Introduction to Natural Language Processing

ELECTRONIC LEXICA

Martin Rajman

Martin.Rajman@epfl.ch

and

Jean-Cédric Chappelier

Jean-Cedric.Chappelier@epfl.ch

Artificial Intelligence Laboratory

Objectives of this lecture

- ➥ How to handle lexica (list of words) electronically
- ➥ Show and compare principal approaches for representing and managing of huge sets of strings

Lexicon

What for? $\longrightarrow$ to recognize and classify "*correct words*" of the language

"correct"? ☞ see next slide

for "incorrect" forms ☞ specific treatments (see next lecture)

What content?

List of records structured in **fields**, describing the correct forms, e.g.:

- surface form: `boards`
- Part-of-Speech tag: `Np` (= Noun plural)
- lemma: `board#Ns` ($\longrightarrow$ a surface form **and** a PoS tag)
- probability: `3.2144e-05`
- pronunciation: `b oa r d z`
- etc...

☞ set of "records" identified by a **reference** (e.g. a database with primary keys)

Correct word??

The notion of "correct word" is difficult to define from an absolute point-of-view:

"*credit card*", "*San Fransisco*", "*co-teachning*": 1 or 2 words?

Is "*John's*" from "*John's car*" one single word? Or are they two words? Is "*'s*" a word?

And it's even worse in languages having *aglutinative morphology* (e.g. German): see Morphology lecture.

And what about: "*I called SC to ask for an app.*", or "*C U*"

☞ definition of words **depends on the application**

Should carrefully think about it!

If you'd like the lexicon to be more portable/universal: choose minimal *tokens* and let a *properly desing tokenizer* glue the adequate pieces for each dedicated application.

Lexicon

A lexicon must provide a set of functions (interface):

- Insertion/deletion of a record or of some field
- Existence test within one or several field(s), often the surface form (check if a given form is in the lexicon)
- Extraction from one or several fields
  (e.g. *all plural nouns ending in ”en”*)
- Listing of the whole content
- ...

Example

field value — by_value access f^o → reference

reference — by_reference access f^o → field value

reference	surface form	PoS	lemma	prononc.	...
...					
25	board	Ns	board#Ns	b oa r d	...
26	boards	Np	board#Ns	b oa r d z	...
...					
34	fly	Vx	fly#Vx	f l i	...
35	fly	Ns	fly#Ns	f l i	...
...					

by_value$_{\text{surface}}$(fly) → $\{34, 35\}$

by_ref$_{\text{PoS}}$(26) → $\mathtt{Np}$

All PoS tags for "`fly`":

$$\text{by\_ref}_{\text{PoS}}(\text{by\_value}_{\text{surface}}(\text{fly})) = \{\mathtt{Vx}, \mathtt{Ns}\}$$

Field representation

☞ External vs. **Internal structure** (i.e. serialization vs. memory representation)

Internal structure: suited for an efficient implementation of the two access methods (by value and by reference) for each field

- ☞ not necessarily the same for all fields
- ☞ not even necessarily the same for the two methods of a given field

Surface forms: methods implementation

- Lists/Tables
- Hash Tables
- Tries (= lexical trees)
- Finite-State Automata (FSA)
- Transducers (FST)

List/Tables implementation

☞ needs an order on the values (e.g. alphabetical order)

