2 - Finite State Methods

1. From Regular Expressions to Finite State Automata
2. Finite State Automata
3. Deterministic Finite State Automata
4. Non Deterministic Finite State Automata
5. Determinization of Finite State Automata
6. Minimization of Finite State Automata
7. Finite State Automata and Regular Languages
8. Finite State Transducers
9. Determinization of Finite State Transducers
10. Application of finite state methods to NLP

Bibliography

- D. Jurasfsky & J. H. Martin - *Speech and Language Processing* - Prentice Hall, 2008.
- Emmanuel Roche & Yves Schabes - *Finite-State Language Processing* - MIT Press, 1997.
- John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman - *Introduction to Automata Theory, Languages, and Computation* - Addison Wesley, 2006.

2.1 – From Regular Expressions to Finite State Automata

- Some sets of strings (named entities, numerals, dates …), which can be infinite, need to be characterized in a compact finite way.

- **Regular expressions** are used to characterize these sets of strings, which constitute **regular languages**.

- **Finite State Automata (FSA**) are abstract computing machines used to recognize **regular languages**.

2.2 - Finite State Automata

- Finite State Automata (FSA) are abstract computing machines, the goal of which is to recognize particular languages: **regular languages**.
- FSA can be normalized to get time and space efficiency (determinization and minimization).
- FSA combine in various manners to produce new FSA.

2.2 - Finite State Automata

A **Finite State Automaton** (FSA) is composed of two parts:

- An infinite **tape** where positions are numbered 0, 1, 2, … until the infinite; a pointer marks the current position which can be read only; the pointer can only move forward.

- A **control unit** which controls the forward movements of the pointer as the actions of reading the tape.

2.3 - Deterministic Finite State Automata

Definition : a Deterministic Finite State Automaton is a 5-tuple $(Q, \Sigma, q_0, F, \tau)$ such that :

- Q is a finite set of states of the control unit.
- Σ is a finite input tape alphabet of symbols.
- q_0 is a particular element of Q, the start state of the automaton.
- F is a subset of Q, the accepting states of the automaton.
- τ is a transition function which associates a source state q from Q and an input symbol a from Σ to a target state from Q denoted $\tau(q,a)$.

2.3 - Deterministic Finite State Automata

- **Definition :** a **computation** on an automaton is a (possibly empty) sequence of transitions in the form : $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} q_n$ where q_0 is the initial state.
  We denote : $q_0 \xrightarrow{a_1 a_2 \cdots a_n} * q_n$ and we say that q_n recognizes the word $a_1 a_2 \dots a_n$

- A word is **recognized** by the automaton if it recognized by an accepting state.

- The language recognized by the automaton is the set of all words recognized by the automaton.

- The time complexity of the recognition problem with a deterministic automaton is linear with respect to the input size.

2.3 - Deterministic Finite State Automata

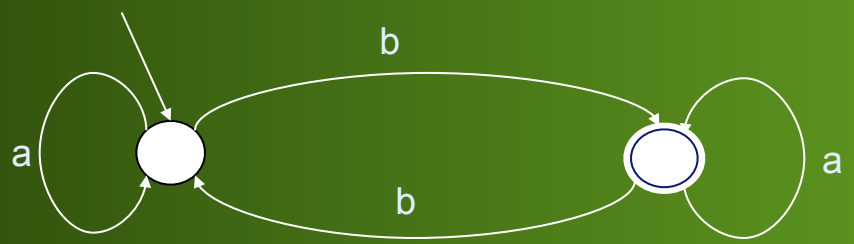
- **Algorithm of recognition**

```
function D-RECOGNIZE (tape, automaton)
index ← beginning_of (tape)
current-state ← initial_state_of (automaton)
loop
    if index = end_of_input (tape) then
            if current-state ∈ accept_states (automaton) then
                    return accept
            else
                    return reject
            endif
    elseif transition_table (automaton)[current-state, tape[index]]) = empty then
            return reject
    else
            current-state ← transition_table (automaton)[current-state, tape[index]]
            index ← index + 1
    endif
endloop
```

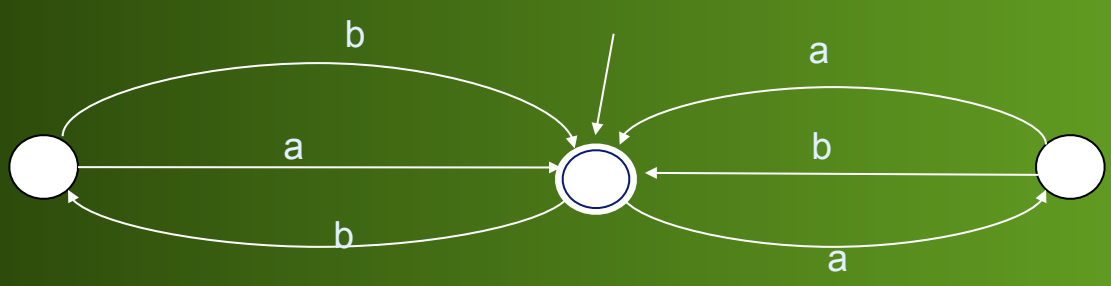
2.3 - Deterministic Finite State Automata

1. Describe the languages recognized by the following FSA:

a)



b)



2.3 - Deterministic Finite State Automata

2. Determine an automaton recognizing the following languages :

a) The set of natural numbers in base 10.

b) $\{0^n 1^m \mid n, m \in \mathbb{N}\}$.

c) The set of strings including 11 (the alphabet is {0, 1}).

d) The set of dates in the format *day/month,* where *day* and *month* are natural numbers with two digits; *month* is between 01 and 12 and *day* must be compatible with *month* (for instance, if *month* = 02 then $01 \leq day \leq 29$).

e) The set of floating numbers. A floating number is written as a mantissa followed by an exponent. The mantissa is composed of a facultative sign followed by a whole part and possibly a decimal which starts with a point. The exponent is composed of the letter E followed by a facultative sign and a natural number.

f) The language of comments in computer programs. A comment is a sequence of characters between /* and */. Such sequence cannot include /* and */ except if these are immediately preceded by the escape character %.

