HLT

Finite State Technology

University of Malta

Acknowledgements

- Richard Sproat, Morphology and Computation, MIT Press, ISBN 0-262-19314-0 (1992)
- Shuly Wintner, Lecture Notes, 2008

Outline

1. Computational Morphology
2. Revision of Formal Language Theory
3. Regular Expressions
4. Finite State Automata
5. FSAs and Morphology

Outline

1. Computational Morphology
2. Revision of Formal Language Theory
3. Regular Expressions
4. Finite State Automata
5. FSAs and Morphology

Outline

Outline

1. Computational Morphology
2. Revision of Formal Language Theory
3. Regular Expressions
4. Finite State Automata
5. FSAs and Morphology

Outline

1. Computational Morphology
2. Revision of Formal Language Theory
3. Regular Expressions
4. Finite State Automata
5. FSAs and Morphology

Computatational Morphology

- Morphology involves the relation between word forms and their constituent morphemes.
- `enlargement`, `en + large + ment`
- Computational morphology is the design of algorithms which computations over that relation.
- Computational morphology is two way:
  - Morphological Analysis
  - Morphological Synthesis
- Computational Morphology is not just about strings, but is also about meanings.

Challenges for Computational Morphology

- Handling Segmentation: what are the parts into which the word is broken.
- Handling Morphotactics:
  - handling the order in which the parts combine together
  - computing the result
- Handling Phonological Alternations
  - pity is realized as piti in pitilessness
  - die becomes dy in dying
- Computational Morphology involves a concrete representation of the lexicon.

Representing the Lexicon

- Finite-state automata are a good model for representing the lexicon.
- They are also perfectly adequate for representing dictionaries (lexicons+additional information)
- They are also for describing morphological processes that involve concatenation etc.
- A natural extension of finite-state automata - finite-state transducers - are a perfect model for most processes known in morphology and phonology including non-segmental ones.

Formal Language Theory

- Formal languages are defined with respect to a given alphabet Σ, which is a finite set of symbols, each of which is called a letter.
- A finite sequence of letters is called a string.
- String Length $\mid w \mid$
- Concatenation $w_1.w_2$
- Exponent $w^n = w_1 \ldots w_{n-1}.w_n$
- Reversal w^R. If $w =< w_1, w_2 \ldots w_n >$ then $w^R =< w_n, w_{n-1} \ldots w_1 >$
- Substring:
  If $w =< x_1 \ldots x_n >$ then
  for any i, j such that $1 \leq i \leq j \leq n$
  $< x_i \ldots x_j >$ is a substring of w.

Prefix and Suffix

- Two special cases of substring are *prefix* and *suffix*.
- If $w = w_l.w_c.w_r$ then
  - w_l is a prefix of w and
  - w_r is a suffix of w

Example

- Let $\Sigma = a, b, c, \ldots y, z$ be an be an alphabet and let $w =$ *indistinguishable* a, string over Σ.
- Then ϵ *in, indis, indistinguish* and *indistinguishable* are prefixes of w , while ϵ *e, able, distinguishable* and *indistinguishable* are suffixes of w.
- Substrings that are neither prefixes nor suffixes include *distinguish, gui* and *is*.

Formal Language

- Given an alphabet Σ, the set of all strings over Σ, is denoted by Σ^*
- A *formal language* over Σ is a subset of Σ^*.

