Introduction to Computational Linguistics

Regular Languages and Finite State Transducers

Jan-Philipp Söhn

jp.soehn@uni-tuebingen.de

January 16th, 2008

Finite State Technology

Regular languages and finite state automata

- deterministic finite state automata,
- nondeterministic finite state automata,
- finite state automata, and
- regular expressions

characterize the same class of languages, *viz.* Type 3 languages

Finite State Technology

Regular languages and finite state automata

- deterministic finite state automata,
- nondeterministic finite state automata,
- finite state automata, and
- regular expressions

characterize the same class of languages, *viz.* Type 3 languages

Finite State Technology

Regular languages and finite state automata

- deterministic finite state automata,
- nondeterministic finite state automata,
- finite state automata, and
- regular expressions

characterize the same class of languages, *viz.* Type 3 languages

Finite State Technology

Regular languages and finite state automata

- deterministic finite state automata,
- nondeterministic finite state automata,
- finite state automata, and
- regular expressions

characterize the same class of languages, *viz.* Type 3 languages

The Bigger Picture

Definition

Regular Languages:

A language L is said to be *regular or recognizable* if the set of strings s such that $s \in L$ is accepted by a DFA.

The Bigger Picture

Definition

Regular Languages:

A language L is said to be *regular or recognizable* if the set of strings s such that $s \in L$ is accepted by a DFA.

Theorem

Kleene, 1956:

The family of regular languages over Σ^* is equal to the smallest family of languages over Σ^* that contains the empty set, the singleton sets, and that is closed under Kleene star, concatenation, and union.

The Bigger Picture

Definition

Regular Languages:

A language L is said to be *regular or recognizable* if the set of strings s such that $s \in L$ is accepted by a DFA.

Theorem

Kleene, 1956:

The family of regular languages over Σ^* is equal to the smallest family of languages over Σ^* that contains the empty set, the singleton sets, and that is closed under Kleene star, concatenation, and union.

$\Rightarrow$ The family of regular languages over Σ^* is equal to the family of languages denoted by the set of regular expressions.

Regular Expressions

Given an alphabet Σ of symbols the following are all and only the regular expressions over the alphabet $\Sigma \cup \{\emptyset, 0, |, *, [,]\}$:

$\emptyset$	empty set	
0	the empty string	(ϵ, [])
σ	for all $\sigma \in \Sigma$	
$[\alpha \mid \beta]$	union (for α, β reg.ex.)	($\alpha \cup \beta$, $\alpha + \beta$)
$[\alpha\ \beta]$	concatenation (for α, β reg.ex.)	
$[\alpha$*]	Kleene star (for α reg.ex.)	

Regular Expressions: Syntactic Extensions

\$A	*contains* $\$A =_{def}$ [?* A ?*] for example: \$[a | b] denotes all strings that contain at least one *a* or *b* somewhere.
A & B	Intersection
A - B	Relative complement (minus)
$\sim$ A	Complement (negation)

More Properties of FSAs

Given the FSAs A, A_1, and A_2 and the string w, the following properties are decidable:

Membership:	$w \overset{?}{\in} L(A)$
Emptiness:	$L(A) \overset{?}{=} \varnothing$
Totality:	$L(A) \overset{?}{=} \Sigma^*$
Subset:	$L(A_1) \overset{?}{\subseteq} L(A_2)$
Equality:	$L(A_1) \overset{?}{=} L(A_2)$

More about Decidability and Closure...

Regular Relations

- Regular expressions can contain two kinds of symbols: unary symbols and symbol pairs.
  - Unary symbols (a, b, etc) denote strings.
  - Symbol pairs (a:b, a:0, 0:b, etc.) denote pairs of strings.
- The simplest kind of regular expression contains a single symbol. E.g., "a" denotes the set {a}.
- Similarly, the regular expression "a:b" denotes the singleton relation $\{\langle \mathrm{a}, \mathrm{b} \rangle\}$.
- A regular relation can be viewed as a mapping between two regular languages. The a:b relation is simply the crossproduct of the languages denoted by the expressions a and b.

Regular Relations

- Regular expressions can contain two kinds of symbols: unary symbols and symbol pairs.
  - Unary symbols (a, b, etc) denote strings.
  - Symbol pairs (a:b, a:0, 0:b, etc.) denote pairs of strings.
- The simplest kind of regular expression contains a single symbol. E.g., "a" denotes the set {a}.
- Similarly, the regular expression "a:b" denotes the singleton relation $\{\langle \mathrm{a}, \mathrm{b} \rangle\}$.
- A regular relation can be viewed as a mapping between two regular languages. The a:b relation is simply the crossproduct of the languages denoted by the expressions a and b.