2.3 - Deterministic Finite State Automata

3. We consider a reduced lexicon for a fragment of French which associates a POS set to every word. The POS used in this lexicon are : Det (determiner), CN(Common Noun), PN (Proper Noun), V (intransitive verb), Vcompl (verb with a sentential clause complementized by “que” as direct object), Conj (subordinating conjunction “que”), Adv (Adverb modifying any verb).

   We consider French sentences and we assume that all sentences are tagged with the previous POS from the lexicon and we want to build an automaton to recognize the grammatical sentences.

   The words (in the sense of automata) that have to be recognized by the automaton are sequences of POS. For instance, for the sentence “$Jean_{PN}\ pense_{Vcompl}\ que_{Conj}\ la_{Det}\ femme_{CN}\ arrive_{V}$”, the input word for the automaton is “*PN Vcompl Conj Det CN V*”.

   Build a DFSA for recognizing the largest fragment of French that can be defined from the given set of POS.

2.4 - Non Deterministic Finite State Automata

Definition : a Non Deterministic Finite State Automaton over an alphabet Σ, is a 5-tuple $(Q, \Sigma, q_0, F, \tau)$ such that :

- Q is a finite set of states of the control unit.
- $\Sigma \cup \{\varepsilon\}$ is a finite input alphabet of symbols. The special symbol ε is the empty string, that is the unit for concatenation of strings from $\Sigma \cup \{\varepsilon\}$.
- q_0 is a particular element of Q, the start state of the automaton.
- F is a subset of Q, the accepting states of the automaton.
- τ is a transition relation which associates a source state q_1 and an input symbol a with a target state q_2.This is denoted: $\tau(q_1, a, q_2)$.

2.4 - Non Deterministic Finite State Automata

An algorithm of recognition :

```
function ND-RECOGNIZE (tape, automaton)
agenda ← { }
index ← beginning_of (tape)
current-state ← initial_state_of (automaton)
loop
        if index = end_of_input (tape) and current-state ∈ accept_states (automaton) then
                return accept
        else
                forall state ∈ transition_table (automaton)[current-state, tape[index]]
                        agenda ← agenda ∪ {(state, index + 1)}
                endforall
                forall state ∈ transition_table (automaton)[current-state, ε]
                        agenda ← agenda ∪ {(state, index )}
                endforall
                if there is no new element in agenda then
                        return reject
                else
                        (current-state, index) ← choose_a_new_element(agenda)
                endif
        endif
endloop
```

2.5 - Determinization of Finite State Automata

- The complexity of the recognition problem with a given DFSA is **linear-time** in the length of the input word and it does not depend on the size of the automaton.

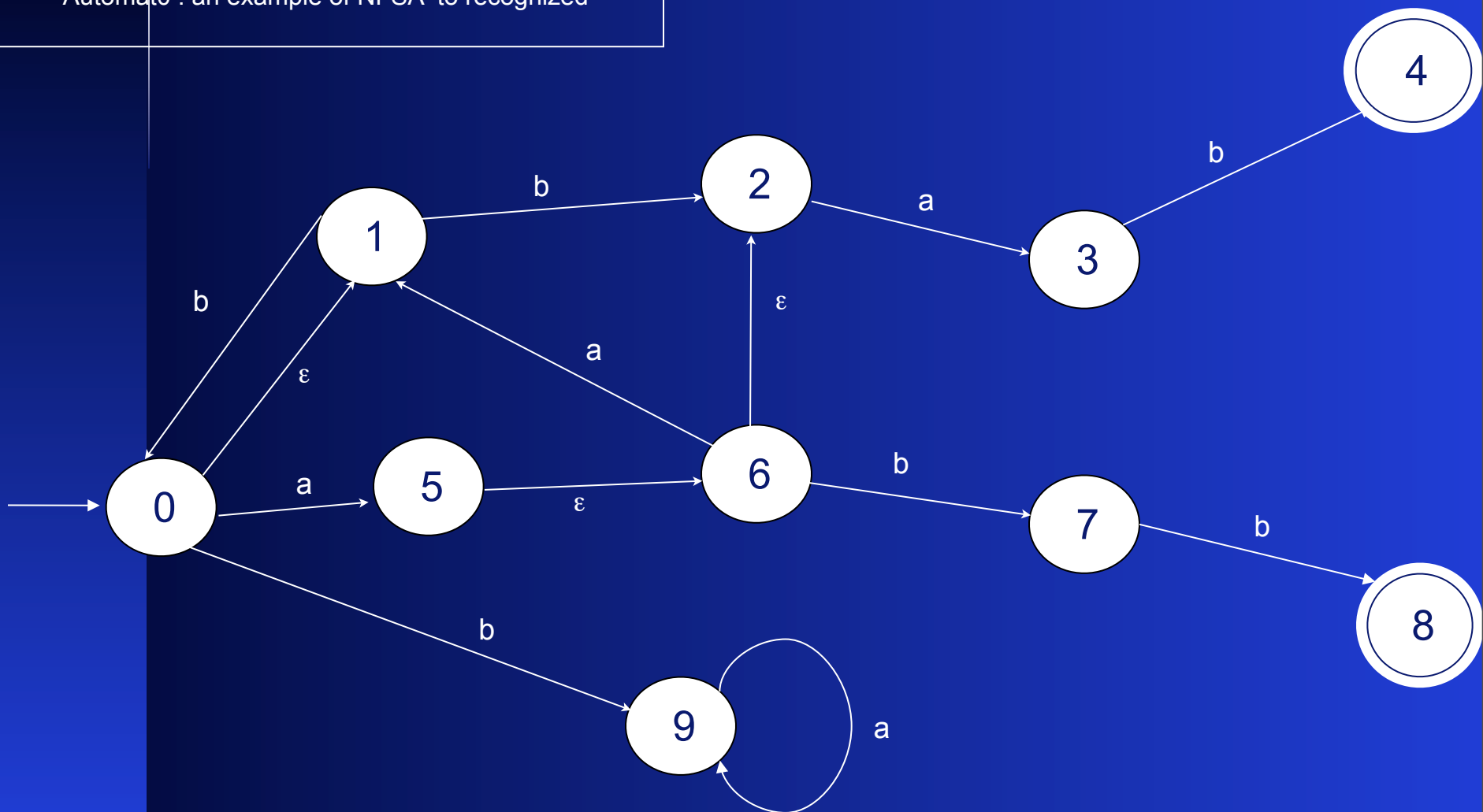
- **Theorem** : for any NFSA, there exists an equivalent DFSA, that is an automaton recognizing the same language

