Introduction to Computational Linguistics

Finite State Automata and Regular Expressions

Jan-Philipp Söhn

jp.soehn@uni-tuebingen.de

January 9th, 2008

Incremental Linguistic Analysis

- tokenization
- morphological analysis (lemmatization)
- part-of-speech tagging
- named-entity recognition
- partial chunk parsing
- full syntactic parsing
- semantic and discourse processing

Form of Grammars of Type 0–3

For $i \in \{0, 1, 2, 3\}$, a grammar $\langle N, T, P, S \rangle$ of Type i, with N the set of non-terminal symbols, T the set of terminal symbols (N and T disjoint, $\Sigma = N \cup T$), P the set of productions, and S the start symbol ($S \in N$), obeys the following restrictions:

T3: Every production in P is of the form $A \longrightarrow aB$ or $A \longrightarrow \epsilon$, with $B, A \in N$, $a \in T$.

T2: Every production in P is of the form $A \longrightarrow x$, with $A \in N$ and $x \in \Sigma^*$.

T1: Every production in P is of the form $x_1 A x_2 \longrightarrow x_1 y x_2$, with $x_1, x_2 \in \Sigma^*$, $y \in \Sigma^+$, $A \in N$ and the possible exception of $C \longrightarrow \epsilon$ in case C does not occur on the righthand side of a rule in P.

T0: No restrictions.

An Example of a Type 2 Grammar

Let $\langle N, T, P, S \rangle$ be a grammar with N, T and P as given below:

- $N = \{S, NP, VP, N, V\}$
- $T = \{\text{Gravity}, \text{sucks}\}$
- $P = \{S \rightarrow NP\ VP, NP \rightarrow N, VP \rightarrow V, N \rightarrow \text{Gravity}, V \rightarrow \text{sucks}\}$

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

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

Kleene star is a unary operation, either on sets of strings or on sets of symbols or characters.

1. If V is a set of strings then V* is defined as the smallest superset of V that contains 0 (the empty string) and is closed under the string concatenation operation. This set can also be described as the set of strings that can be made by concatenating zero or more strings from V.
2. If V is a set of symbols or characters then V* is the set of all strings over symbols in V, including the empty string.

Meaning of Regular Expressions

$\mathsf{L}(\emptyset) = \emptyset$	the empty language
$\mathsf{L}(0) = \{0\}$	the empty-string language
$\mathsf{L}(\sigma) = \{\sigma\}$	
$\mathsf{L}([\alpha \mid \beta]) = \mathsf{L}(\alpha) \cup \mathsf{L}(\beta)$	
$\mathsf{L}([\alpha\ \beta]) = \mathsf{L}(\alpha) \circ \mathsf{L}(\beta)$	
$\mathsf{L}([\alpha^*]) = (\mathsf{L}(\alpha))^*$	

Σ^* is called the universal language. Note that the universal language is given relative to a particular alphabet.

Remarks on Regular Expressions

- $\emptyset^* =_{def} \{0\}$
- The empty string, i.e., the string containing no character, is denoted by 0. The empty string is the neutral element for the concatenation operation. That is:

$$\texttt{for any string } w \in \Sigma^* : \quad w0 = 0w = w$$

- Square brackets, [], are used for grouping expressions. Thus [A] is equivalent to A while (A) is not.
  We leave out brackets for readability if no confusion can arise.

Regular Expressions: Syntax

- ( ) is (sometimes) used for optionality; e.g. (A) ; definable in terms of union with the empty string.
- ? denotes any symbol; L(?) = Σ
  (our ? corresponds to # elsewhere)
- A^+ denotes iteration; one or more concatenations of A. Equivalent to A (A*).
- Note the following simple expressions:
  - [ ] denotes the empty-string language
  - ?* denotes the universal language (= Σ*)

Deterministic Finite-State Automata

Definition (DFA)

A deterministic FSA (DFA) is a quintuple $(\Sigma, Q, i, F, \delta)$ where

Σ is a finite set called *the alphabet*,

Q is a finite set of *states*,

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

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

δ is the transition function from $Q \times \Sigma$ to Q.