Example

- Let $\Sigma = a, b, c, \ldots y, z$ be an be an alphabet.
- The following are formal languages over σ
- Σ^*
- the set of strings consisting of consonants only;
- the set of strings consisting of vowels only;
- the set of strings each of which contains at least one vowel and at least one consonant;
- the set of palindromes;

Lifting String Operations to Languages

- String operations can be lifted to languages
- if L is a language then the *reversal* of L, denoted L^R, is the language

$$\{w \mid w^R \in L\}$$

- if L_1 and L_2 are languages, then the concatenation of L_1 and L_2,

$$L_1.L_2 = \{w_1.w_2 \mid w_1 \in L_1 \text{ and } w_2 \in L_2\}$$

Examples

- $L_1 = \{$i, you, he, she, it, we, they $\}$
- $L_2 = \{$smile, sleep $\}$
- $L_2^R = \{$elims, peels$\}$
- $L_1.L_2 = \{$ismile, isleep, yousmile, yousleep, $\ldots$ theysleep$\}$

Kleene Closure

- The Kleene Closure of L is denoted L^* and defined as

$$\bigcup_{i=0}^{\infty} L^i.$$

- Note also that

$$L^+ = \bigcup_{i=1}^{\infty} L^i.$$

- **Example**

  Let $L = \{dog, cat\}$.

  - $L^0 = \{\epsilon\}$.
  - $L^1 = \{dog, cat\}$,
  - $L^2 = \{dogdog, catcat\}$,

Regular Expressions

- Regular expressions are a formalism for defining (formal) languages.
- Their “syntax” is formally defined and is relatively simple.
- Their “semantics" is sets of strings
- The denotation of a regular expression is a set of strings in some formal language.

- 0 is an RE
- ϵ is an RE
- if $a \in \Sigma$ is a letter then a is an RE
- if r_1 and r_2 are REs, then so are $r_1 + r_2$ and $r_1.r_2$
- if r is an RE then so is $(r)*$
- nothing else is an RE over Σ

Regular Expressions

Examples

- Let $\Sigma = a, b, c, \ldots y, z$ be an be an alphabet. Some REs over Σ include
- 0
- ϵ
- $((c.a).t)$
- $(((m.e).(o))^*.w)$
- $(a + (e + (i + (o + u))))$
- $(a + (e + (i + (o + u))))^*$

For every RE r its *denotation* $[\![r]\!]$ is defined as follows:

- $[\![0]\!] = 0$
- $[\![\epsilon]\!] = \{\epsilon\}$
- if $a \in \Sigma$ is a letter than $[\![a]\!] = a$
- if r_1 and r_2 are REs whose denotations are $[\![r_1]\!]$ and $[\![r_2]\!]$, then
    - $[\![r_1 + r_2]\!] = [\![r_1]\!] \cup [\![r_2]\!]$
    - $[\![r_1 . r_2]\!] = [\![r_1]\!] . [\![r_2]\!]$
    - $[\![(r_1)^*]\!] = [\![r_1]\!]$

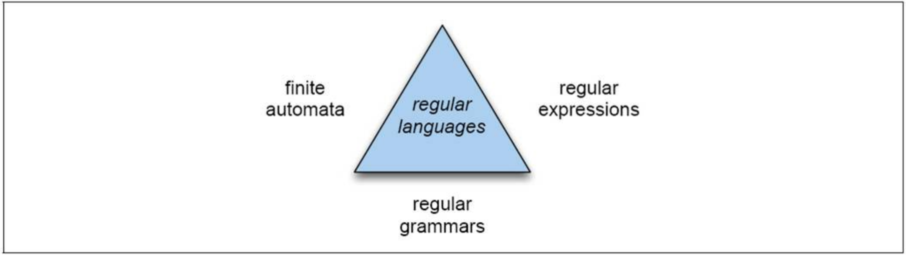
Example

RE	DENOTATION
0	0
a	$\{a\}$
$((c.a).t)$	$\{c.a.t\}$
$(((m.e).(0)*).w)$	$\{mew, meow, meoow, meooow \ldots\}$
$(a + (e + (i + (o + u))))$	$\{a, e, i, o, u\}$
$(a + (e + (i + (o + u))))^*$	all strings of vowels

Regular Languages - Definition

- Definition
  - A Language is **regular** if it is the denotation of some regular expression.
  - Not all formal languages are regular
- Closure
  - A class of languages is said to be **closed** under some operation if and only if whenever two languages are in the class, the result of performing the operation on the two languages is also in this class.