2.5 - Determinization of Finite State Automata

- The principle of transforming a NFSA into a DFSA is the following:
  - *any state of the DFSA is the set of all NFSA states recognizing the same word of* Σ^**;*
  - *the initial state of the DFSA is the set of all NFSA states recognizing the empty word;*
  - *the final states of the DFSA are all states containing a NFSA final state;*
  - *if two DFSA states* s_1 *and* s_2 *are respectively associated with* w *and* w.a *words from* Σ^**, there is a transition from* s_1 *to* s_2*.*
- The DFSA is built state by state recursively from its initial state.

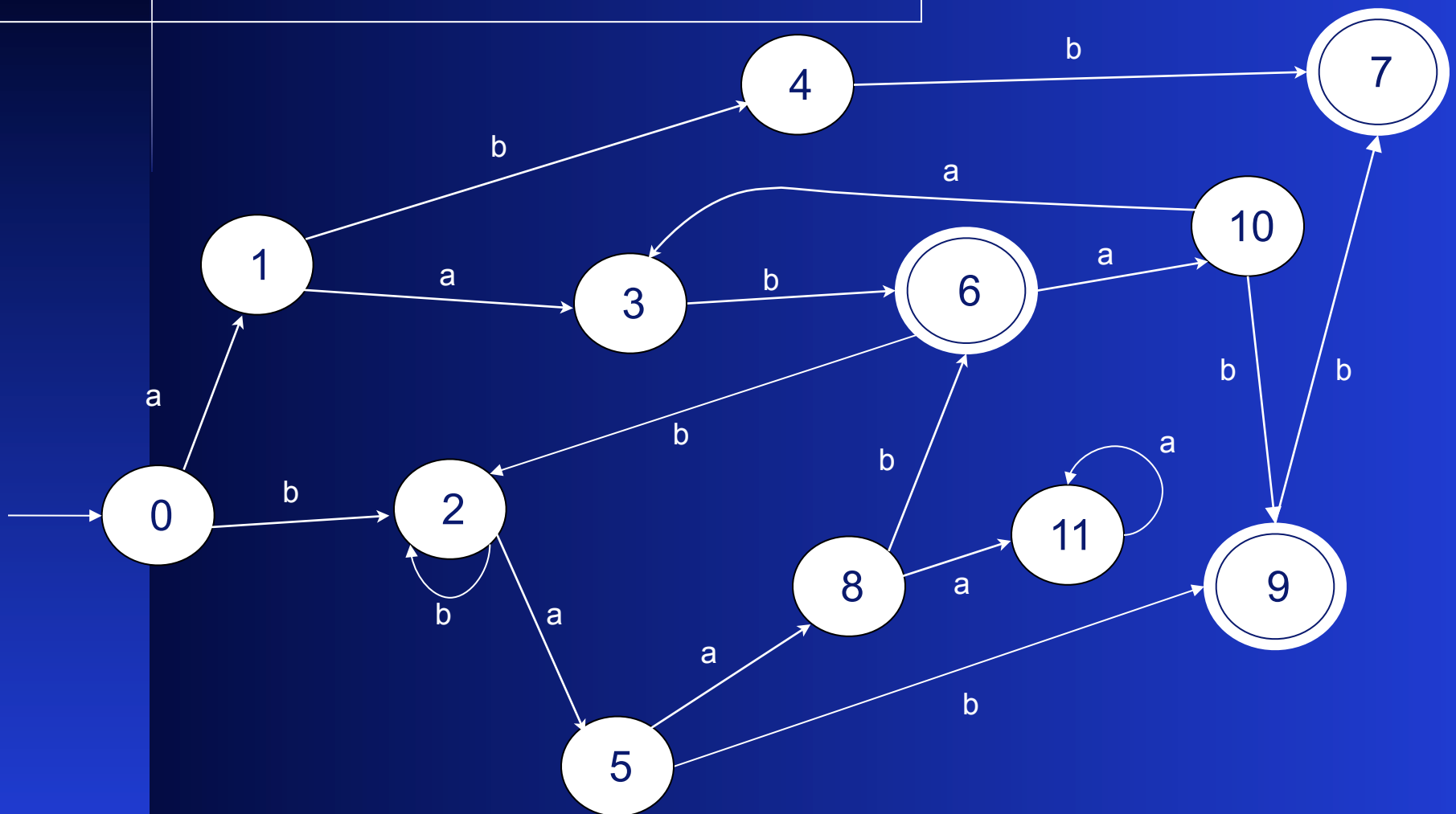
2.5 - Determinization of Finite State Automata

Automat0 : an example of NFSA to recognized



2.5 - Determinization of Finite State Automata

Automat1: a DFSA which results from determinizing *Automat0*.



2.6 - Minimization of Finite State Automata

- If we change the initial state of a FSA with any state *q*, the language recognized by the new FSA is **the language recognized from state** *q* in the former FSA.

- Every DFSA can be **minimized** by identifying its **interchangeable** states, the states from which the same languages are recognized.

