Computational Linguistics:
Part 1: Finite-State Automata
(exercise session)

Pierre Lison

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

Practical announcements

- The usual:
  - Website is still at the same place:
    - http://www.dfki.de/~plison/lectures/compling2010/index.html
  - If you haven't done it yet, please subscribe to the mailing list!
- Regarding the timetable for the exercise session:
  - Seems to have an agreement among us to have it on Thursday, 16-18
  - No strong objections for the lecturers at the moment
  - **But**: the Computerlinguistik Kolloquium also takes place at that time!
  - Need to find a solution agreeable to everyone...
- Please don't wait for months to register the course in the HISPOS database (if you plan to take it, needless to say)
- Note regarding the exam: due to the participation of several lecturers in the course, It would be better if you could all take the written exam
  - But if you really need to have an oral exam for this course, let me know ASAP

Outline of this exercise session

- Short recap‘
- Correction of the exercises
- Advanced topics:
  - ~~Finite-state transducers for morphological parsing~~
  - Weighted finite-state automata
  - Cascading finite-state transducers

Outline of this exercise session

- **Short recap‘**
- Correction of the exercises
- Advanced topics:
    - ~~Finite-state transducers for morphological parsing~~
    - Weighted finite-state automata
    - Cascading finite-state transducers

Recap from last lecture

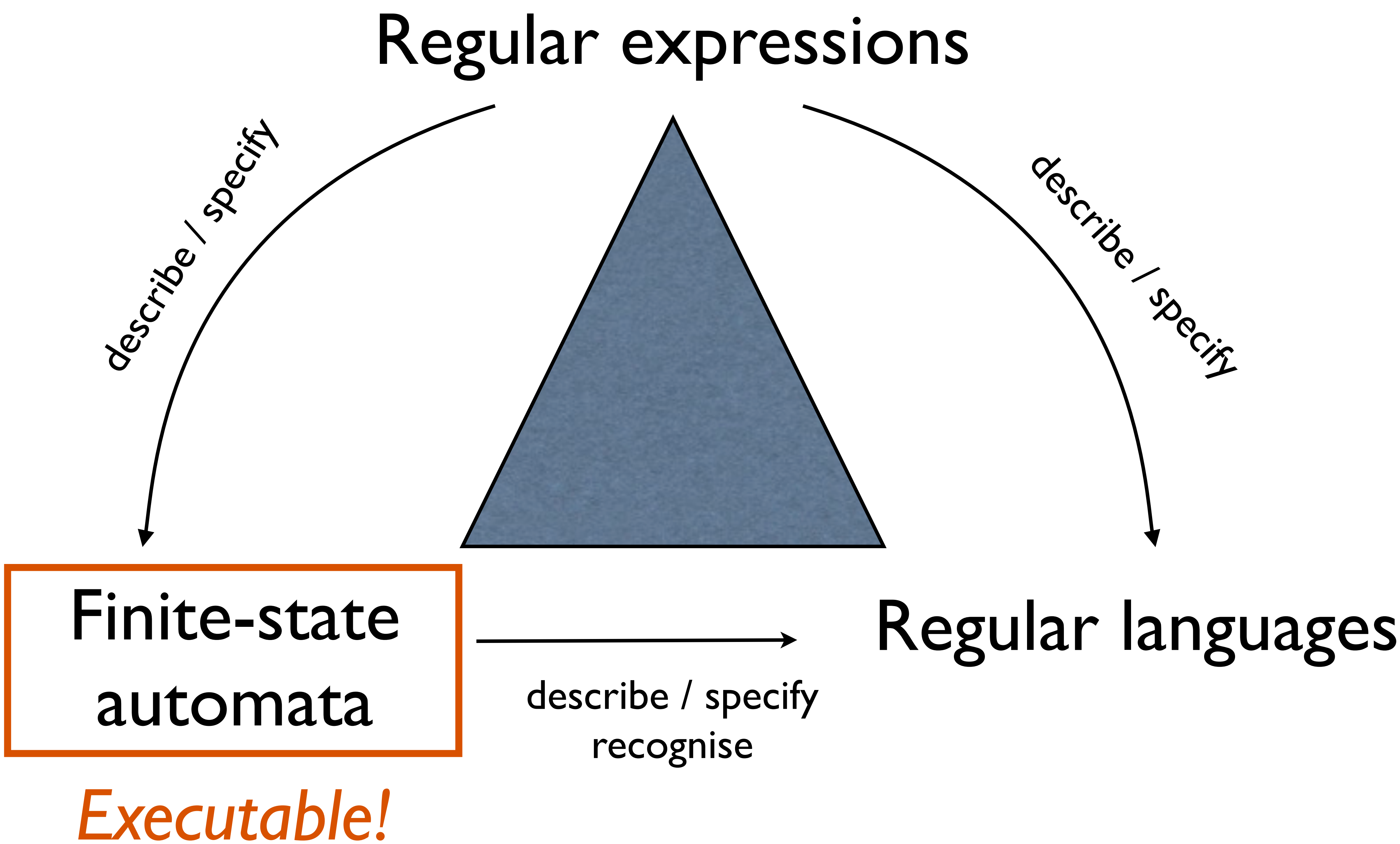
- In the last lecture, we presented finite-state automata and their algorithms
  - FSAs and regular expressions have the same expressive power: they both define a regular language, type-3 in the Chomsky hierarchy
  - FSAs can be automatically constructed from a given regular expression
  - FSAs can be 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.)

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

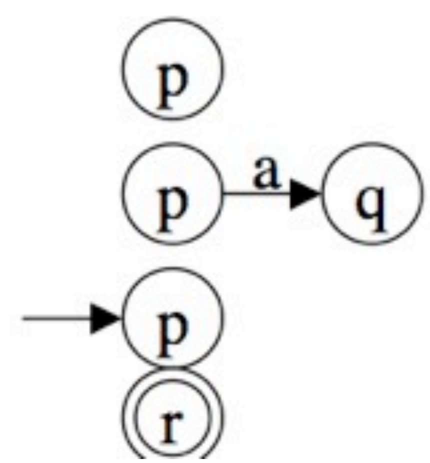
More expressivity
Less computational efficiency

FSAs and regular expressions



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(ıs (diagrams):
  - states: circles
  - transitions: directed arcs between circles
  - initial state
  - final state



$p \in \Phi$

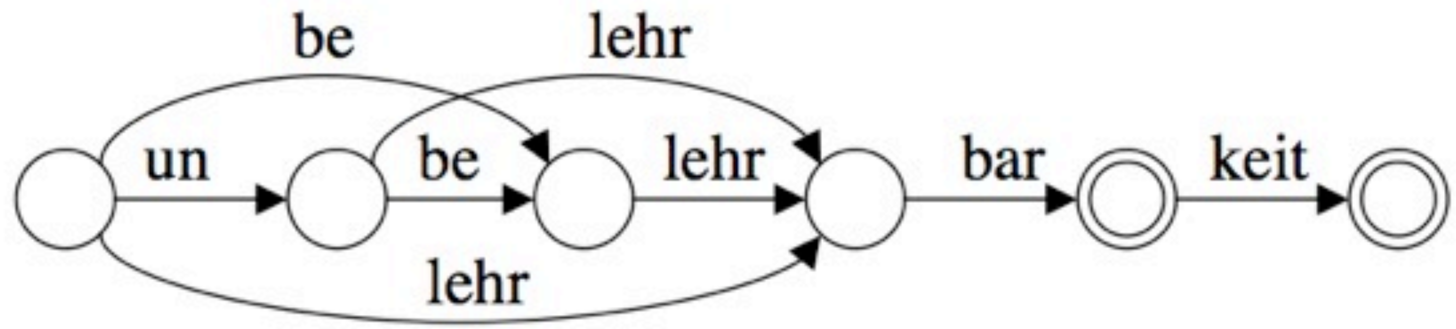
$\delta(p, a) = q$

$p = q_0$

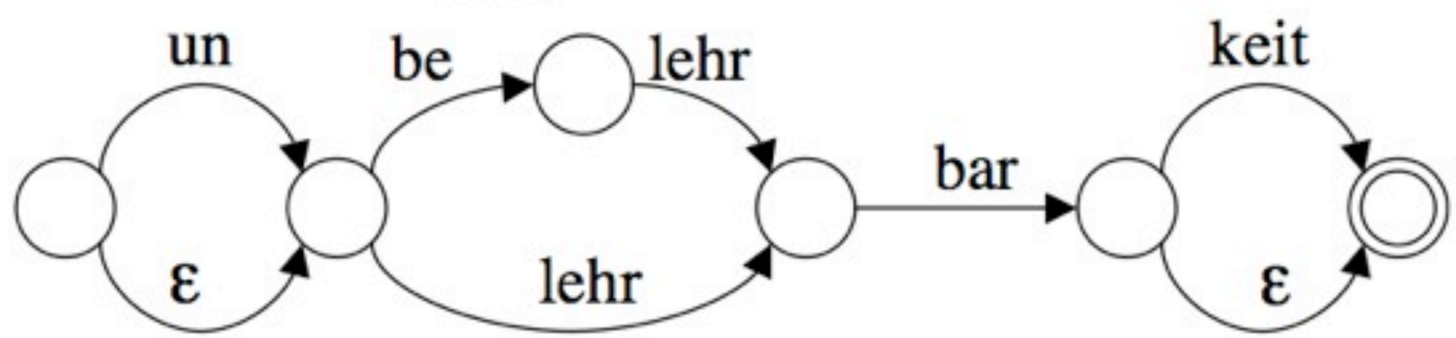
$r \subseteq F$

Multiple equivalent FSAs

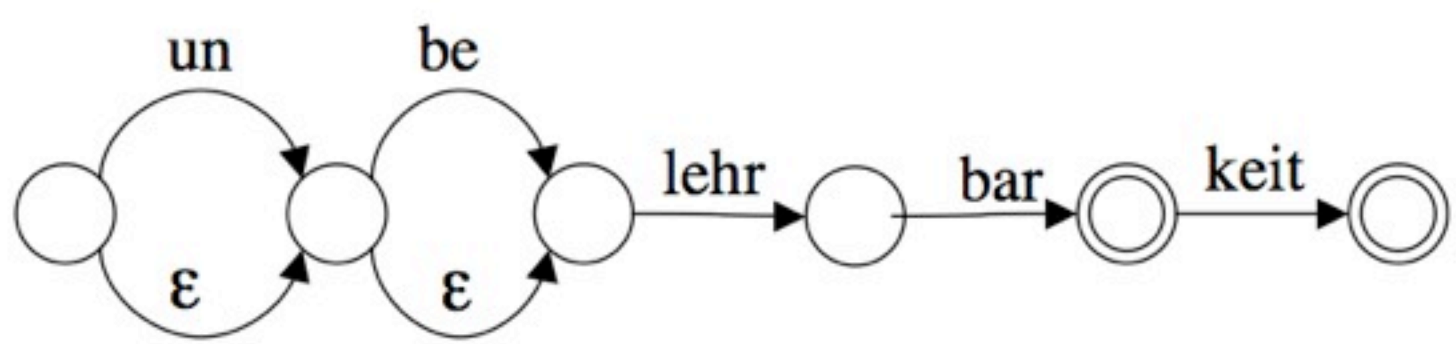
- FSA for the language L_{lehr} = { *lehrbar*, *lehrbarkeit*, *belehrbar*, *belehrbarkeit*, *unbelehrbar*, *unbelehrbarkeit*, *unlehrbar*, *unlehrbarkeit* }
- DFSA for L_{lehr}