Equivalent Ways of Describing Regular Languages



Regular Expressions Closure Properties

Regular languages are closed under:

- Union
- Intersection
- Complementation
- Difference
- Concatenation
- Kleene-star

Some Things that are Regular Languages

- Zero or more a's followed by zero or more b's
- The set of words in an English dictionary
- Dates
- URLs
- English?

Some Things that are not Regular Languages

- Zero or more a's followed by exactly the same number of b's
- The set of all English palindromes
- The set that includes all noun phrases of the form
    - the cat slept
    - the cat the dog bit slept
    - the cat the dog the man fed bit slept

Finite State Automata

- Automata are models of computation.
- A finite state automaton (FSA) is a five-tuple $< Q, q_0, \Sigma, \delta, F >$, where
  - Q is a set of states
  - $q_0 \in Q$ is an initial state
  - $F \subseteq Q$ is a set of final states
  - Σ is a finite set of symbols
  - δ is a relation $Q \times \Sigma \times Q$

FSA Example

Example

a

3 2 a 1

b

- $Q = \{1, 2, 3\}$
- $q_0 = 3$
- $F = \{1\}$
- $\Sigma = \{a, b\}$
- $\delta = \{(3, a, 2), (2, b, 3), (2, a, 1)\}$

Language Accepted by an FSA

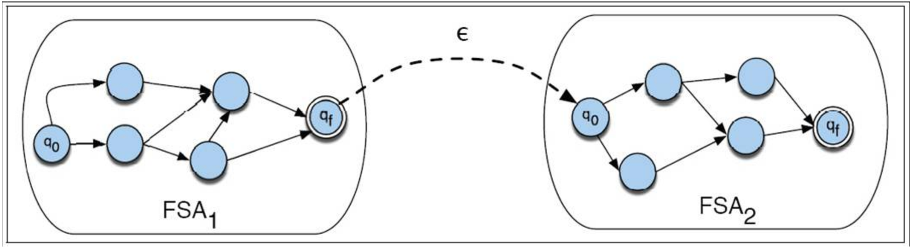
- Define the reflexive transitive extension Δ of δ
  - for every state $q \in Q, (q, \epsilon, q) \in \Delta$
  - for every string $w \in \Sigma^*$ and letter $a \in \Sigma$, if $(q, w, q') \in \Delta$ and $(q', w, q'') \in \delta$ then $(q, w.a, q'') \in \Delta$
- A string w is accepted by an automaton if and only if there exists $q_f \in Q$ such that
$$(q_0, w, q_f) \in \Delta$$
- The language accepted by a finite-state automaton is the set of all strings it accepts
- Theorem (Kleene, 1956): The class of languages recognized by finite-state automata is the class of regular languages.