2.6 - Minimization of Finite State Automata

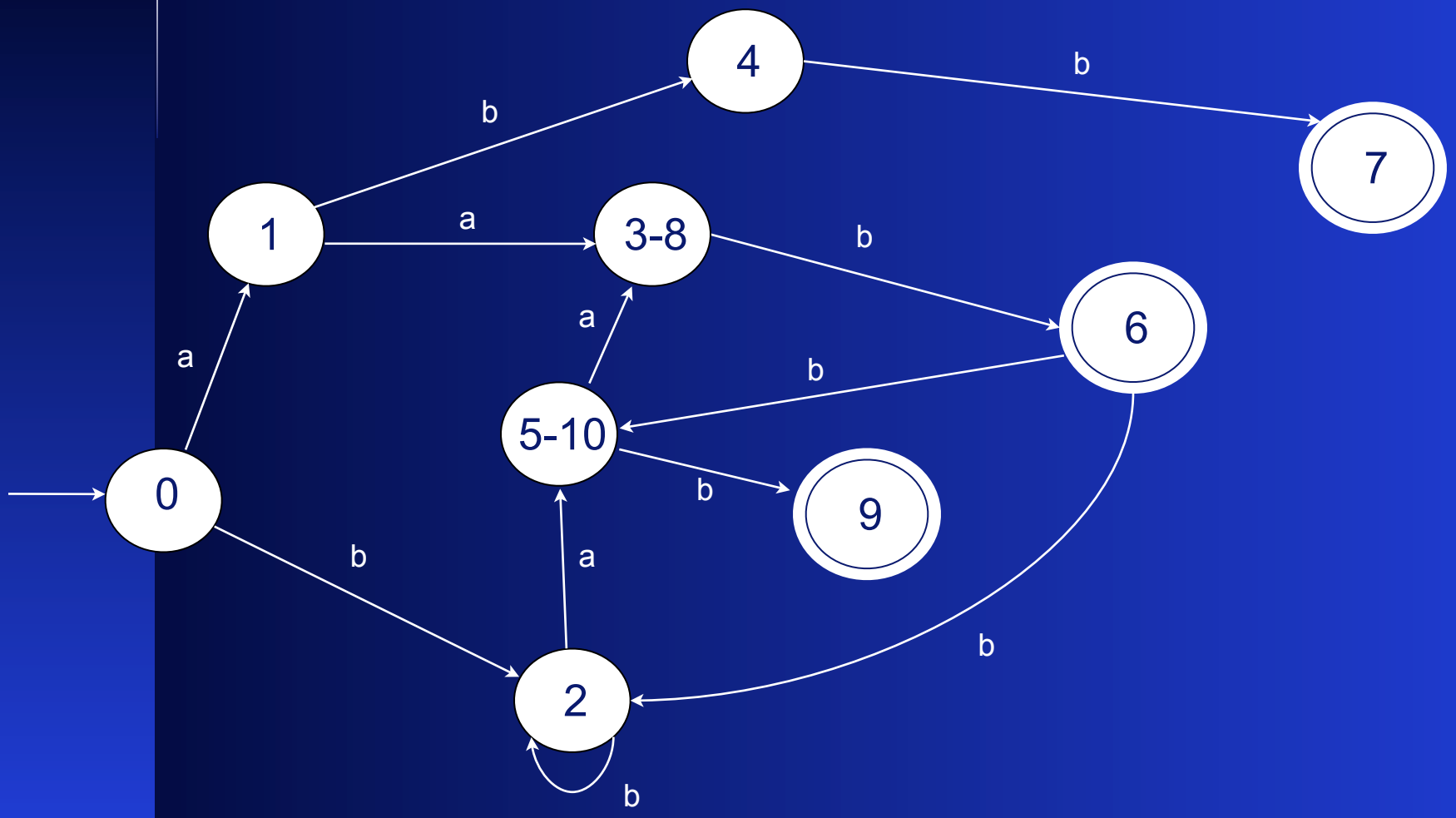
- For every DFSA A_1, the minimal automaton A_2 equivalent to A_1 is defined as follows:
  - *The states of A_2 are the equivalence classes of A_1 interchangeable states;*
  - *The initial state of A_2 is the equivalence class of the A_1 initial state;*
  - *The final states of A_2 are the equivalence classes of the A_1 final states;*
  - *For any transition in A_1 on a symbol a from a state s_1 to a state s_2, there is a corresponding transition on a in A_2 from the equivalence class of s_1 to the equivalence class of s_2*

2.6 - Minimization of Finite State Automata

- The minimal automaton is built recursively by approximating the interchangeability classes step by step, according to Moore's algorithm (1956):
  - The partition between equivalence classes is initialized with two classes : the class of final states and the class of non final states plus the garbage state.
  - Then, the partition is refined step by step. At each step, each class is partitioned according to the input transitions from other classes.
  - The process stops when a fix point is reached.
- The complexity of the algorithm is quadratic in time in the size of the automaton.
- There is an exponential algorithm which consists in determinizing the reverse automaton.

2.6 - Minimization of Finite State Automata

Automat2: resulting from the minimization of *Automat1*.



2.6 - Minimization of Finite State Automata

1. Consider the following transition table defining a DFSA with the initial state q_0 and the unique accepting state q_2. Minimize this automaton.

	q_0	q_1	q_2	q_3	q_4	q_5	q_6	q_7
0	q_1	q_6	q_0	q_2	q_7	q_2	q_6	q_6
1	q_5	q_2	q_2	q_6	q_5	q_6	q_4	q_2

2. Consider the following transition table defining a NFSA with the initial state q_0 and the unique accepting state q_9. Determinize and minimize this automaton.

	q_0	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9
ε	q_2		q_0				q_9			q_6
0	q_1	q_3	q_4			q_6	q_7	q_9		
1		q_2	q_5	q_2	q_6		q_8		q_9	

2.7 - Finite State Automata and Regular Languages

Definition : The family of **regular languages** over an alphabet Σ is the least family of languages over Σ with the following properties :

- The empty set ø is a regular language.
- The singleton $\{\varepsilon\}$ is a regular language.
- For every element a from Σ, the singleton {a} is a regular language.
- If L is a regular language , its **Kleene closure** L* is a regular language.
- If L_1 and L_2 are regular languages, their **concatenation** $L_1 L_2$ and their **disjunction** $L_1 \mid L_2$ are regular languages.