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

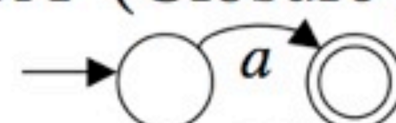


- Equivalent FSA (non-deterministic)

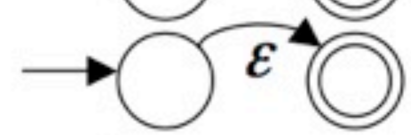


Defining FSAs through regexps

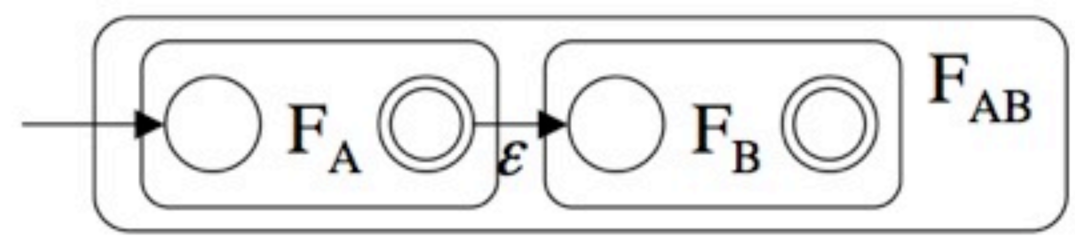
- FSAs for even mildly complex regular languages are best constructed from regular expressions!
- Every regular expression denotes a regular language
  - $L(\varepsilon) = \{\varepsilon\}$
  - $L(a) = \{a\}$ for all $a \in \Sigma$
  - $L(\alpha\beta) = L(\alpha)L(\beta)$
  - $L(\alpha \cup \beta) = L(\alpha)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

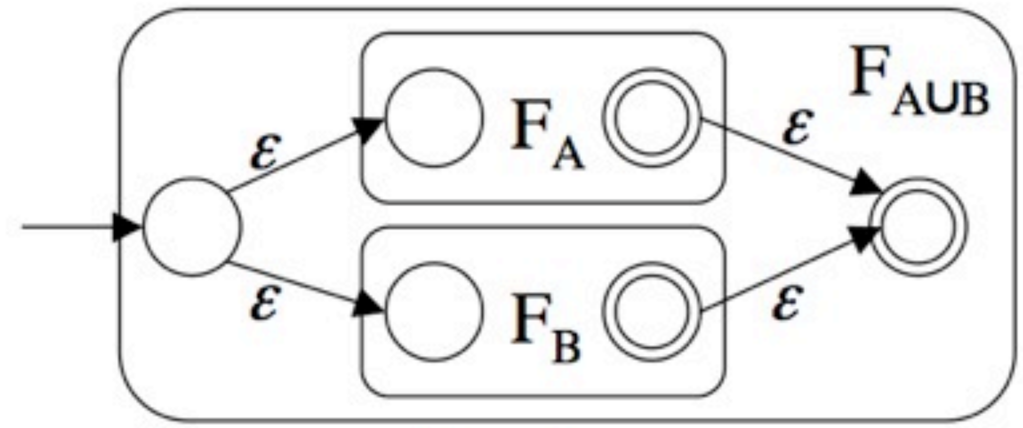


  - Concatenation of two FSAs F_A and F_B:

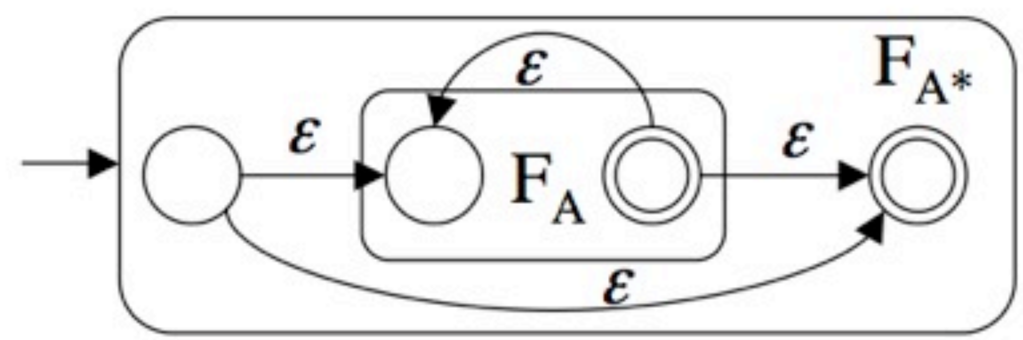


    - $S_{AB} = S_A$ (S initial state)
    - $F_{AB} = F_B$ (F set of final states)
    - $\delta_{AB} = \delta_A \cup \delta_B \cup \{\delta(<q_i,\varepsilon>,q_j) \mid q_i \in F_A,\ q_j = S_B\}$

- 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)
  - $\delta_{AB} = \delta_A \cup$
    $\cup \{\delta(<q_j,\varepsilon>,q_z) \mid q_j \in F_A, q_z = S_A)\}$
    $\cup \{\delta(<q_0,\varepsilon>,q_z) \mid q_0 = S_{A*}, (q_z = S_A \text{ or } q_z = F_{A*})\}$
    $\cup \{\delta(<q_z,\varepsilon>,q_j) \mid q_z \in F_A , q_j \in F_{A*}\}$



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 $\langle q_{source}, z, q_{target} \rangle$
    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 $A = \langle \Phi, \Sigma, \delta, q_0, F \rangle$
  - Φ a finite non-empty set of states
  - $\mathbf{\Sigma}$ a finite alphabet of input letters
  - δ a transition function $\Phi \times \mathbf{\Sigma}^* \rightarrow \wp(\Phi)$ (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 $w_i = z w_{i+1}$ 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!

Equivalence of DFSAs and NFSAs

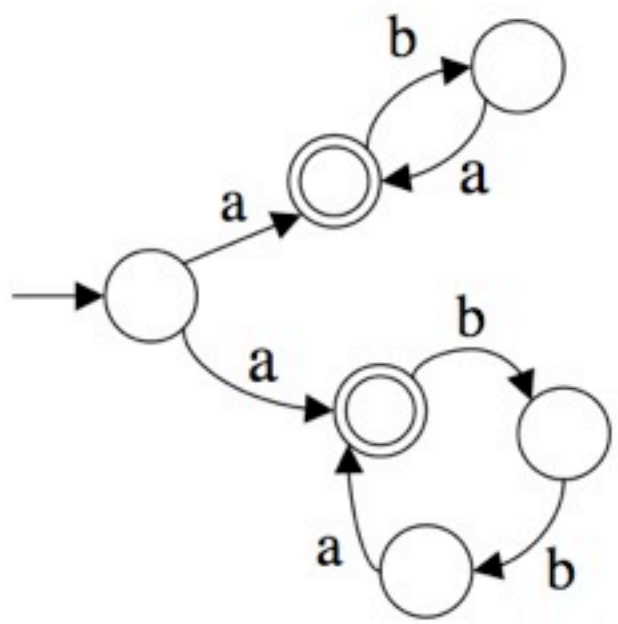
- Despite non-determinism, NFSAs are not more powerful than DFSAs: they accept the same class of languages: regular languages
- For every non-deterministic FSA there is deterministic FSA that accepts the same language (and vice versa)
  - The corresponding DFSA has in general more states, in which it 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

Determin. by subset construction

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



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

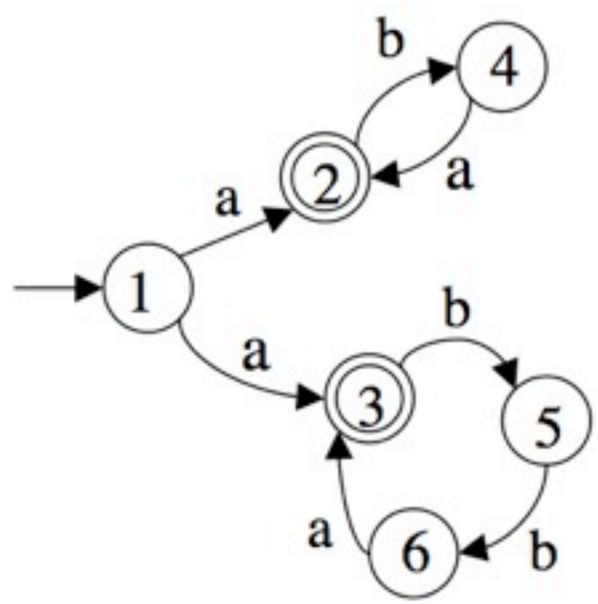
$A' = <\Phi', \Sigma, \delta', q_0', F'>$

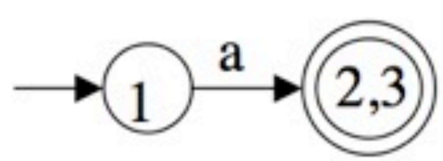
Subset construction:

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

Determin. by subset construction

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





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

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

$q_0'=\{1\}$,

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

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

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

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

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

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

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

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

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

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

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

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

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

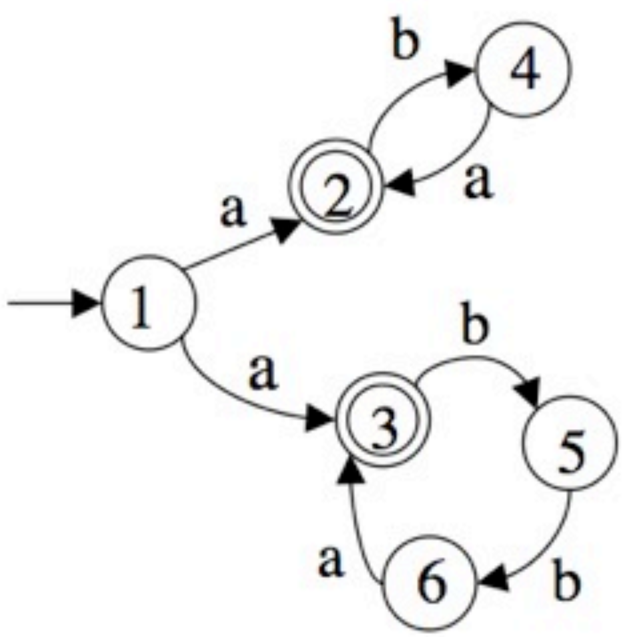
$\delta'(\{3\},b)= \{5\}$,

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

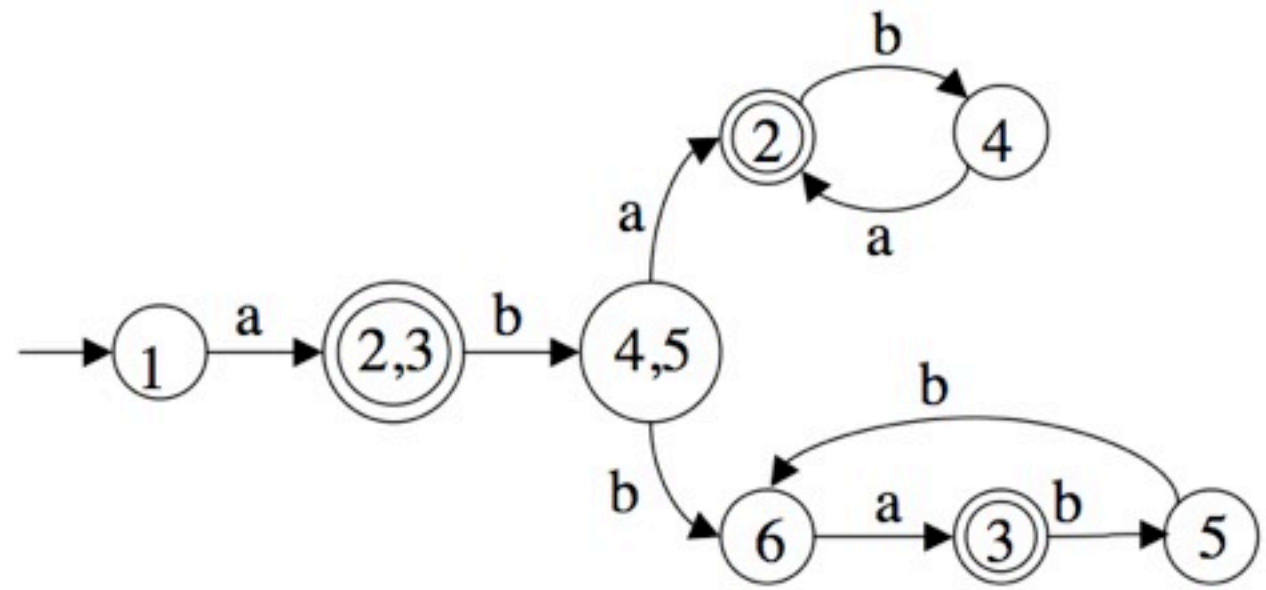
$\delta'(\{5\},b)= \{6\}$

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

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



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



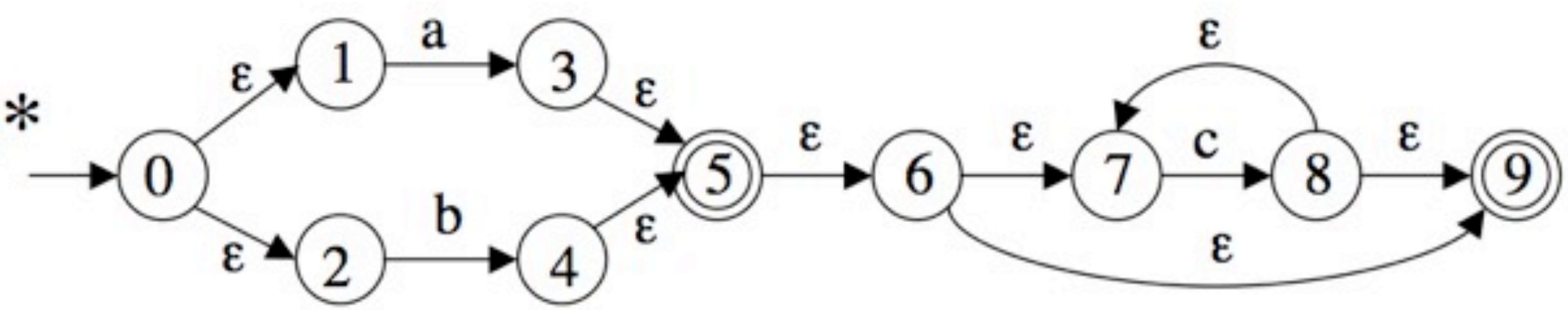
$\boxed{L(A) = L(A') = a(ba)^* \cup a(bba)^*}$

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 $R \subset \Phi$)

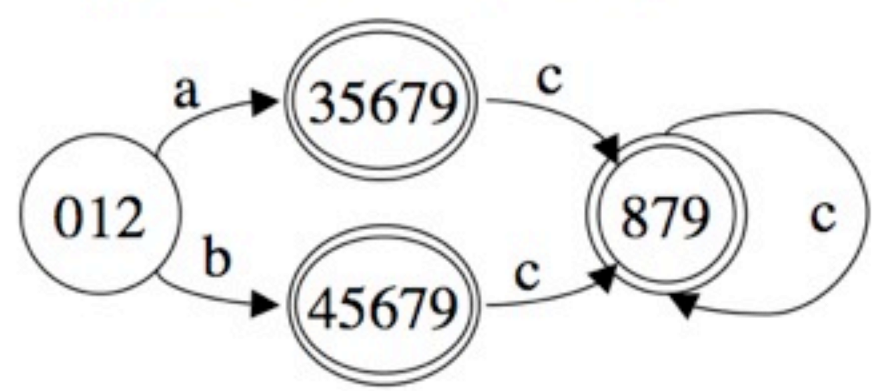
- Subset construction for ε-NFSAs
  - Compute δ' from δ for all subsets $S \subseteq \Phi$ and $a \in \Sigma$ s.th.
    $\delta'(S,a) = \{ s'' \mid \exists s \in S \text{ s.th. } (s,a,s') \in \delta \text{ and } s'' \in \varepsilon\text{-closure}(s') \}$

