Finite State Automata (Exercise Session)

Stephan Busemann

Thanks to Anette Frank,
on whose materials part of this lecture is based

Outline of this exercise session

- **Correction of the exercises**
- **Advanced topics:**
  - Cascading finite-state transducers
  - Application to information extraction

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	-

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
    else:  if 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},δ_1,p,{s}> where δ_1 is as follows:

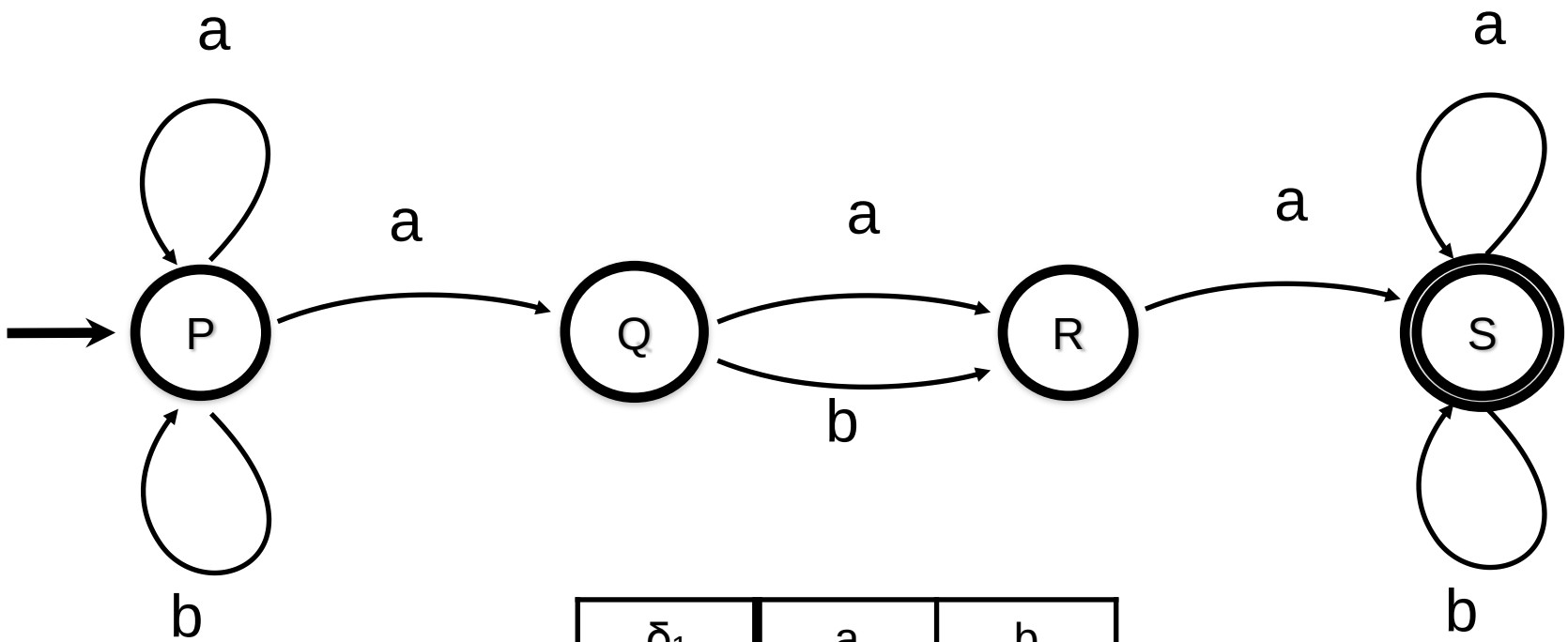
δ_1	a	b
p	p,q	p
q	r	r
r	s	-
s	s	s

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).