A **regular expression** is the expression of a regular language from elementary languages with the operations of Kleene closure, concatenation and disjunction.

2.7 - Finite State Automata and Regular Languages

- **Theorem** : If A_1 and A_2 are two NFSA, there exists a NFSA that recognizes the **intersection** , the **union** and the **concatenation** of their languages $L(A_1)$ and $L(A_2)$.

- **Theorem** : If A is a NFSA, there exists a NFSA that recognizes the **Kleene closure,** the **complement** and the **mirror image** of its language $L(A)$

2.7 - Finite State Automata and Regular Languages

- **Theorem (Kleene)**: the class of regular languages identifies with the class of the languages recognized by FSA.

- **Proof** :
  - First direction : any regular language is recognized by a FSA.

    Proof by induction on the structure of the regular expression defining the language; the induction step uses the previous theorems for the operations of union, concatenation and Kleene closure (Thompson method).

  - Second direction : any language recognized by a FSA is regular.

    Proof by addition of a new initial state and a new final state and by removing all intermediate states step by step (Brzozowski and Mc Cluskey algorithm).

2.7 - Finite State Automata and Regular Languages

- **Corollary :** the class of regular languages over an alphabet *A* is closed under **intersection**, **complementation** and **reverse operation**.

- **Pumping Lemma** : let L be an infinite regular language. Then, there are words x, y and z, such that $y \neq \varepsilon$ and $xy^nz \in L$ for any $n \geq 0$.

- The Pumping Lemma is used to prove that a language is not regular.

2.7 - Finite State Automata and Regular Languages

1. Build the minimal automaton recognizing the language defined by the regular expression (a(ca* | b*c)*bcd*)*

2. Build a regular expression defining the language recognized by the automaton with the following transition table (the initial state is q_0 and the final states are q_2 and q_4):

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

2.7 - Finite State Automata and Regular Languages

3. Define the operations of intersection and complementation of FSA.
   Hints : for the intersection, consider new states which are pairs of states from each initial automaton; for the complementation, introduce a garbage state to complete the automaton.

4. Determine if the following languages are regular :

   a. The set of the words defined in the 2001 edition of the dictionary Larousse over the usual French alphabet plus hyphen.

   b. $\{(ab)^n(ba)^n \mid n \in \mathbb{N}\}$

   c. $\{(ab)^n a(ba)^n \mid n \in \mathbb{N}\}$

   d. The arithmetic expressions on natural numbers in infixed notation (the usual notation).

   e. The arithmetic expressions on natural numbers in polish notation (for instance, *(2-3)x4+10x5* is written *+ x - 2 3 4 x 10 5*).

2.7 - Finite State Automata and Regular Languages

5. Three missionaries and three cannibals want to cross a river. There is only one boat which can carry two persons at most. A missionary cannot use the boat alone. A soon as there are more cannibals than missionaries at some place, the cannibals will eat the missionaries. Is there a way for the six persons to cross the river without being eaten.

   Hints : use an automaton where states represent the possible distributions of missionaries and cannibals between the two sides of the river and transitions represent the crossings of the river.

2.7 - Finite State Automata and Regular Languages

6. The blind barman with boxing gloves

A barman and a client play at the following game :
The barman blindfolds himself and gets boxing gloves. Then, he cannot see and determine if a glass is up or down. In front of the barman, there is a salver with four glasses put in square. These glasses can be up or down. Their direction is chosen initially by the client and unknown by the barman.
The barman can repeat the following operation ten times at most : he indicates the glasses that he wants to turn over; the client turns the salver and then, the barman turns the glasses over as he has indicated. If the glasses are all in the same direction, the barman wins.

2.7 - Finite State Automata and Regular Languages

a) Give an automaton the states of which are the different configurations of the salver and the transitions are the possible changes of configuration, their input symbols representing the indications of the barman.

b) From this automaton, deduce another automaton in which the accepting states are the configurations that are winning for the client.

c) Give an automaton that guarantees the victory to the barman whichever the choice of the client is.

d) Solve the problem again with three glasses in triangle and then with five glasses in pentagon.

2.8 - Finite State Transducers

- The aim of FSA is **language recognition**.
- Sometimes, it is necessary not only to recognize words of a certain language but also to map them to new words (for instance, the command query/replace in text processing). For this, we use **Finite State Transducers** (FST).
- A FST defines a relation between the words of two languages : an input language and an output language. Its aim is **language translation**.

2.8 - Finite State Transducers

Definition : a FST over an input alphabet Σ_i and an output alphabet Σ_o is a 6-uple $(Q, \Sigma_i, \Sigma_o, q_0, F, \tau)$ such that :

- Q is a finite set of states.
- Σ_i is a finite input alphabet of symbols augmented with the empty string ε.
- Σ_o is a finite output alphabet of symbols augmented with the empty string ε.
- q_0 is a particular element of Q, the start state of the FST.
- F is a subset of Q, the accepting states of the FST.
- τ is a transition relation which associates a source state q_1 from Q and an input symbol a_i from $\Sigma_i \cup \{\varepsilon\}$ with a target state q_2 from Q and an output symbol b_o from $\Sigma_o \cup \{\varepsilon\}$. This is denoted: $\tau(q_1, a_i, q_2, b_o)$.