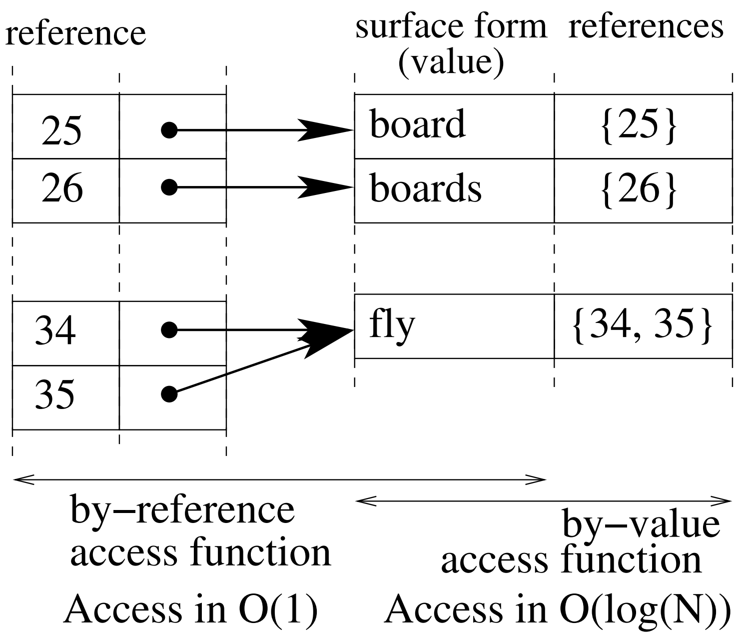
:-) easy and fast to implement

:-) efficient by-reference access function ($\mathcal{O}(1)$)

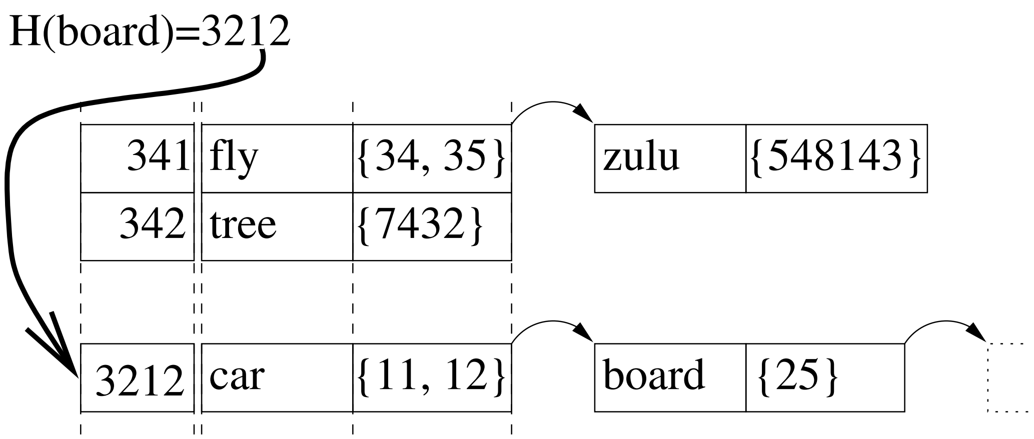
:-(access in $\mathcal{O}(\log N)$, insertion in $\mathcal{O}(N)$ (N = number of records)

:-(large size (replication of all (sub-)strings)