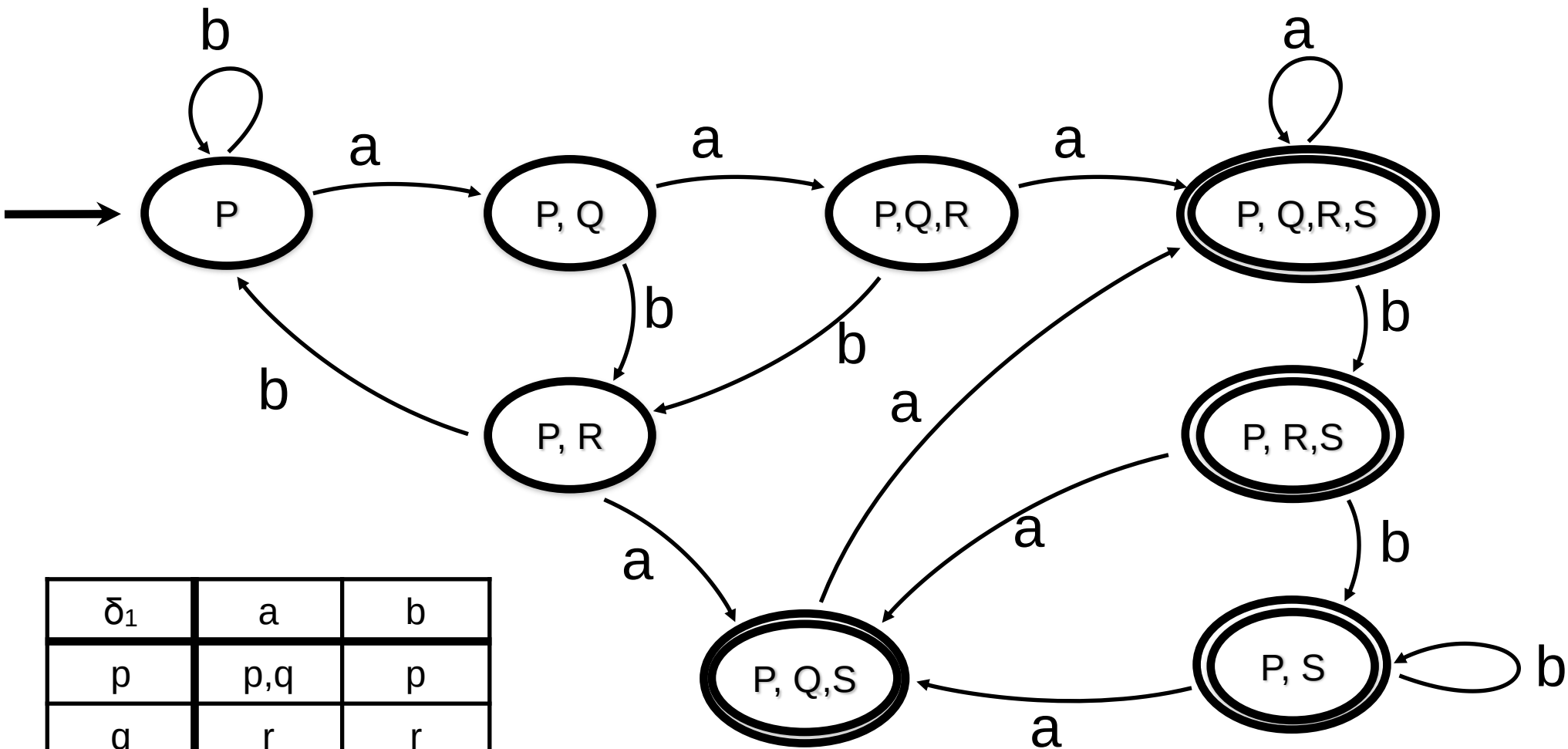
δ_3	0	1
A	B	D
B	B	C
C	D	E
D	D	E
E	C	-

Determinization



δ_1	a	b
p	p,q	p
q	r	r
r	s	-
s	s	s

Determinization



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.
   $A_1=<\{p,q,r,s\},\{a,b\},\delta_1,p,\{s\}>$ 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	-

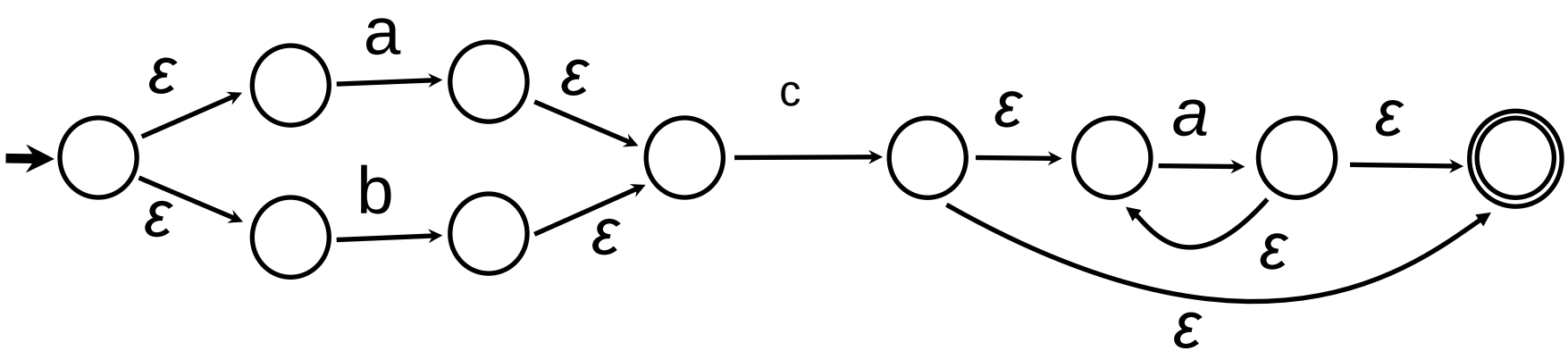
Constructing a FSA from a regexp

The regular expression:

(a|b)ca*

union of two FSA

Kleene Star over FSA

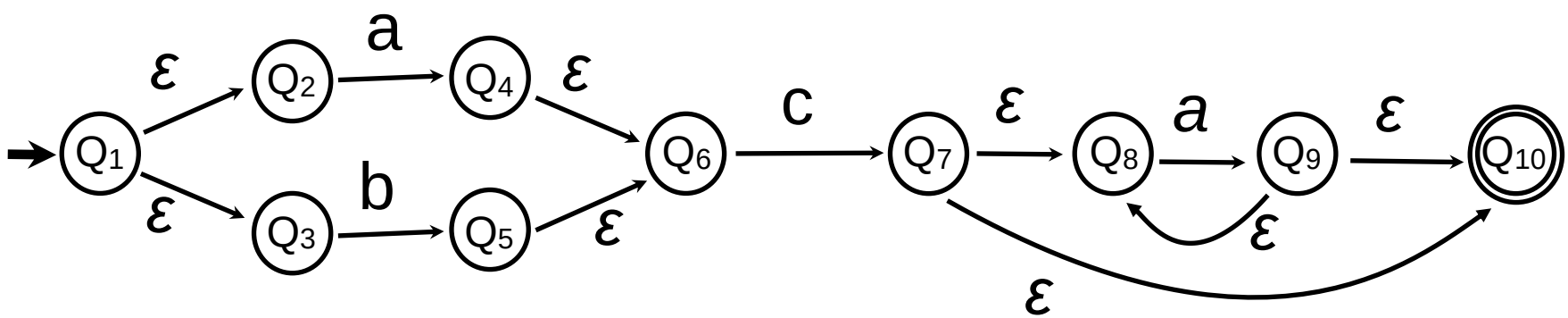


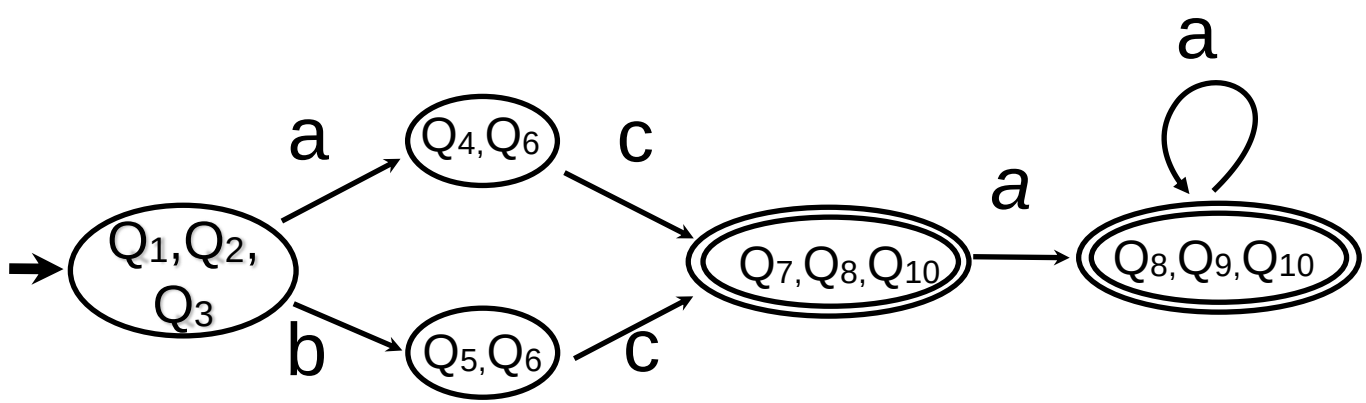
But this is a non-deterministic FSA...