33

2.8 - Finite State Transducers

- **Definition :** a **computation** on a transducer is a (possibly empty) sequence of transitions in the form : $q_0 \xrightarrow{v_1 : w_1} q_1 \xrightarrow{v_2 : w_2} \ldots \xrightarrow{v_n : w_n} q_n$ where q_0 is the initial state.

  We summarize the computation as follows: $q_0 \xrightarrow{v_1 v_2 \cdots v_n : w_1 w_2 \cdots w_n} * q_n$

- The computation is successful if q_n is an accepting state. In this case, the word $v_1 v_2 \ldots v_n$ is **transduced** into the word $w_1 w_2 \ldots w_n$.

- There is an equivalent definition of transducer where transitions are labelled with pairs of words instead of pairs of symbols.

2.8 - Finite State Transducers

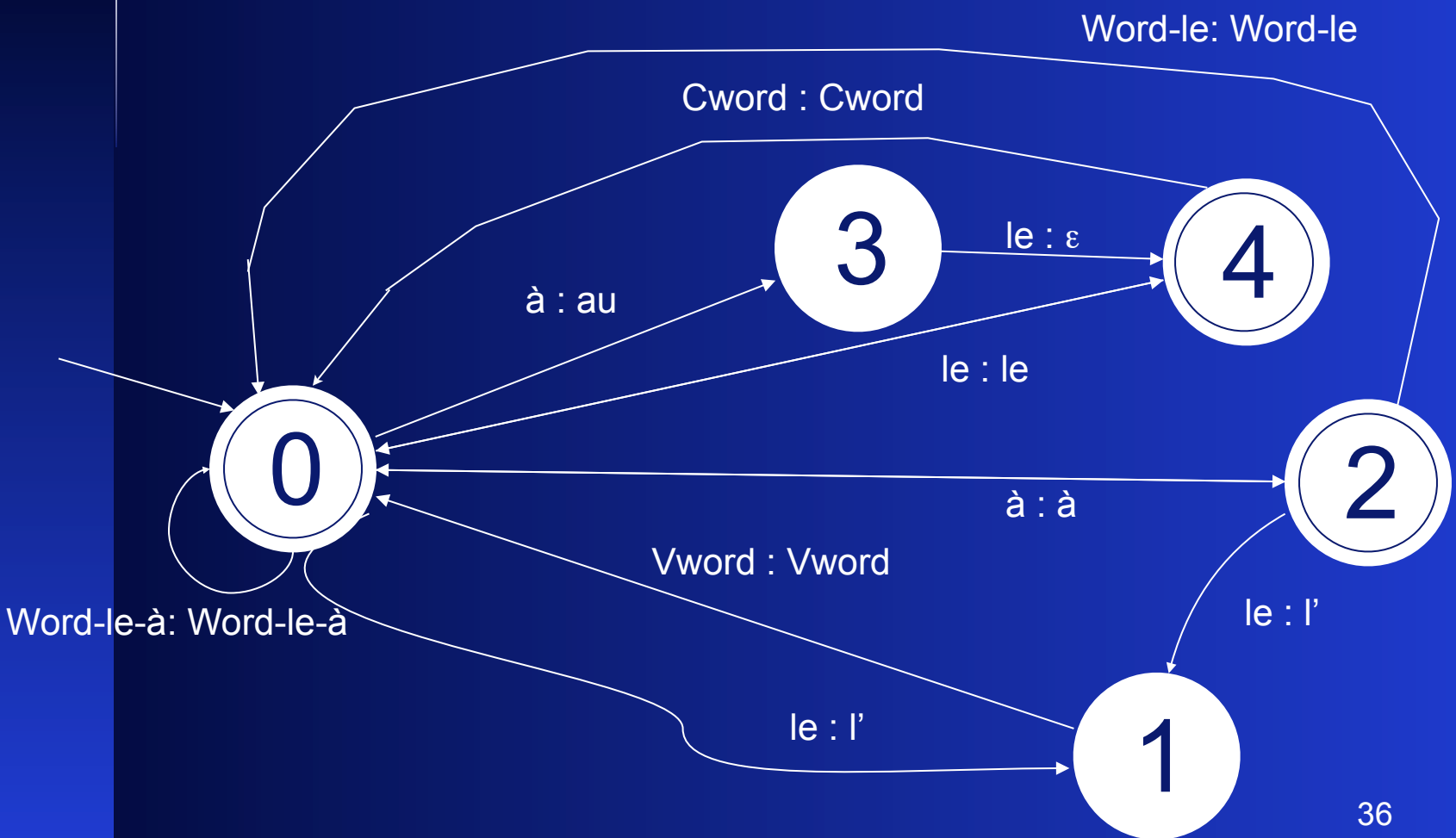
- The relation realized by the transducer is the set of all pairs of words (w_i, w_o) such that w_i is transduced into w_o. This relation is called a **regular (rational) relation**.

- We denote |T| the function that maps every input word w_i to the set |T|(w_i) of all output words that result from the transduction of w_i by T.

- If any input word w_i is transduced into one output word w_o at most, the transducer realizes a **rational function**.

2.8 - Finite State Transducers

- An example of FST modelling some spelling rules of French



2.8 - Finite State Transducers

- A FST over an input alphabet Σ_i and an output alphabet Σ_o can be viewed as a FSA over an alphabet $\Sigma_i \times \Sigma_o$ with some adaptations, so that some results can be transposed from FSA to FST.

- A FST is associated with two FSA : its first projection, which recognizes its input language, and its second projection, which recognizes its output language.

- The following usual operations on FSAs are transposed to FSTs : union, concatenation, Kleene closure. Intersection is transposable only for ε-free FSTs.

2.8 - Finite State Transducers

- We define two additional operations on FSTs :

✓ **Inversion** : the inverse T^{-1} of a FST T results from a simple switching between the input and the output alphabets.

✓ **Composition** : if T_1 is a FST from Σ_1 to Σ_2 and T_2 is a FST from Σ_2 to Σ_3, then T_1 o T_2 maps from Σ_1 to Σ_3 .