Regular Relations

- Regular expressions can contain two kinds of symbols: unary symbols and symbol pairs.
  - Unary symbols (a, b, etc) denote strings.
  - Symbol pairs (a:b, a:0, 0:b, etc.) denote pairs of strings.
- The simplest kind of regular expression contains a single symbol. E.g., "a" denotes the set $\{a\}$.
- Similarly, the regular expression "a:b" denotes the singleton relation $\{\langle a, b \rangle\}$.
- A regular relation can be viewed as a mapping between two regular languages. The a:b relation is simply the crossproduct of the languages denoted by the expressions a and b.

Regular Relations

- Regular expressions can contain two kinds of symbols: unary symbols and symbol pairs.
  - Unary symbols (a, b, etc) denote strings.
  - Symbol pairs (a:b, a:0, 0:b, etc.) denote pairs of strings.
- The simplest kind of regular expression contains a single symbol. E.g., "a" denotes the set {a}.
- Similarly, the regular expression "a:b" denotes the singleton relation $\{\langle \mathrm{a}, \mathrm{b} \rangle\}$.
- A regular relation can be viewed as a mapping between two regular languages. The a:b relation is simply the crossproduct of the languages denoted by the expressions a and b.

Regular Relations

- Regular expressions can contain two kinds of symbols: unary symbols and symbol pairs.
  - Unary symbols (a, b, etc) denote strings.
  - Symbol pairs (a:b, a:0, 0:b, etc.) denote pairs of strings.
- The simplest kind of regular expression contains a single symbol. E.g., "a" denotes the set {a}.
- Similarly, the regular expression "a:b" denotes the singleton relation $\{\langle \mathrm{a}, \mathrm{b} \rangle\}$.
- A regular relation can be viewed as a mapping between two regular languages. The a:b relation is simply the crossproduct of the languages denoted by the expressions a and b.

Regular Relations

- Regular expressions can contain two kinds of symbols: unary symbols and symbol pairs.
  - Unary symbols (a, b, etc) denote strings.
  - Symbol pairs (a:b, a:0, 0:b, etc.) denote pairs of strings.
- The simplest kind of regular expression contains a single symbol. E.g., “a” denotes the set {a}.
- Similarly, the regular expression “a:b” denotes the singleton relation $\{\langle \mathrm{a}, \mathrm{b}\rangle\}$.
- A regular relation can be viewed as a mapping between two regular languages. The a:b relation is simply the crossproduct of the languages denoted by the expressions a and b.

Constructing Regular Relations

- Crossproduct: A .x. B
  - The crossproduct operator, .x., is used only with expressions that denote a regular language; it constructs a relation between them.
  - [A .x. B] designates the relation that maps every string of A to every string of B. If A contains x and B contains y, the pair $\langle x, y \rangle$ is included in the crossproduct.

Constructing Regular Relations

- Composition: A .o. B
  - Composition is an operation on relations that yields a new relation. [A .o. B] maps strings that are in the upper language of A to strings that are in the lower language of B.
  - If A contains the pair $\langle x, y \rangle$ and B contains the pair $\langle y, z \rangle$, the pair $\langle x, z \rangle$ is in the composite relation.

Finite-State Transducer

Definition

A finite-state transducer is a 6-tuple $(\Sigma_1, \Sigma_2, Q, i, F, E)$ where

Σ_1 is a finite alphabet,
(called the *input alphabet*)

Σ_2 is a finite alphabet,
(called the *output alphabet*)

Q is a finite set of *states*,

$i \in Q$ is the *initial state*,

$F \subseteq Q$ the set of *final states*, and

$E \subseteq Q \times ({\Sigma_1}^* \times {\Sigma_2}^*) \times Q$
is the set of edges.

Properties of Transducers

- A transducer is functional iff for any input there is at most one output.
- A transducer is sequential iff no state has more than one arc with the same symbol on the input side.

Replacement Operators

- Unconditional obligatory replacement:

  A $\rightarrow$ B $=_{def}$ [[No_A [A .x. B]]* [No_A]

- Unconditional optional replacement:

  A ($\rightarrow$) B $=_{def}$ [[No_A [A .x. A | A .x. B]]* [No_A]]

- Contextual obligatory replacement:

  A $\rightarrow$ B $\|$ L __ R

  meaning: "Replace A by B in the context L __ R."

Example from Karttunen...

Non-determinism of *replace* (1)

Example:	ab $\rightarrow$ ba $\mid$ x
meaning:	"replace *ab* by *ba* or *x* non-deterministically"
Sample input:	$\underline{\text{a b}}$ c d b $\underline{\text{a b}}$ a
Outputs:	$\underline{\text{b a}}$ c d b $\underline{\text{b a}}$ a
	$\underline{\text{b a}}$ c d b $\underline{\text{x}}$ a
	$\underline{\text{x}}$ c d b $\underline{\text{b a}}$ a
	$\underline{\text{x}}$ c d b $\underline{\text{x}}$ a

Non-determinism of *replace* (2)

Example: [a b | b | b a | a b a] $\rightarrow$ x

meaning: “replace *ab* or *b* or *ba* or *aba* by *x*”

Sample input:	$\underline{\text{a b}}$ a	a $\underline{\text{b}}$ a	a $\underline{\text{b a}}$	$\underline{\text{a b a}}$
Outputs:	x a	a x a	a x	x

Longest match, left-to-right replace

- For many applications, it is useful to define another version of replacement that in all such cases yields a unique outcome.
- The longest-match, left-to-right replace operator, $@\rightarrow$, defined in Karttunen (1996), imposes a unique factorization on every input.
- The replacement sites are selected from left to right, not allowing any overlaps.
- If there are alternate candidate strings starting at the same location, only the longest one is replaced.