Generalizing Finite-State Automata

Definition (rNFA)

A restricted nondeterministic finite-state automaton is a quintuple $(\Sigma, Q, i, F, \Delta)$ where

Σ is a finite set called *the alphabet*,

Q is a finite set of *states*,

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

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

$\Delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$ is the set of edges (the transition *relation*).

Nondeterministic Finite-State Automata

Definition (NFA)

A nondeterministic finite-state automaton is a quintuple $(\Sigma, Q, S, F, \Delta)$ where

Σ is a finite set called *the alphabet*,

Q is a finite set of *states*,

$S \subseteq Q$ is the set of *initial states*,

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

$\Delta \subseteq Q \times \Sigma^* \times Q$ is the set of edges (the transition *relation*).

Some Important Properties of FSAs (1)

- Determinization: For every nondeterministic finite-state automaton there exists an equivalent deterministic automaton.
- Minimization: For every nondeterministic finite-state automaton there exists an equivalent deterministic automaton with a minimal number of states.

What is in a State

Definition (State)

DFA M $= (\Sigma, Q, i, F, \delta)$,

a *state of M* is a triple (x, q, y)

where $q \in Q$ and $x, y \in \Sigma^*$

The *directly derives* relation

Definition (The *directly derives* relation)

Given a DFA $(\Sigma, Q, i, F, \delta)$,

a state (x, q, y) *directly derives* state (x', q', y'):

$(x, q, y) \vdash (x', q', y')$ iff

1. there is $\sigma \in \Sigma$ such that y $= \sigma$y' and x'$=$ xσ (i.e. the reading head moves right one symbol σ)
2. $\delta(q, \sigma) = q'$

The *derives* relation

Definition (The *derives* relation)

Given a DFA $(\Sigma, Q, i, F, \delta)$,

a state A *derives* state B:

$(x, q, y) \vdash^* (x', q', y')$ iff

there is a sequence $S_0 \vdash S_1 \vdash \cdots \vdash S_k$

such that $A = S_0$ and $B = S_k$

Acceptance

Definition (Acceptance)

Given a DFA $M = (\Sigma, Q, i, F, \delta)$ and a string $x \in \Sigma^*$,

M accepts x iff

there is a $q \in F$ such that $(0, i, x) \vdash^* (x, q, 0)$.

Language accepted by M

Definition (Language accepted by M)

Given a DFA $M = (\Sigma, Q, i, F, \delta)$, the language $L(M)$ accepted by M is the set of all strings accepted by M.

Example of a Regular Expression

$$a^+b^+c$$

We illustrate RegExes, FSAs, Transducers etc. with JFLAP (`http://www.jflap.org/`)

Alternatively, on the SfS system there are the FSA Utilities (`http://www.let.rug.nl/~vannoord/Fsa/`)

Example of String Acceptance

Let $M = (\{a, b\}, \{q_0, q_1, q_2\}, q_0, \{q_1\}, \{((q_0, a), q_1), ((q_0, b), q_1),$
$((q_1, a), q_2), ((q_1, b), q_2), ((q_2, a), q_2), ((q_2, b), q_2)\})$.

Example of String Acceptance

Let $M = (\{a, b\}, \{q_0, q_1, q_2\}, q_0, \{q_1\}, \{((q_0, a), q_1), ((q_0, b), q_1), ((q_1, a), q_2), ((q_1, b), q_2), ((q_2, a), q_2), ((q_2, b), q_2)\})$.

M accepts a and b and nothing else, i.e. $L(M) = \{a, b\}$, since

$(0, q_0, a) \vdash (a, q_1, 0)$ and
$(0, q_0, b) \vdash (b, q_1, 0)$

are the only derivations from a start state to a final state for M.