- 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}

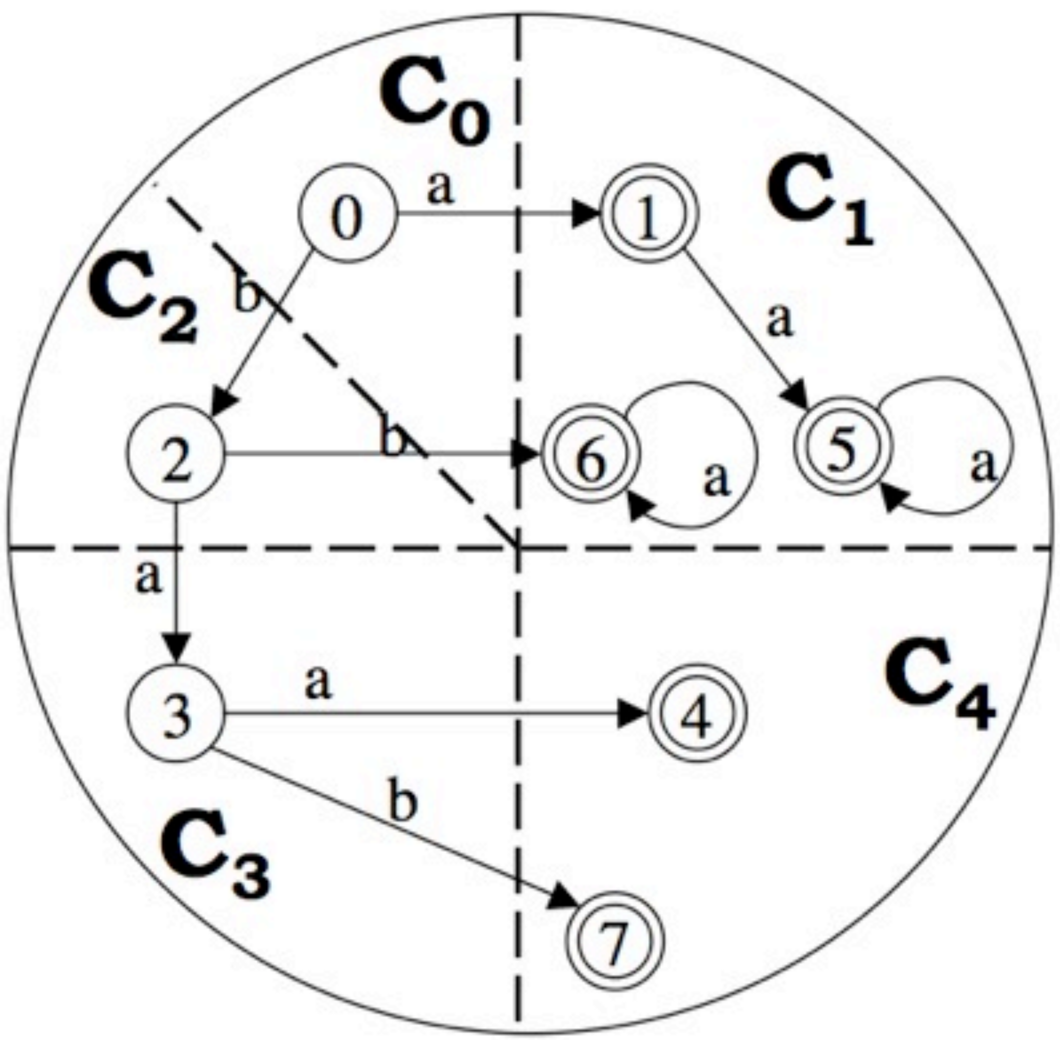


Minimization of FSAs

- 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 $q,q' \in Q_i$

Partitioning in equivalence classes



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

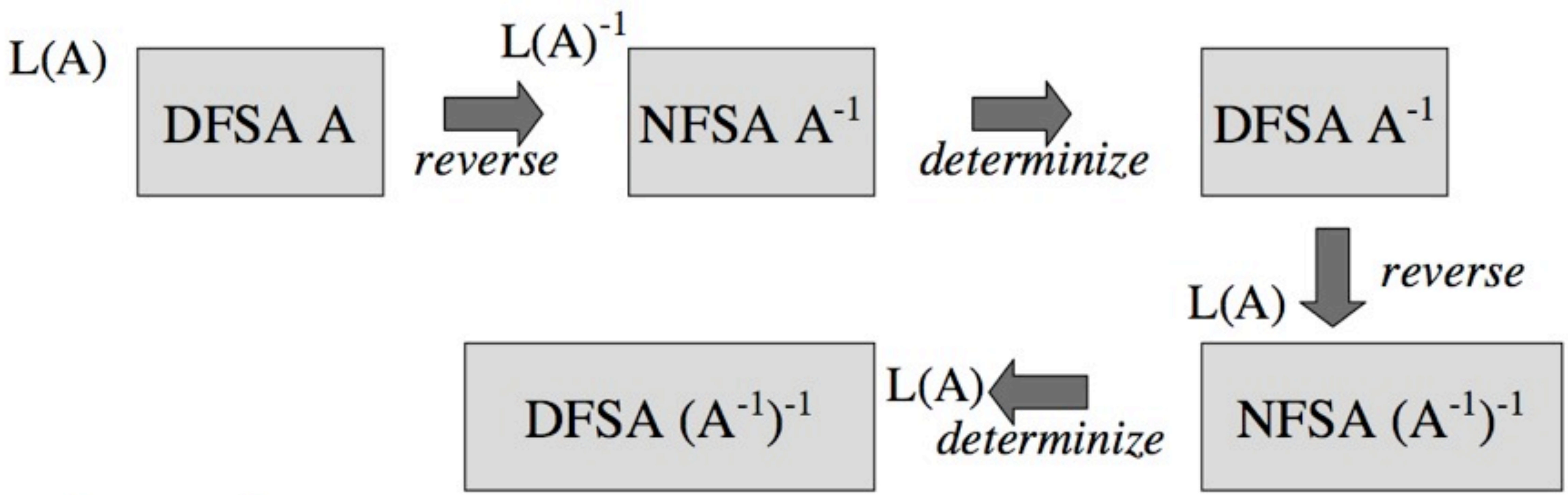
Minimization of a DFSA

A DFSA $A=\langle\Phi,\Sigma,\delta,q_0,F\rangle$ that contains *equivalent states q, q'* can be transformed to a smaller, equivalent DFSA $A'=\langle\Phi',\Sigma,\delta',q_0,F'\rangle$ 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 $A=\langle\Phi,\Sigma,\delta,q_0,F\rangle$ is *minimal* iff there is no pair of distinct but equivalent states $\in\Phi$, i.e. $\forall q, q' \in \Phi: \; q \equiv q' \Leftrightarrow q = q'$

Minimization by reversal and determinization



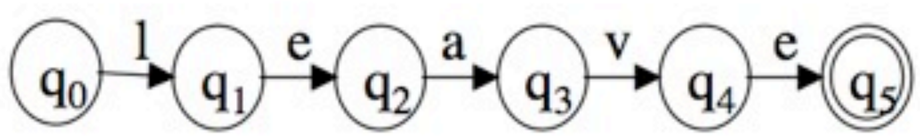
Reversal

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

From automata to transducers

Automata

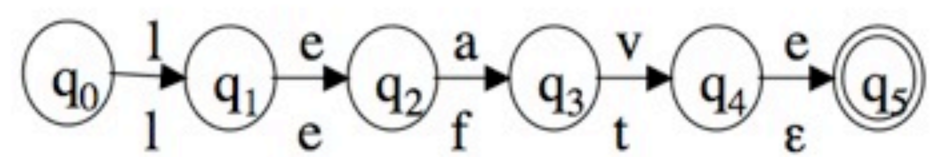
- recognition of an input string *w*



- 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

Outline of this exercise session

- Short recap‘
- **Correction of the exercises**
- Advanced topics:
  - ~~Finite-state transducers for morphological parsing~~
  - Weighted finite-state automata
  - Cascading finite-state transducers

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.
   $A_1=\langle\{p,q,r,s\},\{a,b\},\delta_1,p,\{s\}\rangle$ 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 $A=\langle\{A,..,E\},\{0,1\},\delta_3,A,\{C,D,E\}\rangle$ (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	-

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,D,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	-

Small Python class for DFSA

```
class DFSA:
    states = []                 # List of states
    startState = None           # Starting state
    endStates = []              # Ending states
    transFunction = {}          # Transition function (defined as a dictionary)

    ....        # Initialisation functions

    # Returns the next state if one can be reached from the given state and symbol, or None otherwise
    def getTransition(self, state, symbol):
        if self.transFunction.has_key((state,symbol)):
            return self.transFunction[(state,symbol)]
        else:
            return None

    # Returns true if the string can be recognized by the DFSA, false otherwise
    def isRecognized(self,string):
        return self.isRecognizedFromState(self.startState,string):

    # Returns true if the current string can be recognized by the DFSA starting at curState, false otherwise
    def isRecognizedFromState(self,curState,curString):
        firstSymbol = curString[0]
        stringTail = curString[1:len(curString)]
        nextState = self.getTransition(curState,firstSymbol)
        if nextState == None:
            return False
        elif nextState in self.endStates:
            return True
        else:
            return self.isRecognizedFromState(nextState,stringTail)     # Recursive call
```

Small Python class for DFST

```
class DFST(FSA):
    ....

    ....        # Initialisation functions

    # Returns the next state and output symbol if reachable from state+symbol, return None otherwise
    def getTransition(self, state, symbol):
        if self.transFunction.has_key((state,symbol)):
            return self.transFunction[(state,symbol)]
        else:
            return None

    # Returns the output string if the input string can be transduced by the DFST, or None otherwise
    def transduce(self,string):
        return self.transduceFromState(self.startState,string)

    # Returns output string if the input can be transduced starting at curState, None otherwise
    def transduceFromState(self,curState,curString):
        firstSymbol = curString[0]
        stringTail = curString[1:len(curString)]
        transduction = self.getTransition(curState,firstSymbol)
        if transduction==None:
            return None
        else:
            nextState = transduction[0]
            output = transduction[1]
            if nextState in self.endStates:
                return output
            else:
                nextResult = self.transduceFromState(nextState,stringTail)   # Recursive call
                if nextResult != None:
                    return output+nextResult  # Concatenate the output string
                else:
                    return None
```

Exercises

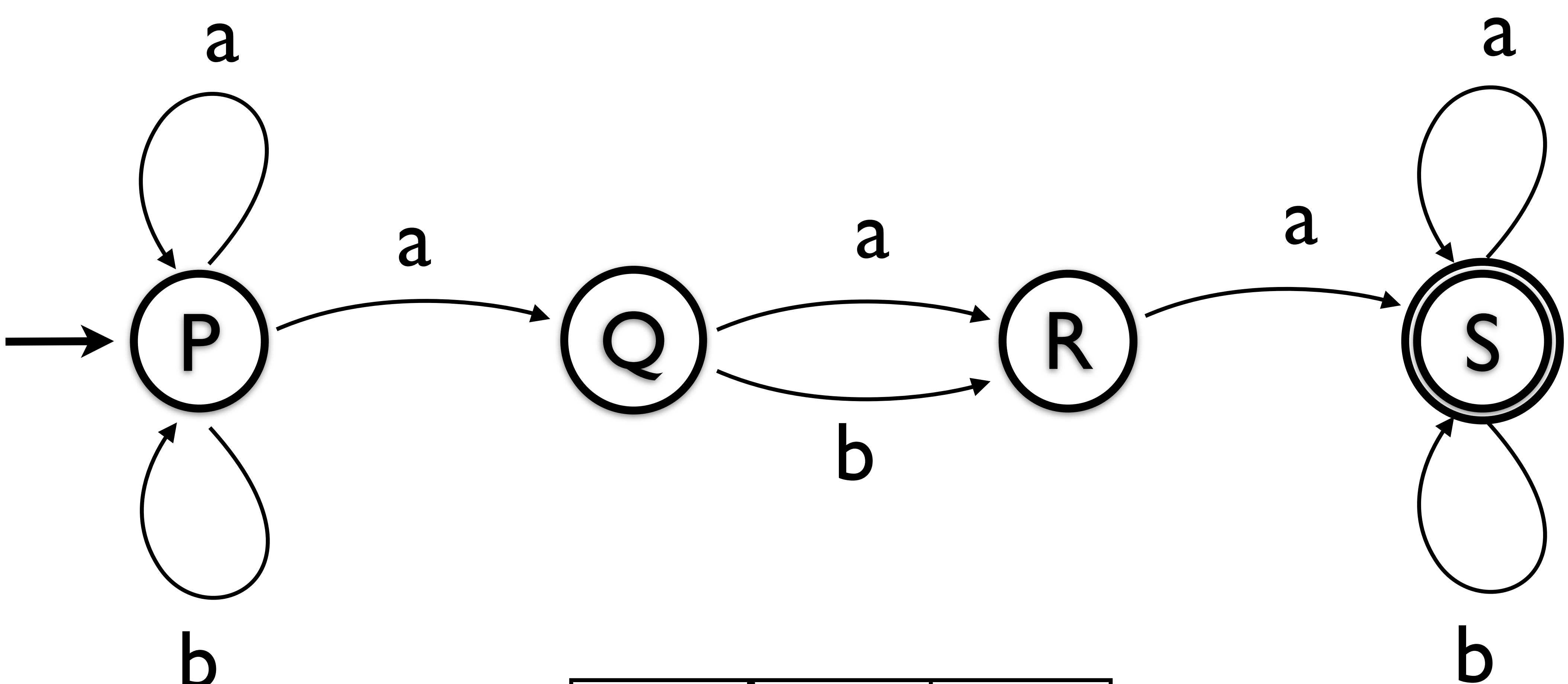
1. Write a program for acceptance of a string by a DFSA.
   Then extend it to a finite-state transducer that can translate a surface form to lemma + POS, or between upper and lower case.
2. Determinize the following NFSA by subset construction.
   $A_1=<\{p,q,r,s\},\{a,b\},\delta_1,p,\{s\}>$ where δ_1 is as follows:

δ_1	a	b
p	p,q	p
q	r	r
r	s	-
s	s	s

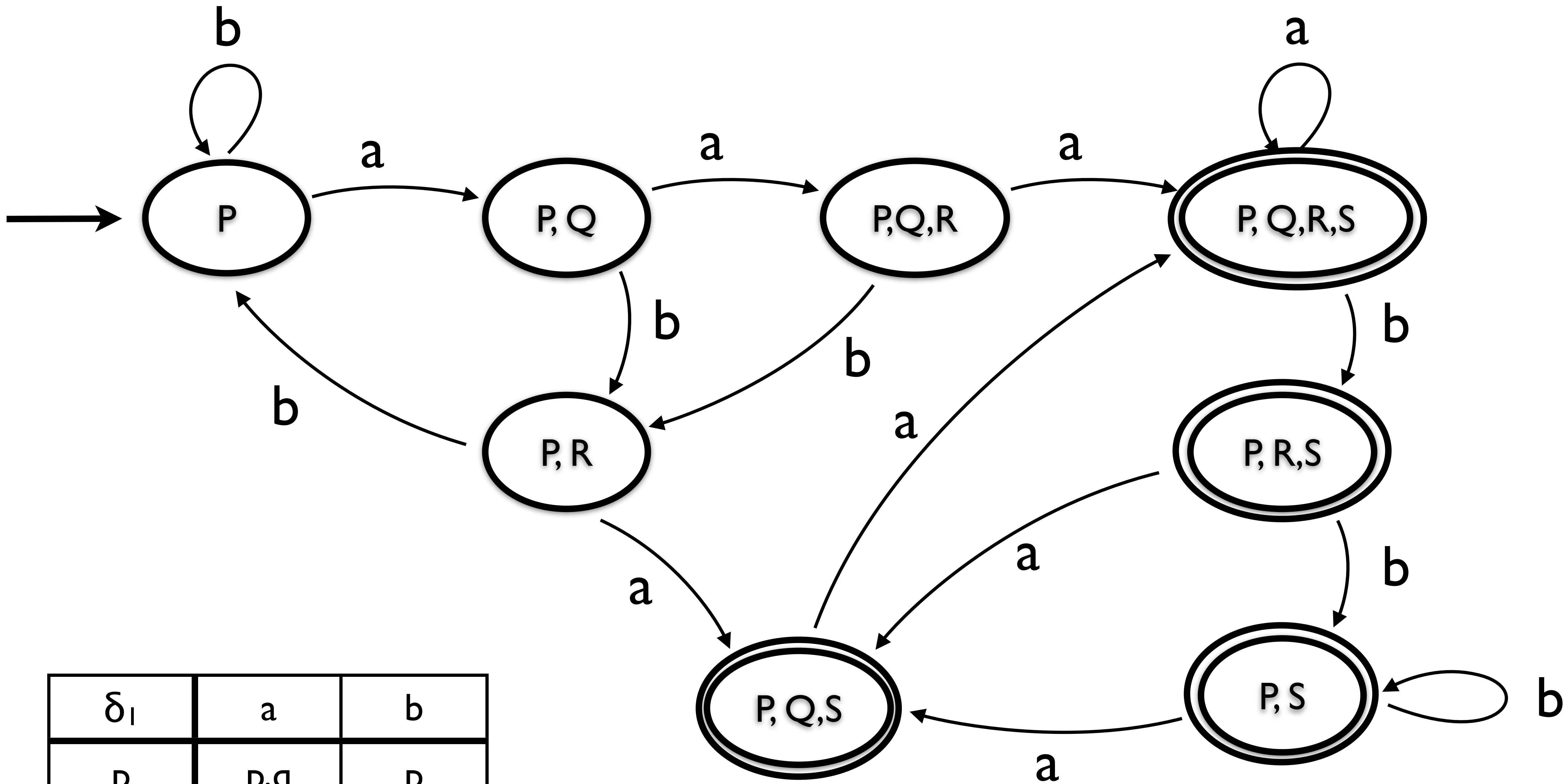
3. Construct an NFSA with ε-transitions from the regular expression (a|b)ca*, according to the construction principled for union, concatenation and kleene star.
   Then transform the NFSA to a DFSA by subset construction.
4. Find a minimal DFSA for the FSA $A=<\{A,..,E\},\{0,1\},\delta_3,A,\{C,D,E\}>$ (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	-

Determinization



δ₁	a	b
p	p,q	p
q	r	r
r	s	-
s	s	s



δ_1	a	b
p	p,q	p
q	r	r
r	s	-
s	s	s

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

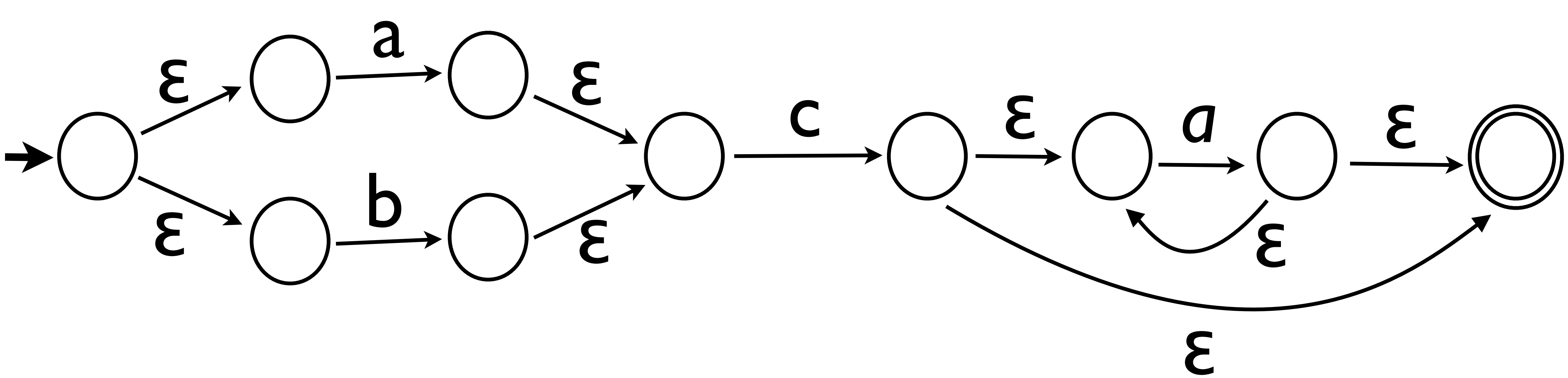
3. Construct an NFSA with ε-transitions from the regular expression (a|b)ca*, according to the construction principled for union, concatenation and kleene star. Then transform the NFSA to a DFSA by subset construction.
4. Find a minimal DFSA for the FSA A=<{A,..,E},{0,1}, δ_3,A, {C,D,E}> (using the table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

δ_3	0	1
A	B	D
B	B	C
C	D	E
D	D	E
E	C	-

The regular expression:
(a|b)ca*

union of two FSA

Kleene Star over FSA

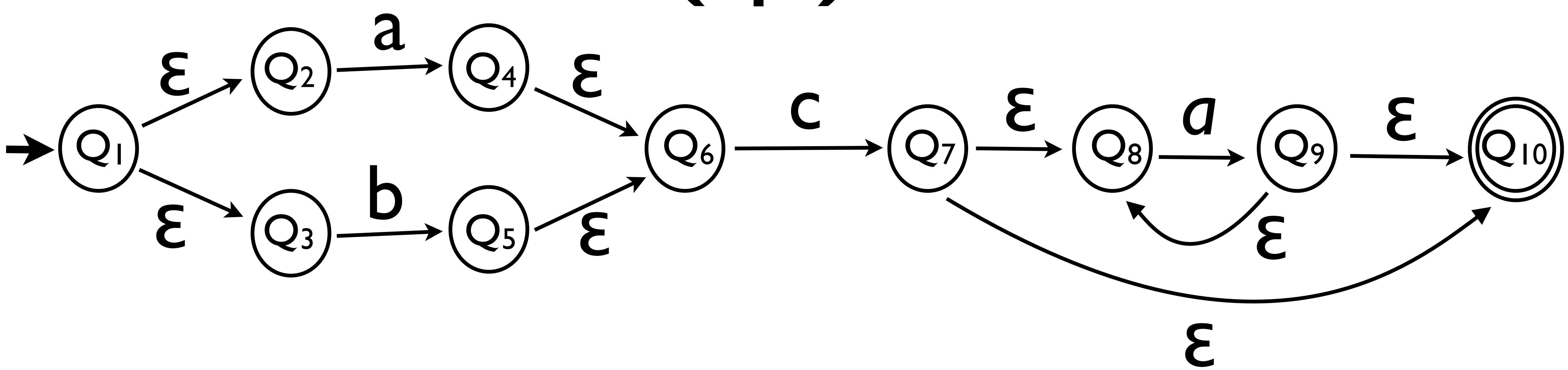


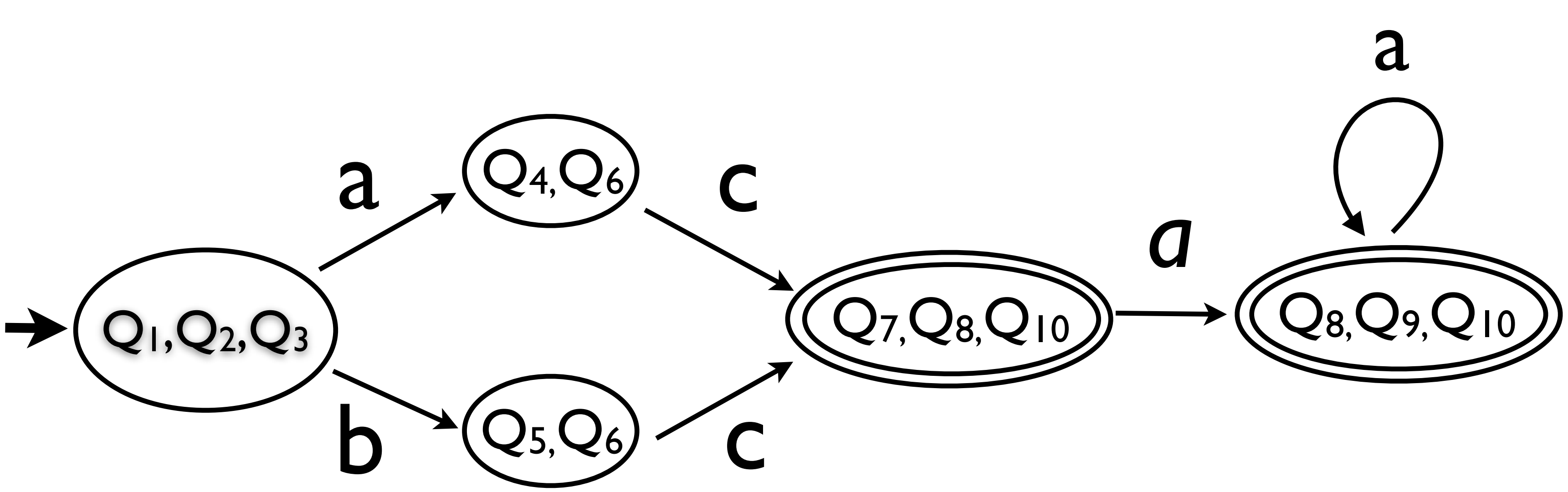
But this is a non-deterministic FSA...

