Language and Computation

week 3, Tuesday, January 28, 2014

Tamás Biró
Yale University
tamas.biro@yale.edu
http://www.birot.hu/courses/2014-LC/

Practical matters

- Sections
- Post-readings: JM Chapters 2, 3.
- Pre-readings: JM 4.1-4.3 (then: 5.1-5.3).
- **Python:** this week H2, next week chapters 3 and 4

Today

- On linguistics:
  Morphology from an NLP perspective

- On computational skills:
  From mathematical abstraction to pseudo-codes

- I suppose the details in JM can be understood and learned based on the lectures. Please let me know asap if this is not the case.

Morphology

meg	szent	ség	telen	ít	het	etlen	ei	tek	ben
PERF	holy	ness	un	turn_into	able	un	PLUR	your(pl)	in

‘in your (pl.) things that cannot be desecrated’

Levels of linguistics

- **Phonetics:** sounds (articulation, acoustics, perception)
- **Phonology:** the sound system of a language
- **Morphology:** words
- **Syntax:** sentences
- **Discourse:** texts

+ **Semantics:** meanings on all levels

Morphology

- **Derivational morphology:** word formation.

- **Inflectional morphology:** rendering words syntactically appropriate to their context:
  - *Verbal morphology:* e.g., tense and aspect; number-person-gender agreement
  - *Nominal morphology:* plural formation, case, etc.
  - Etc.

Morphology

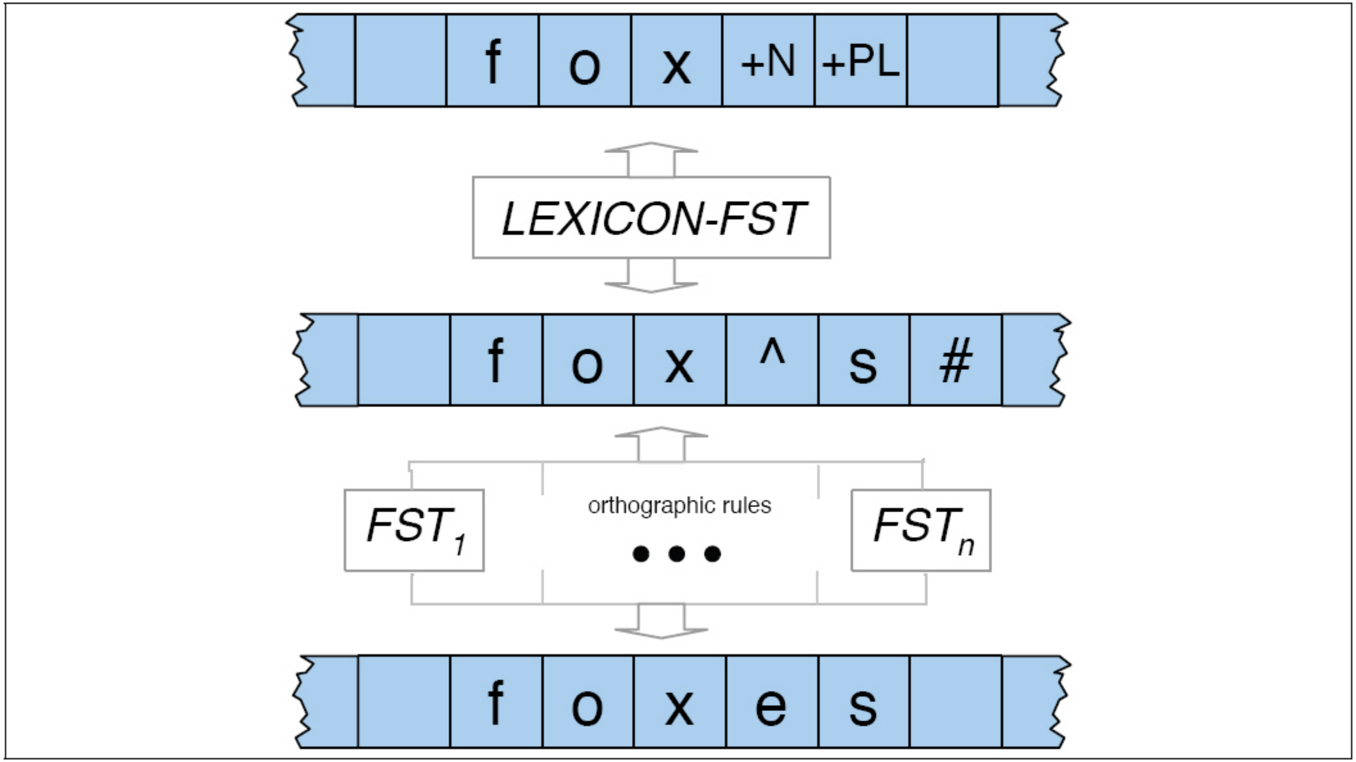
- **Morpheme:** smallest linguistic unit that bears a meaning
  - **root** ($\pm$aka **stem**) vs. **affix**
  - **free** morphemes vs. **bound** morphemes
- **Word:** ???
- Morphological processes:
  - **Affixation:** prefixes, suffixes, infixes, circumfixes
  - **Conversion** (aka zero affixation)
  - **Reduplication** (partial and total)
  - **Compounding**