A Grammar for Date Expressions

1To9	=	[1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9]
0To9	=	[%0 | 1To9]
SP	=	[", "]
Day	=	[Monday | ... | Saturday | Sunday]
Month	=	[January | ... | November | December]
Date	=	[1To9 | [1 | 2] 0To9 | 3 [%0 | 1]]
Year	=	1To9 (0To9 (0To9 (0To9)))
DateExp	=	Day | (Day SP) Month Date (SP Year)

Marking Date Expressions

- A parser for date expressions can be compiled from the following simple regular expression:
  DateExp @$\rightarrow$ %[... %]
- The above expression can be compiled into a finite-state transducer.
- @$\rightarrow$ is a replacement operator which scans the input from left to right and follows a longest-match.
- Due to the longest match constraint, the transducer brackets only the maximal date expressions.
- The dots mean: identity with the upper string. The whole expression means: replace DateExp by DateExp surrounded by brackets.

Overgeneration Problem

- The grammar for date expressions accepts illegal dates.
- Example: It admits dates like “February 30, 2007”.
- More generally:
  - If a grammar admits strings that should not be accepted by the grammar, the grammar is said to *overgenerate*.
  - If a grammar does not admit strings that should be accepted by the grammar, the grammar is said to *undergenerate*.

Tokenizing Date Expressions

Example:

Today is [Wednesday, August 28, 1996] because yesterday was [Tuesday] and it was [August 27] so tomorrow must be [Thursday, August 29] and not [August 30, 1996] as it says on the program.

Incremental Tokenization

input layer	one, two, and so on.
single word layer	one ‖ , ‖ two ‖ , ‖ and ‖ so ‖ on ‖ . ‖
multi-word layer	one ‖ , ‖ two ‖ , ‖ and so on ‖ . ‖

Advantages of Incremental Tokenization

- With finite-state transducers incremental tokenization is implemented by the composition operator for transducers.
- Separation of grammar specification and program code: Each analysis level is specified in a well-defined language of regular expressions.
- Transducers for each layer can be stated independently of each other.
- Regular expressions can be compiled automatically into (composed) finite state transducers.

A Quick Guide to Morphology (1)

- Morphology studies the internal structure of words.
- The building blocks are called morphemes. One distinguishes between free and bound morphemes.
  - Free morphemes are those which can stand alone as words.
  - Bound morphemes are those that always have to attach to other morphemes.

A Quick Guide to Morphology (2)

Linguists commonly distinguish three types of morphological processes:

- Inflectional morphology: refers to the class of bound morphemes that do not change word class.
- Derivational morphology: refers to the class of bound morphemes that do change word class.
- Compounding: a morphologically complex word can be constructed out of two or more free morphemes.

Inflectional Morphemes

- Bound morphemes which do not change part of speech, e.g. *big* and *bigger* are both adjectives.
- Typically indicate syntactic or semantic relations between different words in a sentence, e.g. the English present tense morpheme *-s* in *waits* shows agreement with the subject of the verb.
- Typically occur with all members of some large class of morphemes, e.g. the pural morpheme *-s* occurs with most nouns.
- Typically occur at the margins of words as affixes (prefix, suffix, circumfix)

Derivational Morphemes

- Bound morphemes which change part of speech, e.g. *-ment* forms nouns, such as *judgment*, from verbs such as *judge*.
- Typically indicate semantic relations within the word, e.g. the morpheme *-ful* in *painful* has no particular connection with any other morpheme beyond the word *painful*.
- Typically occur with only some members of a class of morphemes, e.g. the suffix *-hood* occurs with just a few nouns such as *brother*, *neighbor*, and *knight*, but not with many others, e.g. *friend*, *daughter*, *candle*, etc.
- Typically occur before inflectional suffixes, e.g. in *interpretierbare (Antwort)* the derivational suffix *bar* before the inflectional suffix *-e*.

Compounding

- A compound is a word formed by the combination of two independent words.
- The parts of the compound can be free morphemes, derived words, or other compounds in nearly any combination:
  - *girlfriend* (two independent morphemes),
  - *looking glass* (derived word + free morpheme),
  - *life insurance salesman* (compound + free morpheme).

FYI: Change in Syllabus

Jan. 23	Morphological Analysis	[Trost 2003]
Jan. 30	Part of Speech Tagging	[Leech 1997]
Feb. 6	Final exam	
Feb. 13	Part of Speech Tagging contd. Resources in Computational Linguistics	