Computation principle for T_1 o T_2 transitions: *If (q_1, a, q_2, b) is a T_1 transition and if (q'_1, b, q'_2, c) is a T_2 transition, then ((q_1,q'_1) a, (q_2, q'_2), c) is a T_1 o T_2 transition (if b = ε, one of the composed transitions can be an implicit silent transition)*

2.8 - Finite State Transducers

1.

a) Determine a FST T_1 that implements the following spelling rule of contraction in French: if *"de le"* is immediately followed by a word starting with a consonant, it is contracted into *"du"*. For instance, *"de le grand homme"* becomes *"du grand homme"* but *"de le arbre"* remains unchanged.

b) Modify the transducer presented in the lecture for implementing the following rules :
"à le" ⟶ *"au"* *"la" or "le"+ word starting with a vowel* ⟶ "l'" *+ the word*
The resulting transducer is named T_2

c) Combine T_1 with the transducer T_2 in a good order to get a transducer T_3.

d) The transducer T_3 is used to analyse the sentence *"L'individu au chapeau arrive à l'entrée du bois"* . The input of the analysis is an automaton A_1, which realizes the segmentation of the sentence into 10 words and the output is an automaton A_2 labelled with word forms coming from a French lexicon.
Describe the operations on automata and transducers that can be used to realize this task. Then, compute A_2 with these operations.
Hints: an automaton can be viewed as a transducer realizing the identity function for some domain.

2.8 - Finite State Transducers

2. We continue with the example of the previous exercise. We consider the output automaton A_2, which can be presented as follows:
"le|la individu à le chapeau arrive à le|la entrée du|(de le) bois"
We propose to tag the words with POS labels. For this, we use the following lexicon: le → {DET,PRO}, la → {DET,PRO}, individu → {N}, à → {PREP}, chapeau → {N}, arrive → {V}, entrée → {N, PASTPART}, du → {DET}, de → {PREP}, bois → {N,V}.
A tag is composed of a word followed by an hyphen and a POS label, la-DET for instance.

a) Define the transducer T_{lex} from the lexicon that replaces every sentence built from the words included in the lexicon with all possible tagged sentences.

2. Compose A_2 with T_{lex} to produce the automaton expressing the tagging of the initial sentence.

3. In French, the tag patterns DET V and DET PASTPART are not possible. Build two minimal DFA that recognize tagged sentences including these respective patterns. Build their complements and then the intersection A_3 of these complements.

4. By composing T_{lex} with A_3, build a transducer T_{tag} that tags sentences using the previous forbidden patterns and apply T_{tag} to A_2.

2.8 - Finite State Transducers

3. The Soundex algorithm is a method commonly used in libraries and older Census records for representing people's names. It has the advantage that versions that are slightly misspelled or otherwise modified (common, for example, in handwritten census records) will still have the same representation as correctly-spelled names. (e.g., Jurafsky, Jarofsky, Jarovsky, and Jarovski all map J612).
   By using transducers, realize the following ordered tasks:

a) Keep the first letter of the name, and drop all occurrences of non-initial a,e,h,i, o,u,w,y.

b) Replace the remaining letters with the following numbers:

- b, f, p, v → 1
- c, g, j, k, q, s, x, z → 2
- d, t → 3`
- l → 4
- m, n → 5
- r → 6

2.8 - Finite State Transducers

c) Replace any sequences of identical numbers with a single number (i.e. 666 → 6)

d) Convert to the form *Letter Digit Digit Digit* by dropping digits past the third (if necessary) or padding with trailing zeros (if necessary).

2.9 - Determinization of transducers

- The determinization of a transducer is different from its determinization as automaton. The aim is to make it **deterministic on its input**.

- **Epsilon removal for transducers** : if a transducer T is such that $|T|(\varepsilon) = \emptyset$ and such that there is no loop labelled by ε as the input word and by a non empty word as the output word, then there exists a transducer T' that is equivalent to T and that is not labelled by ε as input symbol.

2.9 - Determinization of transducers

- An **unambiguous transducer** is a transducer for which each input word is the label of at most one successful path.
- An unambiguous transducer is functional but an ambiguous transducer can be functional or not.
- **Theorem (Eilenberg 74)** : any rational function ρ can be represented by an unambiguous transducer if $\rho(\varepsilon) = \emptyset$ or $\rho(\varepsilon) = \varepsilon$.

2.9 - Determinization of transducers

- Non ambiguous transducers can be locally non deterministic on their input.

  Transducers, which are locally deterministic on their input are sequential transducers.

- A **sequential transducer** is a transducer $(Q, \Sigma_i, \Sigma_o, q_0, F, \tau)$ such that :

  ✓ if $\tau(q_1, a_i, q_2, w_o)$ is any transition of the transducer, then $a_i \in \Sigma_i$ and $w_o \in \Sigma_o^*$

  ✓ if $\tau(q_1, a_i, q_2, w_o)$ and $\tau(q_1, a_i, q'_2, w'_o)$ are transitions of the transducer,

  then $q_2 = q'_2$ and $w_o = w'_o$

2.9 - Determinization of transducers

- The transduction that is realized by a sequential transducer is functional : it is called a **sequential function**.

- Sequential transducers can be generalized by introducing the possibility of generating an additional output string at final states. Such transducers are called **subsequential transducers**.

2.9 - Determinization of transducers

- Subsequential transducers are extended to **p-subsequential** transducers to express ambiguity in a limited manner. At the final states, it is possible to generate one output string chosen among p possibilities at most.

- There is an algorithm to determine if a transducer is equivalent to a p-subsequential transducer. Its principle is to try to determinize the transducer by postponing the transition choices on input until they become deterministic.

2.9 - Determinization of transducers

1. Sequentialize the following transducers, if it is possible. In all cases, the initial state is 0 and the final states are marked with an asterisk.