Morphology in NLP

Based on phonological segments (phonemes, allophones, IPA-characters) vs. based on orthography?

- Preprocessing of texts:
  - **Lemmatization**: undo inflectional morphology (only)
    Lemma: aka base form / dictionary form / citation form.
  - **Stemming**: find the root/stem (with a crude heuristics).
- Text generation: step subsequent to syntax.
- Spell checkers & co.
- Good approximation provided by FSA/regex.

FST in NLP: an example



Transducers and finite-state morphology

Automata and transducers

Automaton:

defines a set

Transducer:

defines a mapping

in case of "text-like" information:

Automaton:

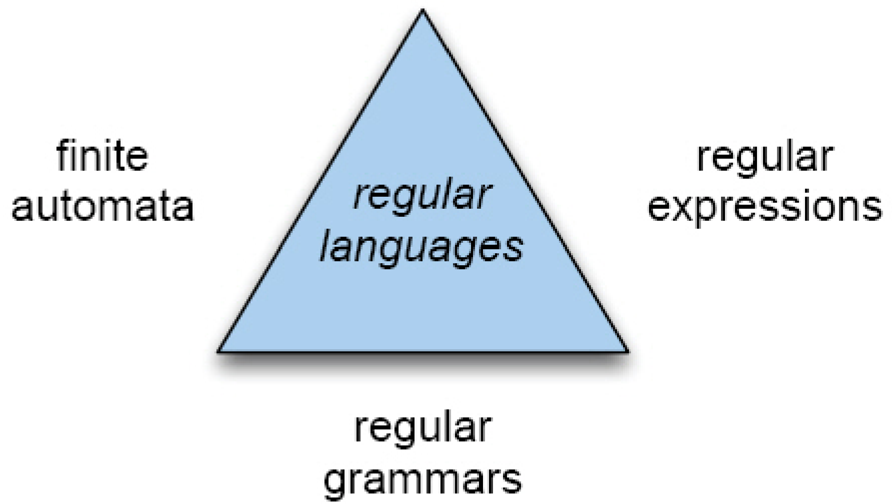
- Input: string $\in \Sigma^*$
- Output: accept or reject.

Transducer:

- Input: string $\in \Sigma^*$
- Output: string $\in \Delta^*$

Various machineries

The sets of formal languages accepted / generated by



are the same! The **regular languages**.

Various machineries

. . . various perspectives:

- A formal grammar *generates* the strings of a language.
- A regular expression *matches* the strings of a language.
- An automaton *accepts* the strings in a language.
- A Markov model *emits* the strings of a language.

Deterministic finite state automaton

- Q finite set of states
- Σ (input) alphabet
- $q_0 \in Q$ start state
- $F \subseteq Q$ set of final states (can be empty)
- $\delta(q, i)$ transition function $Q \times \Sigma \cup \{\epsilon\} \rightarrow Q$

Deterministic finite state transducers

- Q finite set of states
- Σ input alphabet and Δ output alphabet
- $q_0 \in Q$ start state
- $F \subseteq Q$ set of final states (can be empty)
- $\delta(q, i)$ transition function $Q \times \Sigma \cup \{\epsilon\} \to Q$
- $\sigma(q, i)$ output function $Q \times \Sigma \cup \{\epsilon\} \to \Delta \cup \{\epsilon\}$

Deterministic finite state automaton

Automaton **accepts** input string $i = i_0 i_1 \ldots i_{n-1}$ iff

there is a series of states $q_0, q_1, \ldots q_{n-1}, q_n$ $(\in Q^{n+1})$ such that

1. $q_{j+1} = \delta(q_j, i_j)$ for all $j < n$, and
2. q_0 is the start state, and
3. $q_n \in F$ is a final state.