Operations on FSAs

- Concatenation
- Union
- Intersection
- Minimization
- Determinization

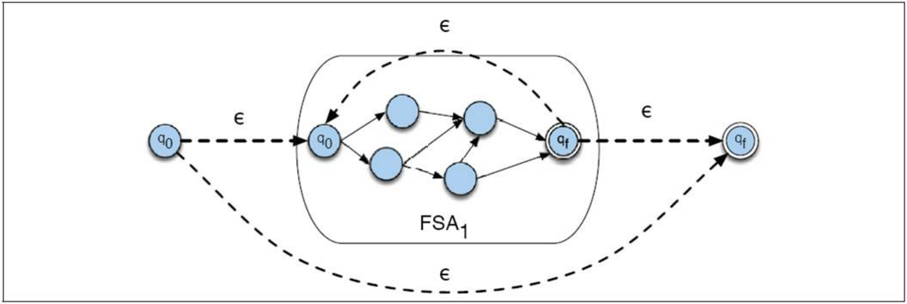
Operations on FSAs

Concatenation



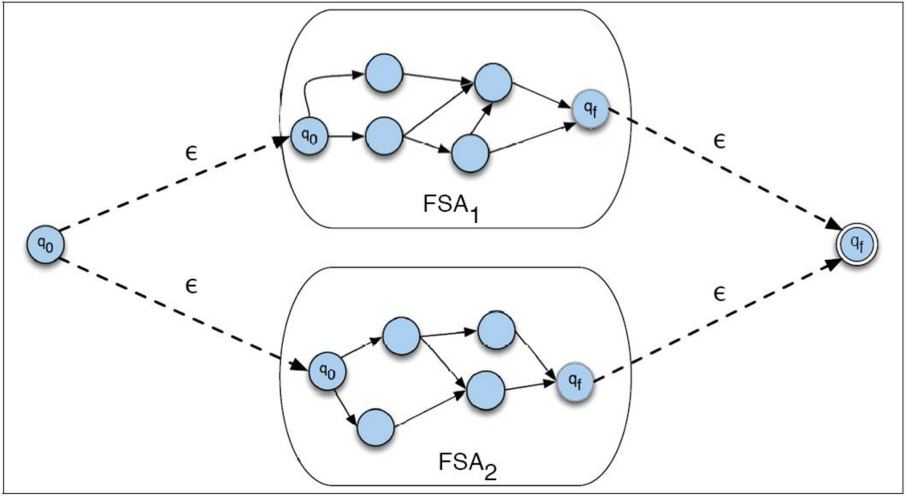
Operations of FSAs

Kleene *



Operations of FSAs

Union



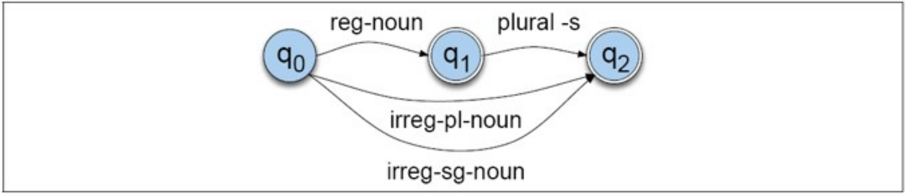
- A lexicon is a repository of words
- Full form lexicon: every word is listed explicitly
- This is sometimes impractical
- English nominal inflection

English Nominal Inflection

- With respect to plural nouns are either regular or irregular
- If regular they add s
- If irregular they may have a special plural form which includes no change

reg-noun	irreg-pl-noun	irreg-sg-noun	plural
fox	geese	goose	-s
cat	sheep	sheep	
aardvark	mice	mouse	

FSA for English Nominal Inflection



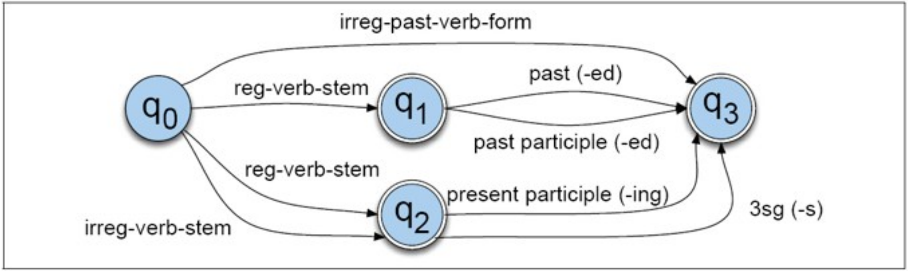
English Verb Inflection

The lexicon has

- three stem classes (reg-verb-stem, irreg-verb-stem, irreg-verb-form)
- four affix classes (*-ed* past, *-ed* participle, *-ing* participle, *-s* third-singular)

reg-verb-stem	irreg-verb-stem	irreg-past-stem	past	past-part	pres-part	3sg
walk	cut	caught	-ed	-ed	-ing	-s
fry	speak	ate				
talk	sing	eaten				
impeach		sang				