Constructing a FSA from a regexp

The regular expression:

(a|b)ca*

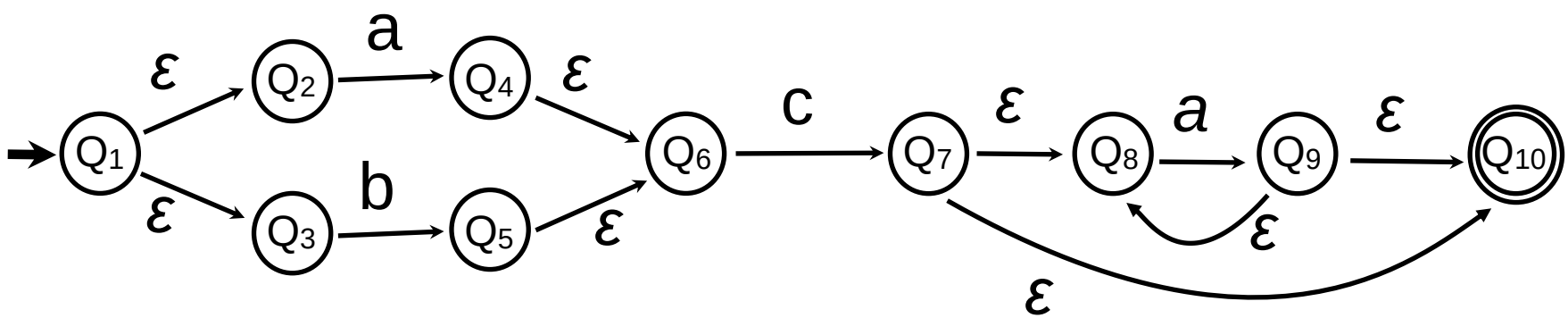




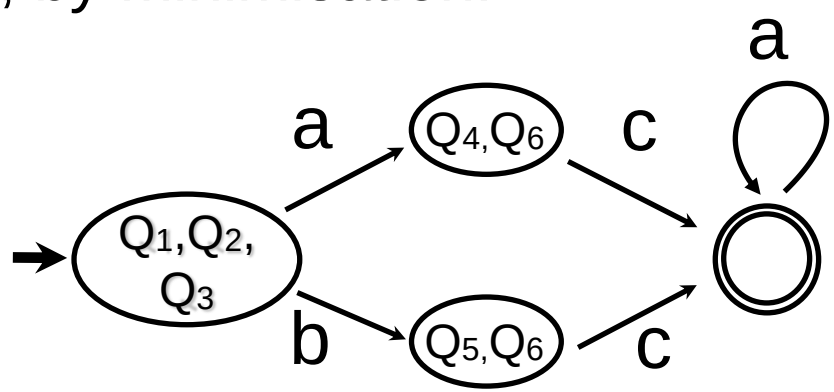
Constructing a FSA from a regexp

The regular expression:

(a|b)ca*



Or even simpler, by minimisation:



Exercises

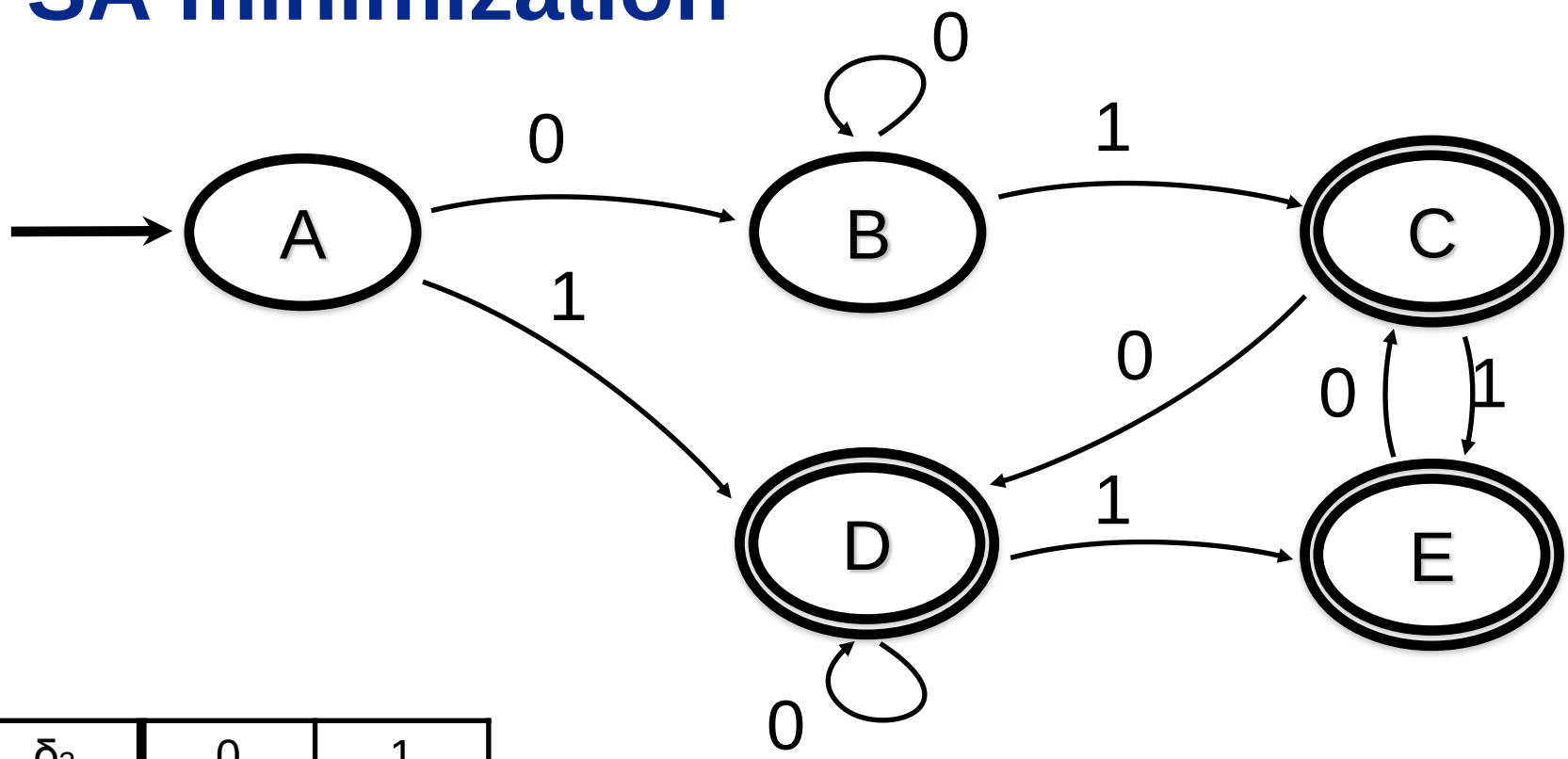
1. Write a program for acceptance of a string by a DFSA.
   Then extend it to a finite-state transducer that can translate a surface form to lemma + POS, or between upper and lower case.
2. Determinize the following NFSA by subset construction.
   $A_1=<\{p,q,r,s\},\{a,b\},\delta_1,p,\{s\}>$ where δ_1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