Constructing a FSA from a regexp

The regular expression:

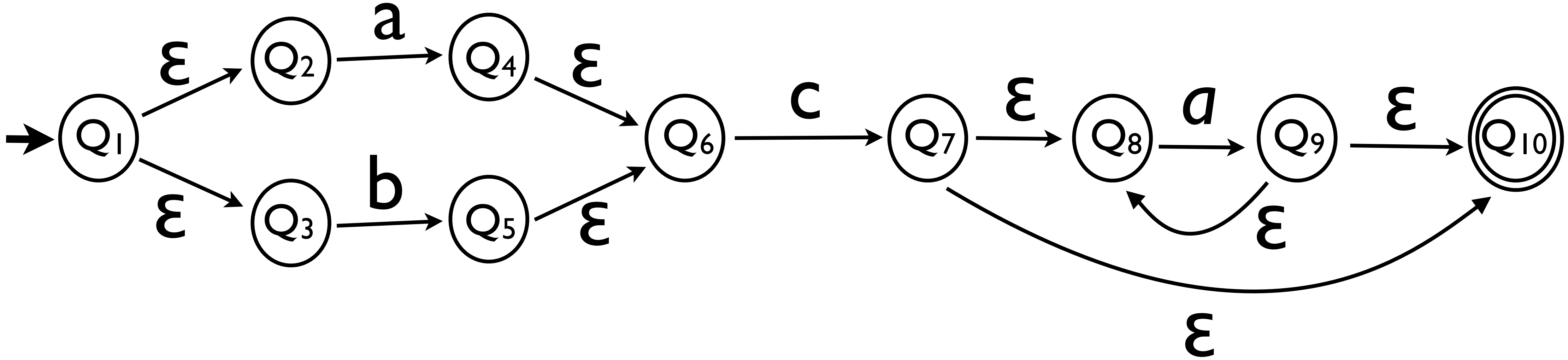
(a|b)ca*



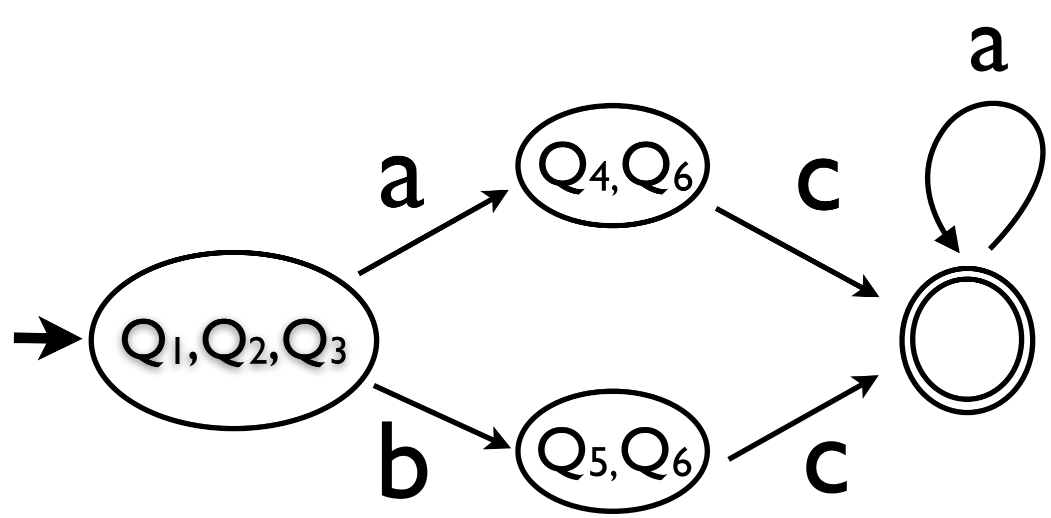


The regular expression:

(a|b)ca*



Or even simpler, by minimisation:



Exercises

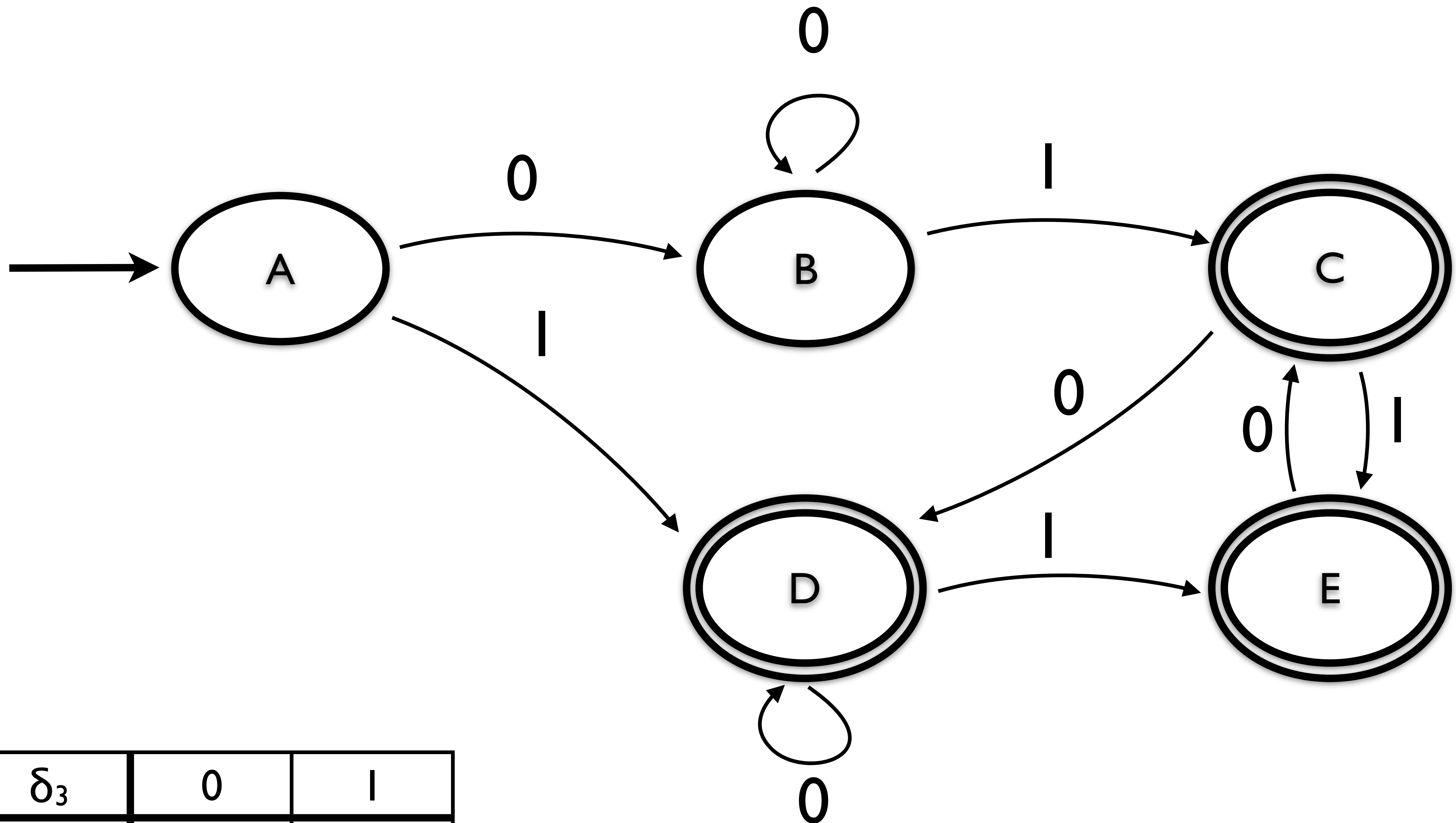
1. Write a program for acceptance of a string by a DFSA.
   Then extend it to a finite-state transducer that can translate a surface form to lemma + POS, or between upper and lower case.
2. Determinize the following NFSA by subset construction.
   A_1=<{p,q,r,s},{a,b},δ_1,p,{s}> where δ_1 is as follows:

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,D,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	-

