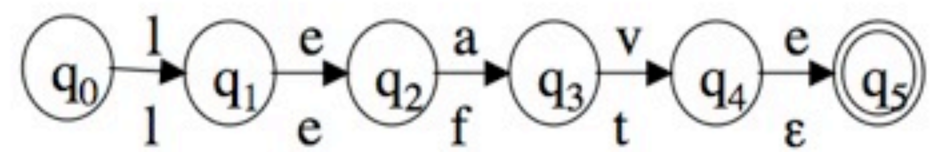
- recognition of an input string *w*



- define a *language*
- accept *strings*, with transitions defined for *symbols* ∈Σ

Transducers

- recognition of an input string *w*
- *generation* of an output string *w*‘



- define a *relation* between languages
- equivalent to FSAs that accept *pairs of strings*, with transitions defined for pairs of symbols <*x*,*y*>
- operations: *replacement*
  - deletion <a, ε>, a ∈Σ-{ε}
  - insertion <ε, a>, a ∈Σ-{ε}
  - substitution <a, b>, a,b ∈Σ, a ≠ b

Overview of the lecture

- Background
  - Chomsky hierarchy of languages
  - Basic definitions, generic operations on languages
- Generalities about Finite-State Automata (FSA)
  - Regular languages, regular expressions and FSAs
  - Constructing a FSA from a regular expression
  - Non-deterministic FSAs
- Optimization algorithms for FSAs
  - *Determinization* of a FSA via subset construction
  - *Minimization* of a FSA: equivalence classes, Brzozowski's algorithm
- Applications of FSAs & extensions to finite-state transducers
- **Conclusions, exercises**

Conclusions

- In this lecture, we presented finite-state automata and their algorithms
  - FSAs and regular expressions have the same expressive power: they both define a regular language, type-3 in the Chomsky hierarchy
  - FSAs can be automatically constructed from a given regular expression
  - FSAs can be deterministic or non-deterministic
- We also saw two algorithms used to improve the (runtime) efficiency of a finite-state automata:
  - FSA determinization, via subset construction
  - FSA minimization, via either equivalence classes, or Brzozowski
- Finite-state automata can be extended to finite-state transducers, which define *relations* between languages
- Due to their simplicity and efficiency, FSAs are pervasive in computational linguistics (morphology, parsing, dialogue management, etc.)

Exercises

1. Write a program for acceptance of a string by a DFSA.
   Then extend it to a finite-state transducer that can translate a surface form to lemma + POS, or between upper and lower case.
2. Determinize the following NFSA by subset construction.
   A_1=<{p,q,r,s},{a,b},δ_1,p,{s}> where δ_1 is as follows:

δ_1	a	b
p	p,q	p
q	r	r
r	s	-
s	s	s

3. Construct an NFSA with ε-transitions from the regular expression (a|b)ca*, according to the construction principled for union, concatenation and kleene star.
   Then transform the NFSA to a DFSA by subset construction.
4. Find a minimal DFSA for the FSA A=<{A,..,E},{0,1}, δ_3,A,{C,E}> (using the table filling algorithm by propagation).

δ_3	0	1
A	B	D
B	B	C
C	D	E
D	D	E
E	C	-