by-reference access function: list of pointers ordered by reference

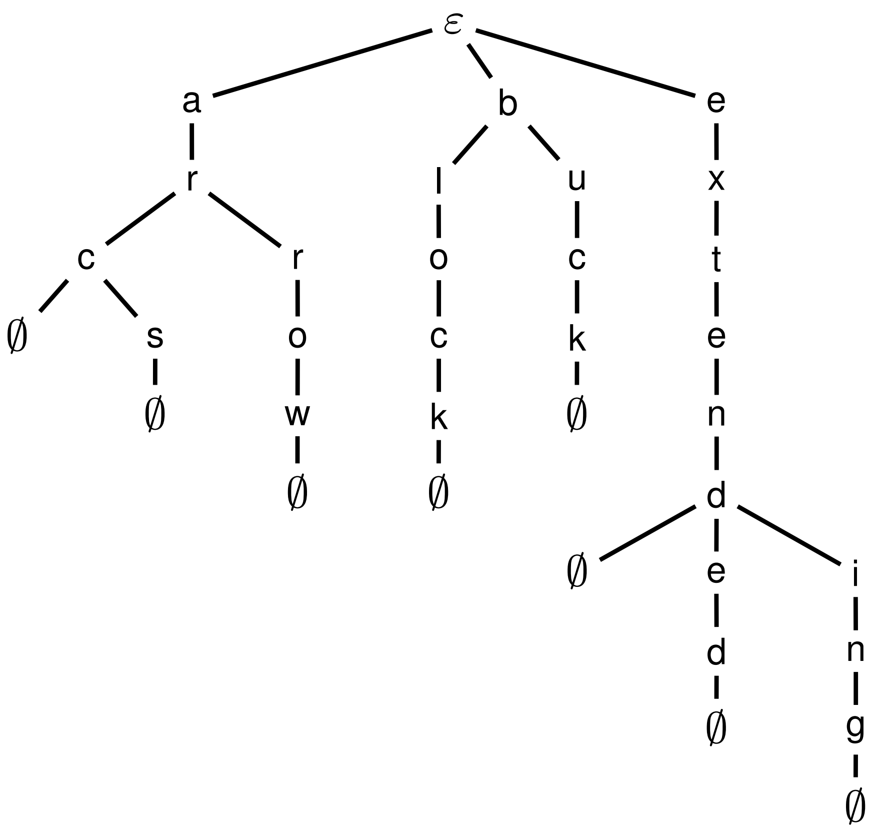


Hash Tables



- :-) easy and fast to implement
- :-| complexity of access and insertion difficult to predict (collisions)
- :-(no by-reference access-function ($\longrightarrow$ extra inversion table)
- :-(large size (replication of all (sub-)strings)

Tries: lexical trees



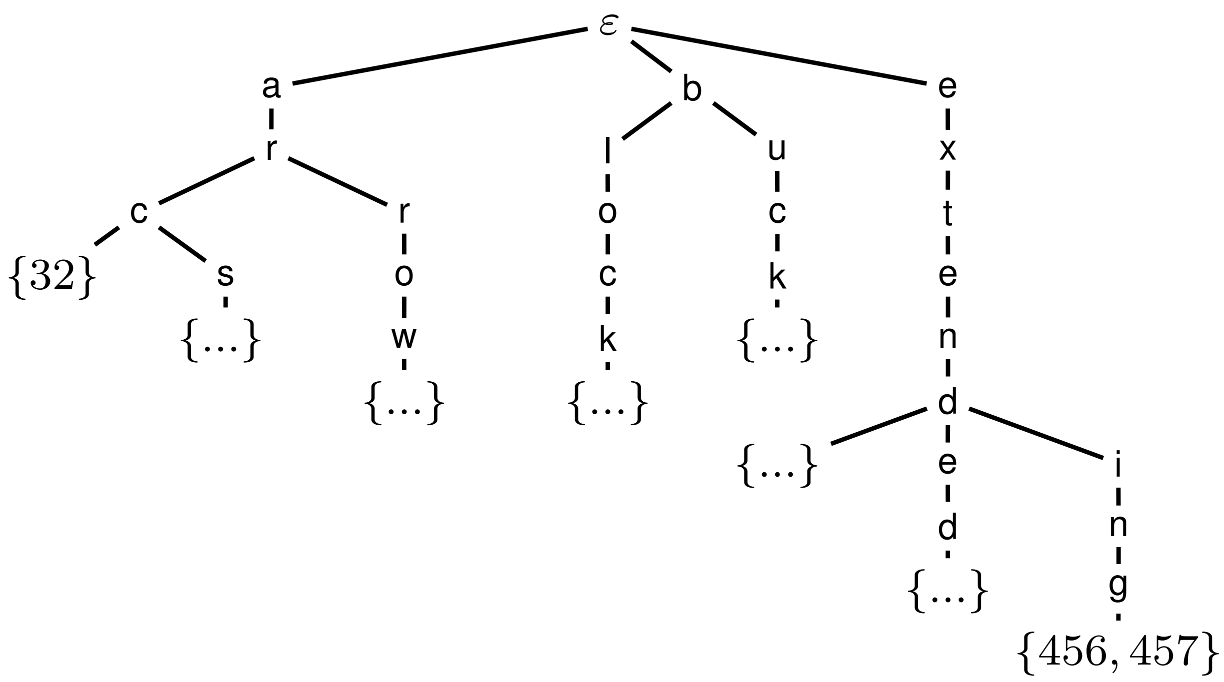
arc
arcs
arrow
block
buck
extend
extended
extending