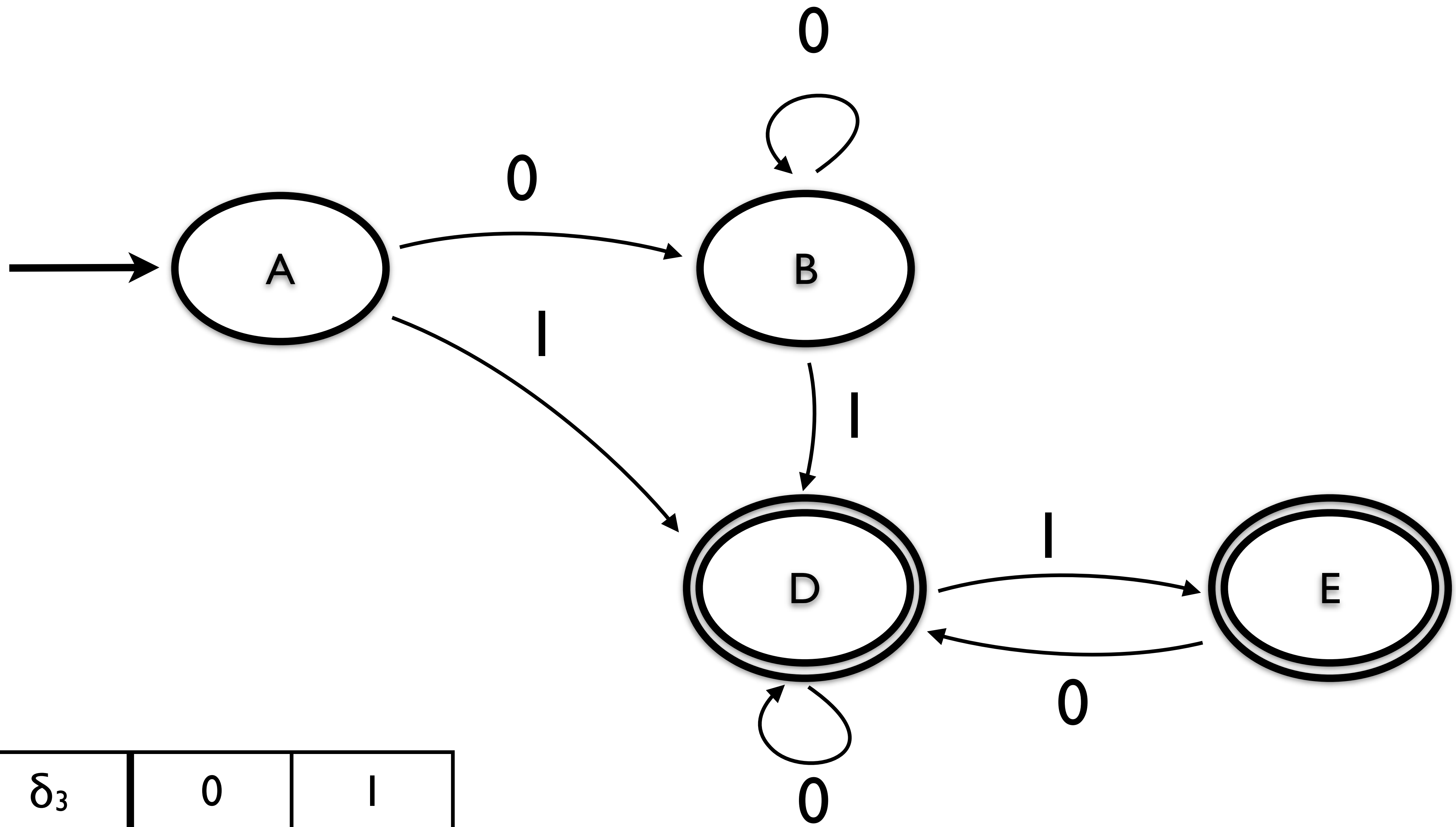
FSA minimization



δ_3	0	1
A	B	D
B	B	C
C	D	E
D	D	E
E	C	

C and D have the same right language:
0*|(0*(10)*)*|(0*(10)*1)*

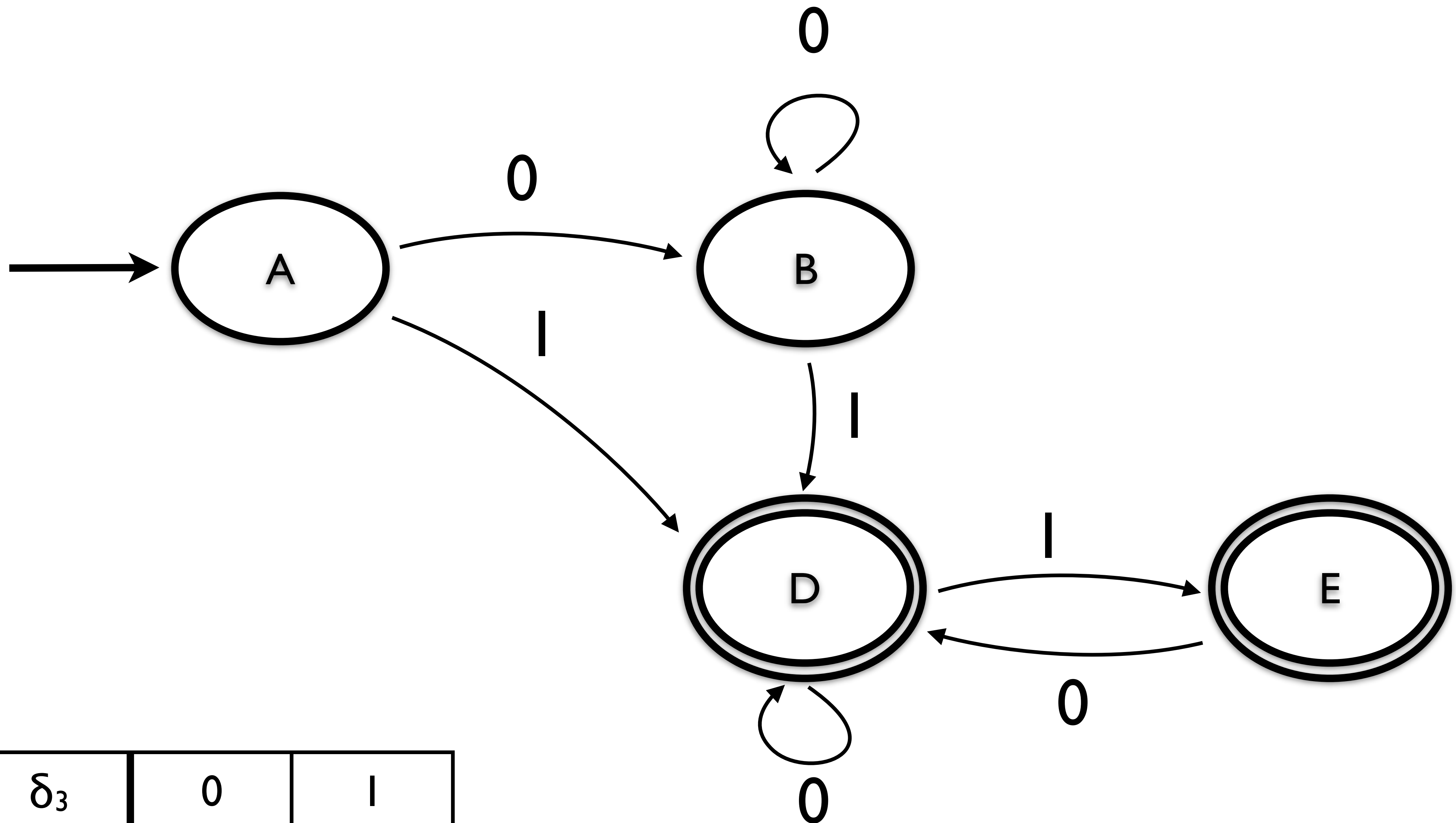
FSA minimization



δ_3	0	1
A	B	D
B	B	D
D	D	E
E	D	

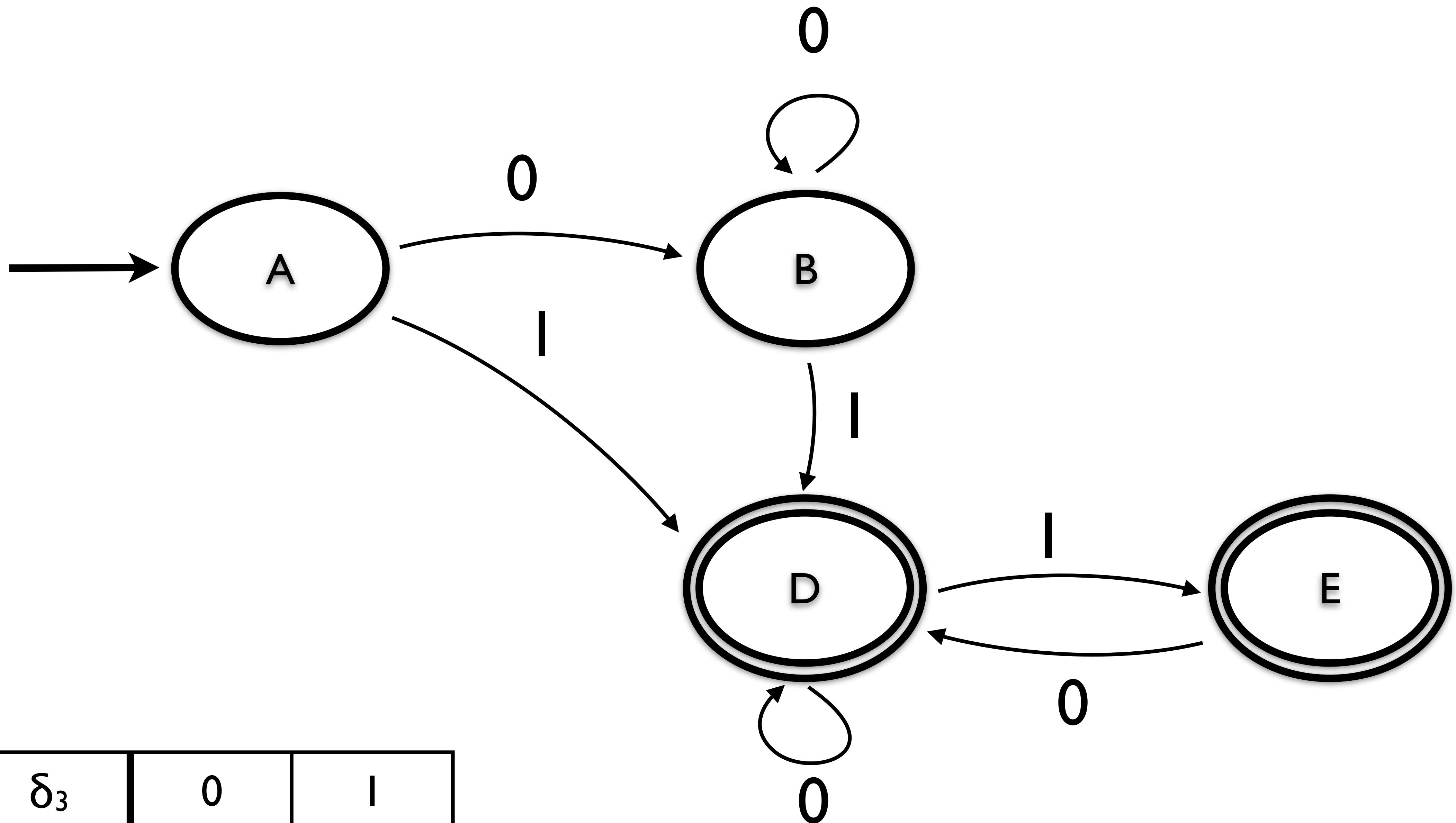
We can thus remove C and redirect all its incoming edges to D