NB: i_j can also be ϵ, beside the letters of i.

Non-deterministic finite state automaton

Automaton **accepts** input string $i = i_0 i_1 \ldots i_{n-1}$ iff

there is a series of states $q_0, q_1, \ldots q_{n-1}, q_n$ $(\in Q^{n+1})$ such that

1. $q_{j+1} \in \delta(q_j, i_j)$ for all $j < n$, and
2. q_0 is *the* start state, and
3. $q_n \in F$ is *a* final state.

NB: i_j can also be ϵ, beside the letters of i.

Deterministic finite state transducer

Transducer **maps** input string $i = i_0 i_1 \dots i_{n-1}$
. onto output string $o = o_0 o_1 \dots o_{n-1}$ iff

there is a series of states $q_0, q_1, \dots q_{n-1}, q_n$ $(\in Q^{n+1})$ such that

1. $\delta(q_j, i_j) = q_{j+1}$ for all $j < n$, and
2. $\sigma(q_j, i_j) = o_j$ for all $j < n$, and
3. q_0 is *the* start state, and
4. $q_n \in F$ is *a* final state.

NB: i_j can also be ϵ beside the letters of i,
and o_j can also be ϵ beside the letters of o.

Finite state automata and transducers

What to do when in state q and reading character i?

The transition function $\delta(q, i)$ — variation on a topic:

- Deterministic FSA: $\delta(q, i) \in Q$
- Non-deterministic FSA: $\delta(q, i) \in \mathcal{P}(Q)$
- Deterministic FST: $\delta(q, i) \in (Q \times \Delta)$
- Non-deterministic FST: $\delta(q, i) \in \mathcal{P}(Q \times \Delta)$

Finite state automata and transducers

What to do when in state q and reading character i?

The transition function $\delta(q, i)$ — variation on a topic:

- Deterministic FSA: $\delta(q, i) \in Q$
- Non-deterministic FSA: $\delta(q, i) \in \mathcal{P}(Q)$
- Markov chain: $\delta(q)$ is a probability distribution on Q
- Markov model: $\delta(q)$ is a probability distribution on $Q \times \Delta$

On pseudo-codes

- The lingo when speaking about algorithms
- Half way between human language and programming languages
- Relatively straightforward to translate to your favorite programing language
- Focus on important aspects, skip over details

Running a deterministic FSA

```
function D-RECOGNIZE(tape, machine) returns accept or reject

 index ← Beginning of tape
 current-state ← Initial state of machine
 loop
  if End of input has been reached then
   if current-state is an accept state then
     return accept
   else
     return reject
  elsif transition-table[current-state,tape[index]] is empty then
    return reject
  else
    current-state ← transition-table[current-state,tape[index]]
    index ← index + 1
 end
```

Running a non-deterministic FSA

Q: How to have a deterministic computer simulate a non-deterministic automaton?

A: Replace states with set of states

```
function ND-RECOGNIZE(tape, machine) returns accept or reject

 agenda ← {(Initial state of machine, beginning of tape)}
 current-search-state ← NEXT(agenda)
 loop
   if ACCEPT-STATE?(current-search-state) returns true then
      return accept
   else
      agenda ← agenda ∪ GENERATE-NEW-STATES(current-search-state)
   if agenda is empty then
      return reject
   else
      current-search-state ← NEXT(agenda)
 end

function GENERATE-NEW-STATES(current-state) returns a set of search-states

 current-node ← the node the current search-state is in
 index ← the point on the tape the current search-state is looking at
 return a list of search states from transition table as follows:
   (transition-table[current-node,ϵ], index)
   ∪
   (transition-table[current-node, tape[index]], index + 1)

function ACCEPT-STATE?(search-state) returns true or false

 current-node ← the node search-state is in
```

current-search-state ← NEXT(*agenda*)
end

function GENERATE-NEW-STATES(*current-state*) **returns** a set of search-states

current-node ← the node the current search-state is in
index ← the point on the tape the current search-state is looking at
return a list of search states from transition table as follows:
(*transition-table[current-node,ϵ], index*)
∪
(*transition-table[current-node, tape[index]], index + 1*)

function ACCEPT-STATE?(*search-state*) **returns** true or false

current-node ← the node search-state is in
index ← the point on the tape search-state is looking at
if *index* is at the end of the tape **and** *current-node* is an accept state of machine
then
return true
else
return false

See you on Thursday!