FSA minimization

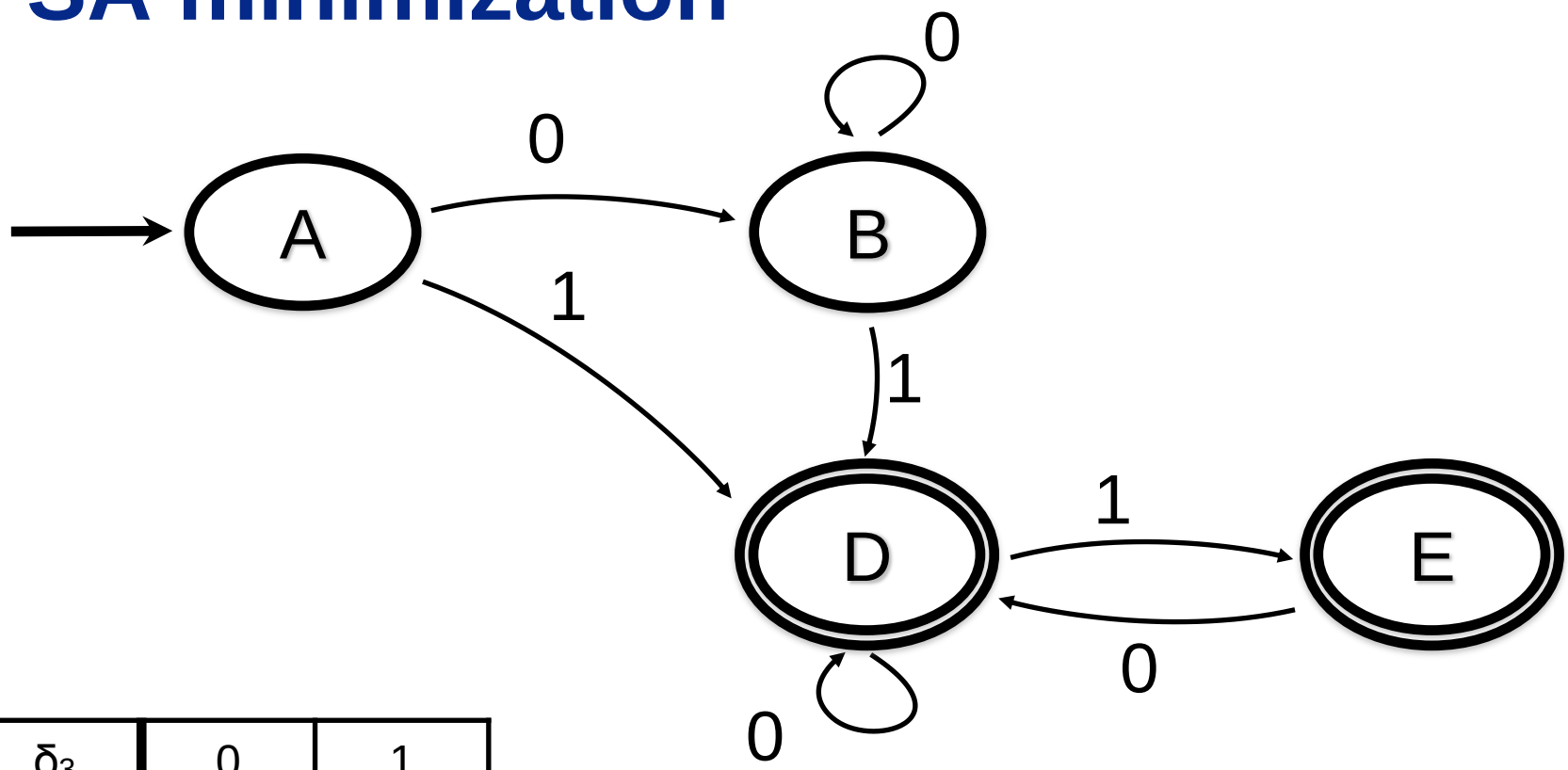


δ_3	0	1
A	B	D
B	B	C
C	D	E
D	D	E
E	C	

C and D have the same right language:

0*|(0*(10)*)*|(0*(10)*1)*

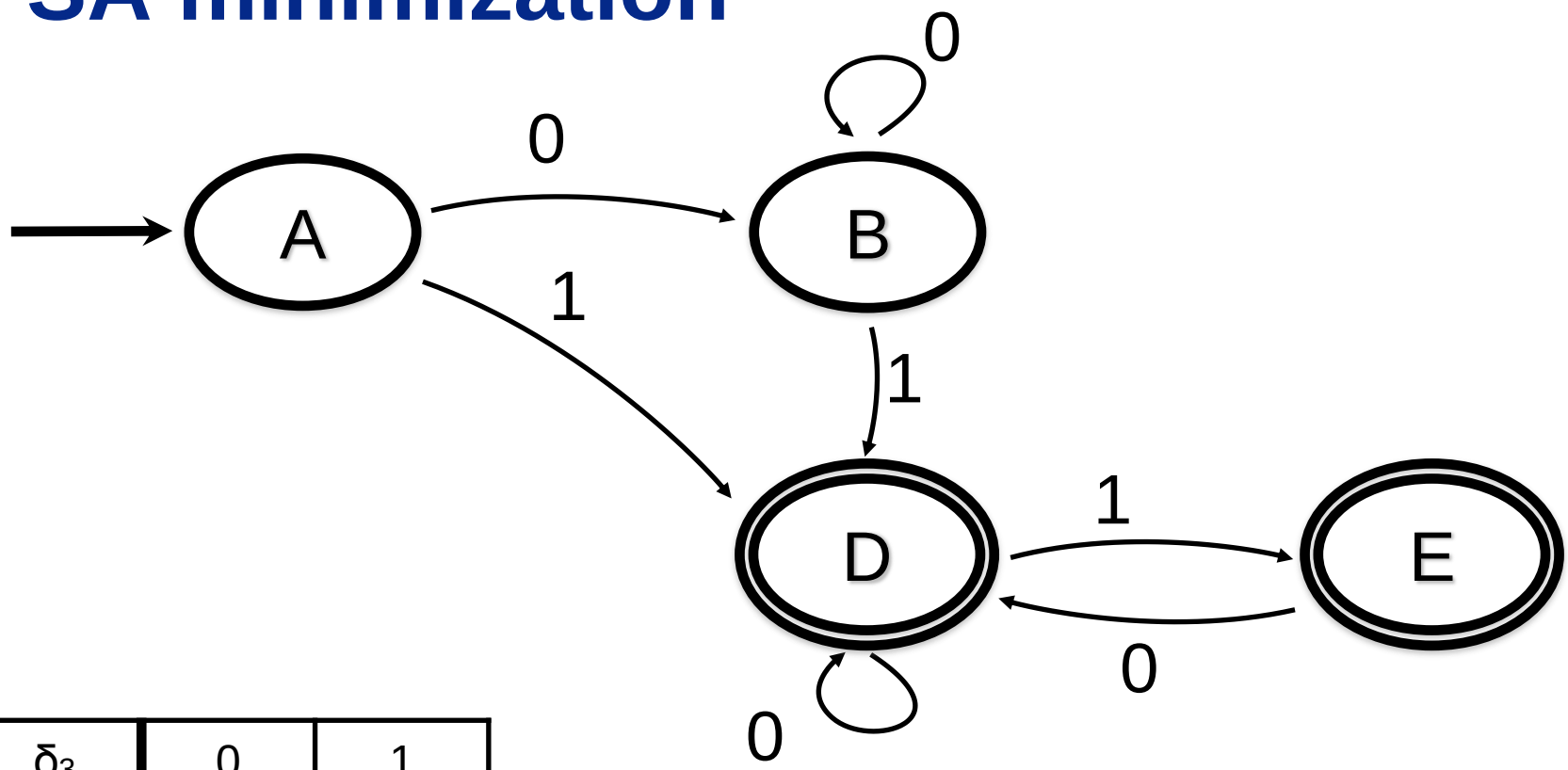
FSA minimization



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