FSA minimization



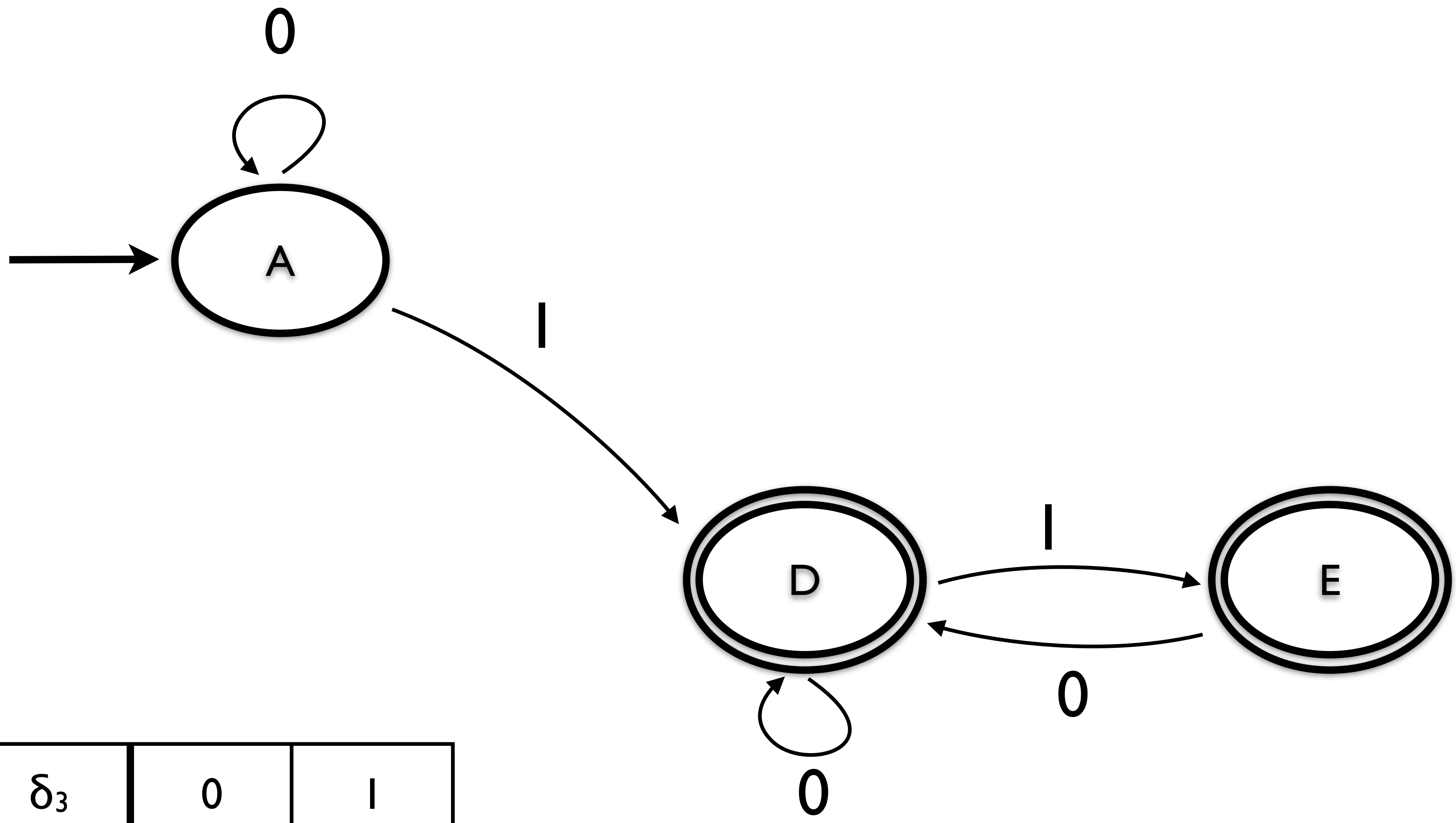
δ_3	0	1
A	B	D
B	B	D
D	D	E
E	D	

We can thus remove C and redirect all its incoming edges to D



δ_3	0	1
A	B	D
B	B	D
D	D	E
E	D	

A and B also have the same right language:
0*1+right language of D



δ_3	0	1
A	A	D
D	D	E
E	D	

And we‘re done!

Outline of this exercise session

- Short recap‘
- Correction of the exercises
- **Advanced topics:**
  - ~~Finite-state transducers for morphological parsing~~
  - **Weighted finite-state automata**
  - **Cascading finite-state transducers**

(Following slides adapted on existing slides by Fei Xia)

- Probabilistic finite-state automata is a generalisation of classical finite-state automata
- Also called: Weighted FSA
- Allow us to provide an explicit account the **uncertainty** of our observations / of our model
- Plus, probabilistic models can often be combined with machine learning techniques to automatically *train* the model from data
    - instead of specifying it manually

- In a probabilistic finite-state automata, each arc is associated with a probability.
- The probability of a path is the multiplication of the arcs on the path.
- The probability of a string x is the sum of the probabilities of all the paths for x.
- Possible tasks :
  - Given a string x, find the best path for x.
  - Given a string x, find the probability of x in a PFA.
  - Find the string with the highest probability in a PFA
  - etc.

Formal definition of a PFSA

- A PFA is
  - Q: a finite set of N states
  - Σ: a finite set of input symbols
  - I: Q ➔ R^+ (initial-state **probabilities**)
  - F: Q ➔ R^+ (final-state **probabilities**)
  - $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$: the transition relation between states.
  - P: δ ➔ R^+ (transition probabilities)

Normalised probabilities

Normalisation constraints on function:

$$\sum_{q \in Q} I(q) = 1$$

$$\forall q \in Q : F(q) + \sum_{a \in \Sigma \wedge q' \in Q} P(q, a, q') = 1$$

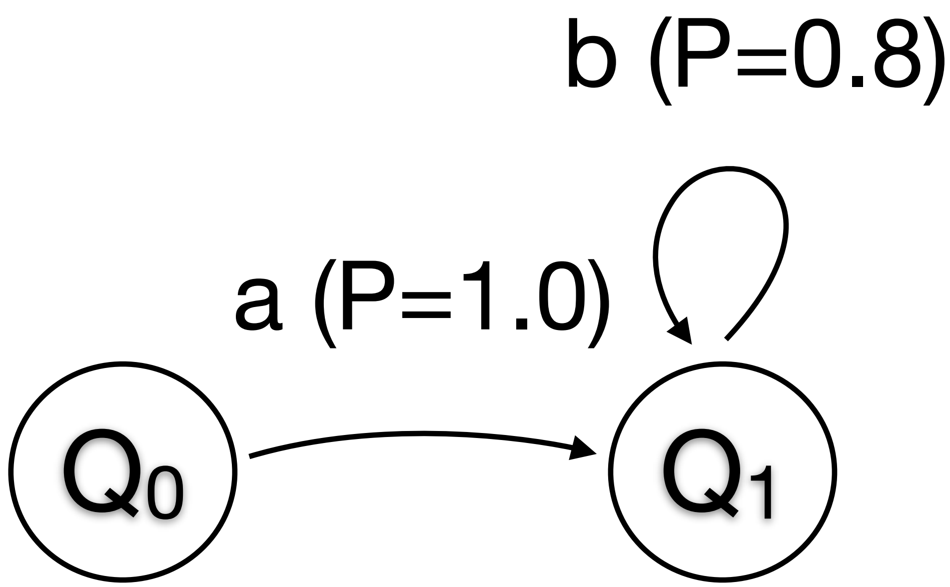
Probability of a string:

$$P(w_{1,n}, q_{1,n+1}) = I(q_1) \times F(q_{n+1}) \times \prod_{i=1}^{n} P(q_i, q_i, q_{i+1})$$

$$P(w_1, n) = \sum_{q_{1,n+1}} P(w_{1,n}, q_{1,n+1})$$

Let A be a PFA.

- Def: P(x | A) = the sum of all the valid paths for x in A.
- Def: a **valid** path in A is a path for some string x with probability greater than 0.
- Def: A is called **consistent** if $\sum_x P(x \mid A) = 1$
- Def: a state of a PFA is **useful** if it appears in at least one valid path.
- Proposition: a PFA is consistent if all its states are useful.



$I(Q_0) = 1.0$
$I(Q_1) = 0.0$

$F(Q_0) = 0.0$
$F(Q_1) = 0.2$

$P(ab^n)=0.2*0.8^n$

And if we add all possible paths:

$$\sum_x P(x) = \sum_{n=0}^{\infty} P(ab^n) = 0.2 * \sum_{n=0}^{\infty} 0.8^n = 0.2 * \frac{0.8^0}{1-0.8} = 1$$

Relation with other models

- A Markov chain is a special case of probabilistic FSA in which the input sequence uniquely determines which states the automaton will go through
  - only useful for assigning probabilities to unambiguous sequences
  - Ex: N-gram models
- Probabilistic FSA can be shown to be equivalent to Hidden Markov Models
  - Same expressivity, but different way of representing things

Cascaded finite-state transducers

- *"Partial Parsing via Finite-state cascades„* by Steven Abney 1996
- Different levels L_i
- For each level exists a deterministic FSA T_i
- Output of one level automaton is input to the next one:
- Level recognizer T_i has L_{i-1} as input, and ouputs symbols on level L_i
- Input elements that can't be recognized by an automaton are simply ignored and passed over to the next level
- Advantages: efficient, robust (partial parsing)
- Limitations: cannot model complex linguistic phenomena

Cascaded finite-state transducers

L_3							S			S
T_3										
L_2		NP				PP	VP	NP		VP
T_2										
L_1		NP	P			NP	VP	NP		VP
T_1										
L_0	D	N	P	D	N	N	V-tns	N	Aux	V-ing
	$_0$ *the* $_1$	*woman* $_2$	*in* $_3$	*the* $_4$	*lab* $_5$	*coat* $_6$	*thought* $_7$	*you* $_8$	*were* $_9$	*sleeping*

$$T_1: \left\{ \begin{array}{l} \text{NP} \rightarrow \text{D? N+} \\ \text{VP} \rightarrow \text{V-tns} \mid \text{Aux V-ing} \end{array} \right\}$$

$$T_2: \{\text{PP} \rightarrow \text{P NP}\}$$

$$T_3: \{\text{S} \rightarrow \text{PP}* \text{ NP PP}* \text{ VP PP}*\}$$