Transducer T1			
Source state	**Target state**	**Input word**	**Output word**
0*	0*	b	b
0*	0*	c	c
0*	1	a	b
0*	2*	a	a
1	0*	b	b
2*	1	a	b
2*	2*	a	a

Transducer T2			
Source state	**Target state**	**Input word**	**Output word**
0	0	a	bc
0	1*	a	bd
1*	1*	a	b

2.9 - Determinization of transducers

2. Sequentialize the following transducers, if it is possible. In all cases, the initial state is 0 and the final states are marked with an asterisk.

Transducer T3			
Source state	**Target state**	**Input word**	**Output word**
0	0	a	b
0	1	a	d
0	2*	a	bd
1	2*	b	a

Transducer T4			
Source state	**Target state**	**Input word**	**Output word**
0	1	a	b
0	2*	a	bcd
0	3	a	bc
1	2*	b	cd
2*	2*	b	ε
3	2*	b	d

2.10 - Application of Finite State methods to NLP

- **Information retrieval and text indexation**: regular expressions are used efficiently to retrieve specific words in texts (in editors for instance) (Ken Thompson 1968).

  Regular expressions, automata and transducers are also used to index texts automatically (Mehryar Mohri 1995).

- **Tokenization**: regular expressions are used to segment texts into sentences and tokens automatically.

2.10 - Application of Finite State methods to NLP

- **Phonology and morphology**: phonological and morphological rules are represented separately by transducers (Johnson 1972, Ronald Kaplan & Martin Kay 1981, Kimmo Koskenniemi 1983, Lauri Kartunnen 1983). Then, these transducers are combined by different operations to represent the simultaneous action of all these rules.

- **Speech processing**: transducers implement text-to-speech systems in an efficient way.

2.10 - Application of Finite State methods to NLP

- **Representation of large scale dictionaries**: pronunciation or morphological dictionaries, taken as extensive lists of units, can be represented by automata and p-subsequential transducers, which can be minimized (Mehryar Mohri 1995).

2.10 - Application of Finite State methods to NLP

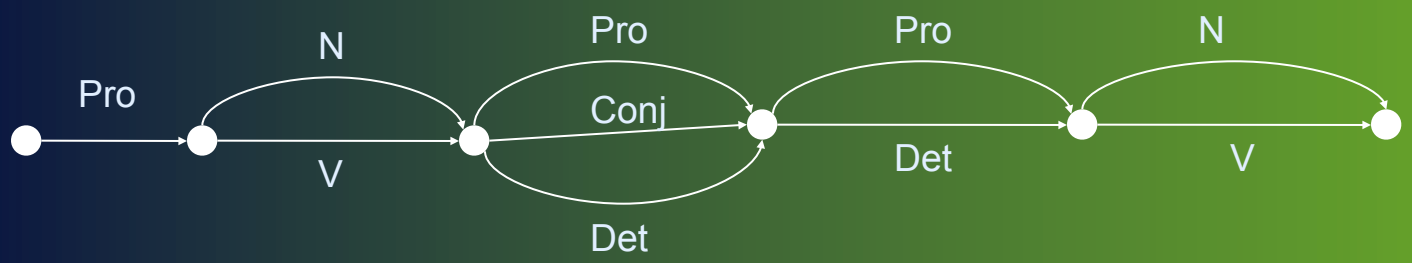
- **Morpho-syntactic tagging:** it aims at labelling the words of text with morpho-syntactic information. The fineness of the tagset is decisive for the tagging quality.

  Brill taggers start with labelling texts in a rough manner and then correct the labelling step by step by using rules. The list of correction rules is learned automatically from a training corpus.

  A way of implementing Brill taggers is to use transducers (Emmanuel Roche & Yves Schabes 1997). Every correction rule is represented by a specific transducer and then all transducers are composed together.

2.10 - Application of Finite State methods to NLP

1. We consider English sentences which are tagged with POS. A tag of a sentence appears as an automaton , the states of which are positions between words of the sentence, and the transitions are labelled with POS.
For instance, here is the automaton A_0 corresponding to the sentence:"He hopes that this works".



a) In English, a Det label cannot immediately follow another Det label. Define an automaton A_1 that recognizes sentence tags including a sequence of two Det labels at least.

b) Build its complement $A_1^{C,}$ and then the intersection $A_0 \cap A_1^C$. What is the meaning of this intersection ?

1. In English, a V label cannot immediately follow a Det label. Define an automaton A_2 that recognizes sentence tags including such sequences. Then, build its complement A_2^C.

a) By using operations on automata, simplify the automaton A_0 to drop the two kinds of forbidden POS sequences mentioned above.

54

2.10 - Application of Finite State methods to NLP

2. Brill Tagger

We want to implement a Brill tagger for English using transducers. We simplify the sequence of correction rules that are learned from a reference corpus and we assume that the sequence is composed of the two following rules:

1. np vbn → np vbd
2. vbd by → vbn by

In theses rules, “np” stands for proper noun, “vbn” for verb in past participle form, “vbd” for verb in past tense and “by” is the preposition “by”.
The meaning of the rules is the following: in a tagged text, every sequence of tags that matches the left hand side of the rule is replaced with the right hand side. For instance, consider the sentence: “$John_{np}\ Lennon_{np}\ was_{bedz}\ shot_{vbd}\ by_{by}\ Chapman_{np}$ “. After application of rule 1, the sentence remains unchanged. After application of rule 2, the sentence becomes: “$John_{np}\ Lennon_{np}\ was_{bedz}\ shot_{vbn}\ by_{by}\ Chapman_{np}$ “.

- Represent the actions of the two rules on tagged texts with two transducers T_1 and T_2
- Compose T_1 o T_2 to represent the action of rule 1 followed by rule 2. We obtain a transducer T, which represents one pass of the Brill tagger.
- Represent the tagging of the initial sentence with an automaton A. We consider A as a transducer representing the function identity. Under this condition, compose A o T to compute the output tagging of the sentence.

55