☞ Requires an end-of-word marks

Tries (2)

Notice that: Existence test $\neq$ access function

For access function ☞ Requires the adjunction of reference labels



arc $\{32\}$

...

extending

$\{456, 457\}$

Tries: by-reference access-function

- bidirectional (father-son) links: consuming space
- inversion codes:

  $\simeq$ list of the numbers of the sons

  ! no need to code unambiguous positions

  example:

  arrow $\longrightarrow$ 1,(1,)2

  block $\longrightarrow$ 2

  buck $\longrightarrow$ 2,2

  extend $\longrightarrow$ 3

  extended $\longrightarrow$ 3,(1,1,1,1,1,)2

Tries: Summary

:-) access and insertion in $\mathcal{O}(1)$

:-| space: substring sharing but not optimal

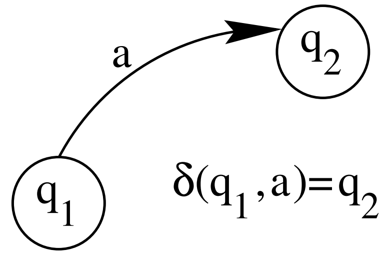
:-| Implementation complexity

$\Rightarrow$ good for ”dynamic” lexica (i.e. build on-line: typically ”*user lexica*”)

FSA

Reminder?

- Q: (finite) set of states
- Σ: (finite) alphabet
- δ: arcs (mapping from $Q \times \Sigma$ to $Q \cup \{0\}$)
- $q_0 \in Q$: inital state
- $F \subset Q$: final states



<u>Theorems:</u>

- Equivalence between DFSA and NFSA
- Equivalence between NFSA with and without ε
- Equivalence between regular expressions and DFSA
- For a given regular language, existence of a unique **minimal** DFSA

Regular expressions (regexp)

Let L_1 and L_2 be two subsets of Σ^*

- $L_1\, L_2$ is the subset of Σ^* resulting of the concatenation of elements of L_1 and elements of L_2
- L_1^i the set $\underbrace{L_1 ... L_1}_{i \text{ fois}}$ (with $L_1^0 = \varepsilon$)
- L_1^* the set $\bigcup_{i=0}^{\infty} L_1^i$

A regexp represents a subset of Σ^* defined by :

- the empty set and ε (which are 2 differents objects) are not represented
- For $a \in \Sigma$, the regexp a represents the set $\{a\}$
- if r and s are two regexp representing respectively the sets R and S :

  $r|s$ stands for $R \cup S$ $\quad$ $r\,s$ stands for $R\,S$ $\quad$ $r*$ stands for R^*

Example: $(a|b)*|c$

Implementation of methods for lexica with FSA

- :-)) regexp (e.g. numbers, dates, ...)
- :-) access in $\mathcal{O}(1)$
- :-) optimal size (minimal number of states)
- :-(Implémentation
- :-(insertion

Implementation of methods for lexica with FSA: Access function?

Two problems:

1. How to attach a reference to a string in an FSA?
2. How to represent several times the same string? (i.e. different records with the same surface form field)

   ☞ Attach **several** references to a single string

Association of a reference:

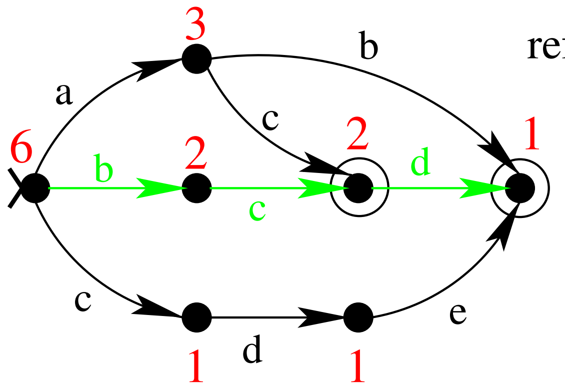
1. a priori order on Σ ("*alphabetical*" order)
2. then reference = rank of the string in the alphabetical order of strings

Access function? (2)

Construction: associate to each state q, the number $N_f(q)$ of paths leading to final states (including itself if it is a final state).