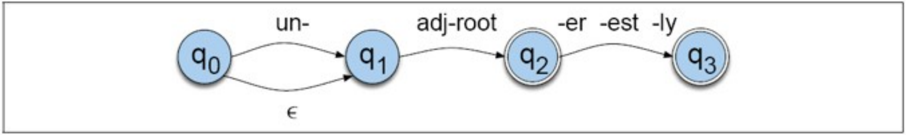
FSA for English Verb Inflection



Derivational Morphology of Adjectives

- big, bigger, biggest
- happy, happier, happiest
- unhappy, unhappier, unhappiest
- clear, clearer, clearest, clearly, unclear, unclearly
- cool, cooler, coolest, coolly
- red, redder, reddest
- real, unreal, really

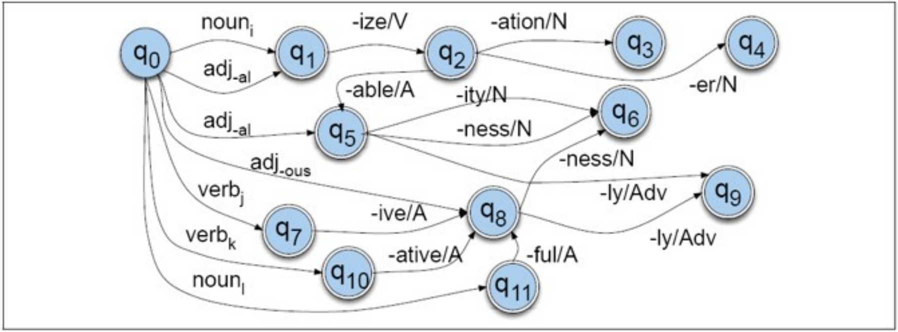
FSA for Derivational Morphology of Adjectives



Problems

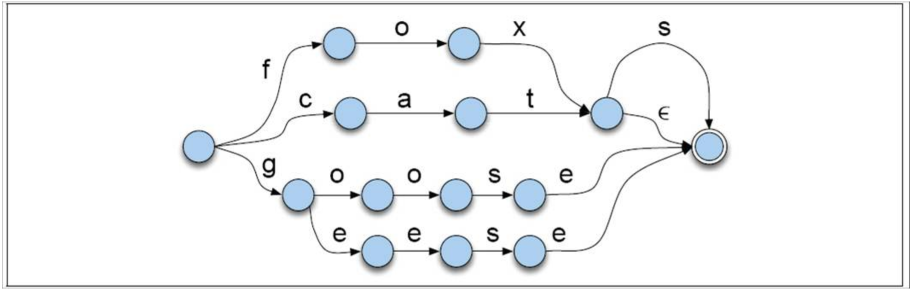
- FSA overgenerates: it will recognise forms like *unbig, smally*
- We need to set up classes of roots and specify their possible suffixes such as
  - adj-root1: adjectives that can occur with un- and -ly
  - adj-root2: adjectives that cannot so occur
- Need to handle generalisations such as:
  - verbs ending in -ize can be followed by -ation (realize, realization)
  - adjectives ending in -al or -able can take suffix -ity (equal, formal)
  - or sometimes -ness (naturalness)

Morphotactic FSA for Fragment of English Derivational Morphology



Handling the Words

- We can use these FSAs to solve the problem of morphological recognition.
- We do this by plugging *sub-lexicons* into the morphotactic FSAs defined earlier
- Given the right infrastructure, this kind of operation can be performed *algebraically*



Summary

- Finite-state automata are reversible: they can be used both for analysis and for generation.
- As recognisers, they can clearly be used for dictionary lookup.
- They are efficient computational devices.
  - Most algorithms on finite-state automata are linear.
  - In particular, the recognition problem is linear.
- Most phonological and morphological process of natural languages can be straightforwardly described using the operations under which regular languages are closed.
- The closure properties of regular languages naturally support modular development of finite-state grammars.