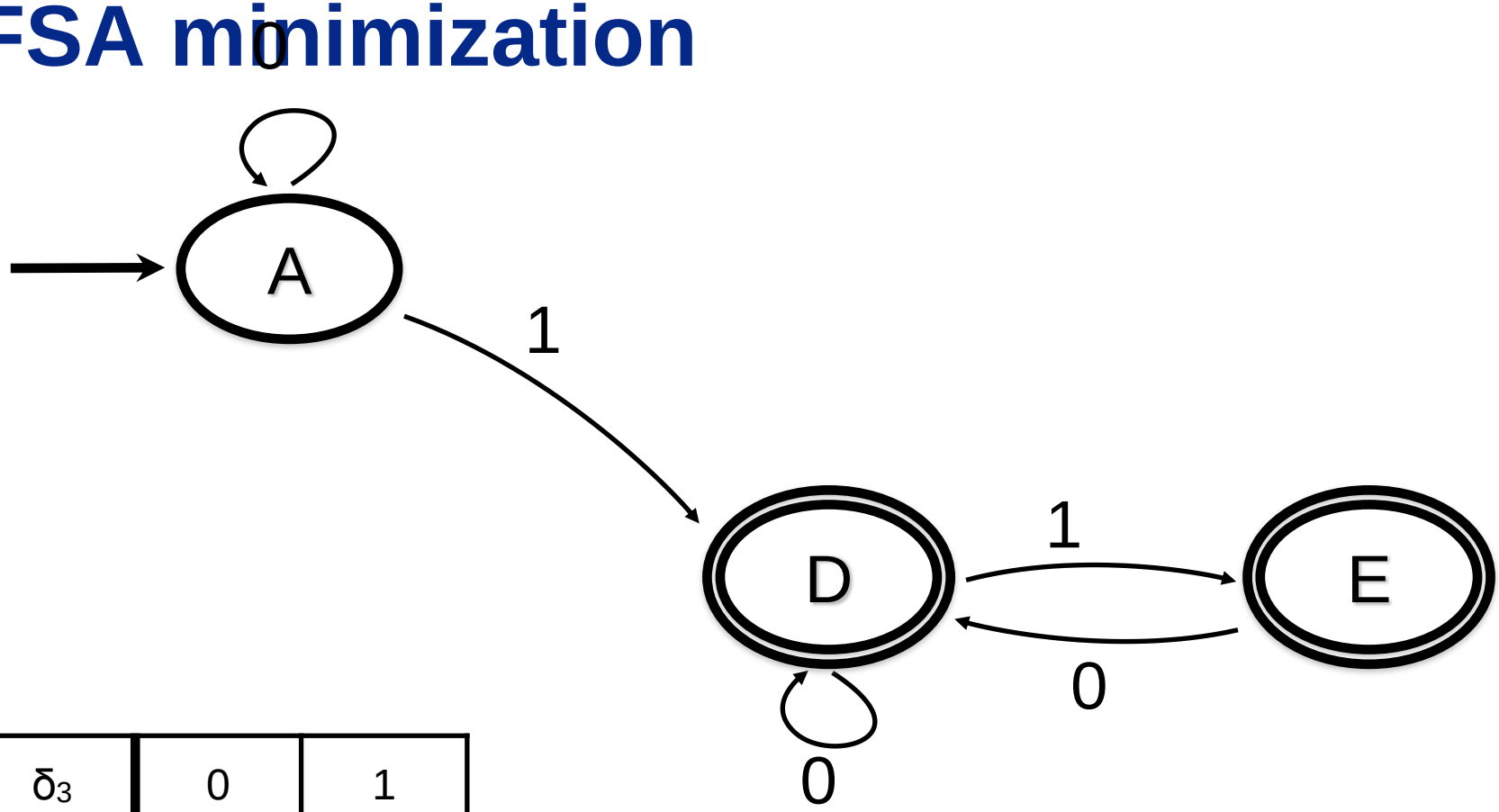


δ_3	0	1
A	B	D
B	B	D
D	D	E
E	D	

A and B also have the same right language:

0*1+right language of D

FSA minimization



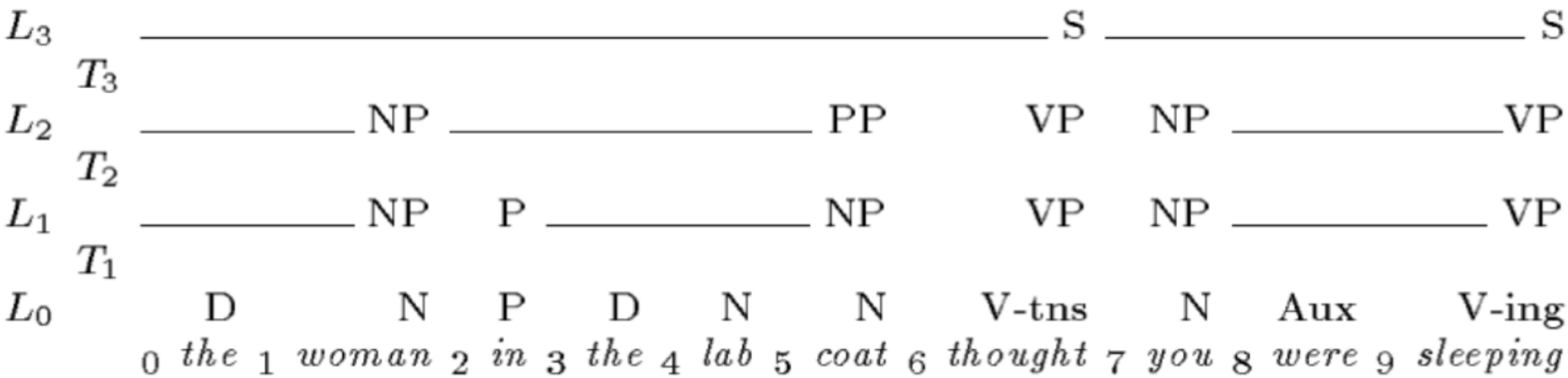
δ_3	0	1
A	A	D
D	D	E
E	D	

And we‘re done!

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



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

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

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

Application to Information Extraction

- **FASTUS, IE system by Doug Appelt et al., 1991-1994**
  - Breakthrough in efficient processing
  - Cascaded, nondeterminstic FSA, five stages
- **Stage 1**
  - Recognizing multiwords, names and other fixed form expressions
- **Stage 2**
  - Basic noun groups, verb groups, prepositions, some particles
- **Stage 3**
  - Complex noun groups and verb groups
- **Stage 4**
  - Patterns of events of interest, buiding „event structures“
- **Stage 5**
  - Recognizing and merging events structures describing the same event, generating database entries

FASTUS Stage 2: Basic groups

Company Name	*Bridgestone Sports Co*.
Verb Group	*said*
Noun Group	*Friday*
Noun Group	*it*
Verb Group	*had set up*
Noun Group	*a joint venture*
Preposition	*in*
Location	*Taiwan*
Preposition	*with*
Noun Group	*a local concern*
Particle	*and*
Noun Group	*a Japanese trading house*
Verb Group	*to produce*
Noun Group	*golf clubs*
Verb Group	*to be shipped*
Preposition	*to*
Location	*Japan*

"Company Name" and "Location" are special kinds of noun groups.

FASTUS Stage 3: complex groups

- **Attachment of appositions to their head noun group**
  - *the joint venture, Bridgestone Sports Taiwan Co.,*
- **Construction of measure phrases**
  - *20,000 iron and "metal wood" clubs a month*
- **Attachment of “of” and “for” PP to their head noun groups**
  - *production of 20,000 iron and "metal wood" clubs a month*
- **Noun group conjunction**
  - *a local concern and a Japanese trading house*

FASTUS Stages 4 and 5: event structures

- **Stage 4**
  - Everything that is not covered by a complex phrase in stage 3 is ignored
  - Significant source of system robustness
  - Partially filled templates

- **Stage 5**
  - Operates over whole text rater than single sentences
  - All information collected about a single entity or relationship is integrated into a unified whole
  - First approaches towards coreference resolution involved in merging
    - **Type compatibility**
    - **Nearness**
    - **Consistency of content of single event structures**

Conclusions

- **Because of their mathematical and computational simplicity, regular languages and finite-state machines have been applied in many information processing tasks**
- **Non NLP**
  - Global search pattern specification, e.g. Word processors, Unix grep
  - Lexical analysis of most modern programming language compilers, e.g. identifier classes, punctuation, numbers etc
- **NLP**
  - An approximation to proper grammatical language description is good enough
  - Information extraction
  - Storing and accessing NL dictionaries by representing words in a state transition graph
  - Transducers in phonology: in+practical – impractical, stop+ing – stopping
  - Two level morphology