Example:

1 ab
2 ac
3 acd
4 bc
5 bcd
6 cde



ref(bcd) = 0 + (3+0) + (0+1) + (0+1)
= 5

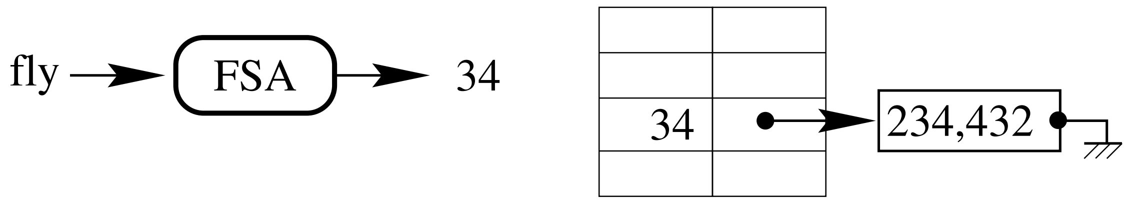
Construction of the reference: during the walk through the FSA:

$$n(q_{t+1}) = n(q_t) + \sum_{\substack{q:q_t \to q \\ q < q_{t+1}}} N_f(q) + (\text{final?}(q_{t+1}))$$

with $n(q_0) = \text{final?}(q_0)$, $= 0$ except if ε is recognized by the FSA

Access function? (3)

Representation of several occurences of the same string: needs an additional association table that provides the extra reference numbers:



Finite language recognized by the FSA:

	...
11	car
	...
34	fly
	...
128,673	zulu

Other lexical form (repetitions of the FSA strings):

128,674	...
⋮	...
234,432	fly
⋮	...

Implementation of methods for lexica with FSA: By-reference access-function?

How to go from the reference to the string?

❶ For references larger than the number of strings in the FSA: inverse table of the additional association table

$\longrightarrow$ it provides a reference "contained in" the FSA

By-reference access-function? (2)

❷ For references ”contained in” the FSA: reference driven walk through the FSA:

❶ $m(q_0)$ =reference

❷ from state q_t go to first state q_{t+1} such that $\left(\sum_{\substack{q:q_t \to q \\ q \leq q_{t+1}}} N_f(q) \right) \geq m(t)$

❸ update m:

$$m(t+1) = m(t) - \sum_{\substack{q:q_t \to q \\ q < q_{t+1}}} N_f(q) - (\text{final?}(q_{t+1}))$$

❹ stop in the final state q_T where $m(q_T) = 0$

Summary

	existence	by_value access	by_ref access
lists/tables	✘	✘	✘
hash-tables	✘	✘	✘
Tries	✘	–	–
Tries + labeled leaves	✘	✘	–
Tries with invertion[a]	✘	✘	✘
FSA	✘	–	–
FSA + numeration	✘	✘	✘
Transducers	✘	✘	✘

[a] e.g. bidirectional links or inversion codes

Keypoints

- Usage of lexica: recognition and classification of language forms
- Principal functions of lexica: existence test, by value and by reference access functions
- Principal approaches for surface-form field representation: tries, automata

References

[1] A. Aho, J. Ullman, *Concepts fondamentaux de l'Informatique*, pp. 235-308, 311-365, Dunod, 1993.

[2] D. E. Knuth, *The Art of Computer Programming, V. 1, Fundamental Algorithms*, pp. 232-424, Addison-Wesley, 1997.

[3] J. E. Hopcroft, R. Motwani, J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, pp. 28-34, 37-80, Addison-Wesley, 2001.

[4] D. Jurafsky & J. H. Martin, *Speech and Language Processing*, pp. 21-49, Prentice Hall, 2000.

[5] E. Roche, Y. Schabes, *Finite-state Language Processing*, pp. 1-14, A Bradforf Book, 1997.

[6] M. G. Ciura, S. Deorowicz, *How to squeeze a lexicon*, Software – Practice and Experience, vol. 31, pp. 1077-1090, 2001.