Mathematics for linguists

Gerhard Jäger
gerhard.jaeger@uni-tuebingen.de

Uni Tübingen, WS 2009/2010

November 24, 2009

Regular expressions

- fourth kind (next to type-3 grammars, deterministic and non-deterministic finite automata) to describe regular languages
- very useful for search in texts
- important technique in corpus studies
- many software packages include implementations of regular expressions, for instance
  - Emacs
  - Word
  - OpenOffice
  - Perl
  - Python
  - Unix-Tools wie grep/egrep oder sed
- specific syntax might differ slightly, but the underlying concepts are identical

Regular expressions

Definition (Syntax of regular expressions)

Let Σ be a finite alphabet.

- $\emptyset$ is a regular expression.
- ϵ is a regular expression.
- For each $a \in \Sigma$: a is a regular expression.
- If α and β are regular expressions, then
  - $\alpha\beta$,
  - $(\alpha \cup \beta)$ (sometimes written as $\alpha|\beta$), and
  - α^*

  are regular expressions.

In concrete implementations, the syntax is usually extended by expressions for word boundaries and line boundaries, finite classes of single symbols, finite iterations etc.

Regular expressions

Regular expressions are interpreted recursively as formal languages over Σ^*. For this, two operations over formal languages have to be defined, **concatenation** and **iteration**.

The concatenation of two formal languages

Definition

Let L_1 and L_2 be two formal languages. The *concatenation* $L_1 \cdot L_2$ of L_1 and L_2 is defined as

$$L_1 \cdot L_2 = \{\vec{x} \cdot \vec{y} | \vec{x} \in L_1, \vec{y} \in L_2\}$$

The concatenation of two formal languages

Example:

- $L_1 = \{a^n b^n | n > 1\}$
- $L_2 = \{c^m | m \geq 0\}$
- $L_1 \cdot L_2$
$$\begin{aligned} &= \{aabb, aabbc, aabbcc, aabbccc, aabbcccc, aaabbbc, ...\} \\ &= \{a^n b^n c^m | n > 1, m \geq 0\} \end{aligned}$$
- Notational convention:

$$\begin{aligned} L^0 &= \{\epsilon\} \\ L^1 &= L \\ L^2 &= L \cdot L \\ L^{n+1} &= L^n \cdot L \end{aligned}$$

Iteration

Definition

Let L be a formal language. The *iteration* of L is defined as
$$L^* = \{\ \vec{x} \mid \text{there is an } n \in \mathbb{N}, \text{ such that } \vec{x} = \vec{x}_1 \cdot \vec{x}_2 \cdot \dots \cdot \vec{x}_n \text{ and } \vec{x}_i \in L \text{ für } 1 \leq i \leq n\ \}$$

- Note that the empty string ϵ is also an element of L^*, for arbitrary L. (n ist in dem Fall gleich 0.)
- L^* can also be defined as

$$L^* = L^0 \cup L^1 \cup L^2 \cup \cdots$$

Regular expressions

The function $L(\cdot)$ assigns a formal language to each regular expression.

Definition

$$
\begin{aligned}
L(\emptyset) &= \emptyset \\
L(\epsilon) &= \{\epsilon\} \\
L(a) &= \{a\} \text{ (wenn } a \in \Sigma) \\
L(\alpha\beta) &= L(\alpha) \cdot L(\beta) \\
L((\alpha \cup \beta)) &= L(\alpha) \cup L(\beta) \\
L(\alpha^*) &= L(\alpha)^*
\end{aligned}
$$

Regular expressions, type-3 grammars and finite automata

Three kinds of operations over formal languages can be expressed using regular expression,

- union,
- concatenation, and
- iteration.

The class of regular languages is closed under these operations.

Union of regular languages

Theorem

If L_1 and L_2 are regular languages, then $L_1 \cup L_2$ is also a regular language.

Union of regular languages

Idea of proof:

If L_1 is a regular language, there is a type-3 grammar $G_1 = \langle V_{T,1}, V_{N,1}, S_1, R_1 \rangle$ that generates L_1. (Without restriction of generality, we asume that $V_{N,1} \cap V_{N,2} = \emptyset$.)
Likewise, there is a type-3 grammar $G_2 = \langle V_{T,2}, V_{N,2}, S_2, R_2 \rangle$, that generates L_2. We construct a new grammar $G = \langle V_T, V_N, S, R \rangle$ (with $S \notin V_{N,1} \cup V_{N,2}$) that generates $L_1 \cup L_2$:

- $V_T = V_{T,1} \cup V_{T,2}$
- $V_N = V_{N,1} \cup V_{N,2} \cup \{S\}$
- $$\begin{aligned} R &= R_1 \cup R_2 \\ &\cup \{S \rightarrow \alpha | S_1 \rightarrow \alpha \in R_1\} \\ &\cup \{S \rightarrow \alpha | S_2 \rightarrow \alpha \in R_2\} \end{aligned}$$

Concatenation of regular languages

Theorem

If L_1 and L_2 are regular languages, then $L_1 \cdot L_2$ is also a regular language.

Concatenation of regular languages

Idea of proof:
If L_1 is a regular language, then there is a type-3 grammar $G_1 = \langle V_{T,1}, V_{N,1}, S_1, R_1 \rangle$ that generates L_1. (Without restriction of generality, we assume that $V_{N,1} \cap V_{N,2} = \emptyset$.)
Likewise, there is a type-3 grammar $G_2 = \langle V_{T,2}, V_{N,2}, S_2, R_2 \rangle$ that generates L_2. We construct a new grammar $G = \langle V_N, V_T, S_1, R \rangle$ that generates $L_1 \cdot L_2$:

- $V_T = V_{T,1} \cup V_{T,2}$
- $V_N = V_{N,1} \cup V_{N,2} \cup \{S\}$
- $R = R_2 \cup \{A \to xS_2 | A \to x \in R_1\}$

Iteration of regular languages

Theorem

If L is a regular language, then L^* is also a regular language.

Iteration of regular languages

Idea of proof:
If L is a regular language, then there is a type-3 grammar $G = \langle V_T, V_N, S, R \rangle$ that generates L.
We construct a new grammar $G' = \langle V_N, V_T, S, R' \rangle$ that generates L^*:

$$R' = R \cup \{A \to xS | A \to x \in R\}$$

Finite languages are regular

Theorem

Every finite language is a regular language.

Idea of proof:
We construct a type-3 grammar that generates L as follows:

$$R = \{S \rightarrow \vec{x} | \vec{x} \in L\}$$

Regular languages and regular expressions

Theorem

If α is a regular expression, then $L(\alpha)$ is a regular language.

Idea of proof:
If $\alpha = \epsilon$, $\alpha = \{\epsilon\}$ or $\alpha = \{a\}$ for some $a \in \Sigma$, then $L(\alpha)$ is finite — and therefore also regular. Furthermore, it follows from the previous theorem:

- If $L(\alpha)$ and $L(\beta)$ are regular, then $L((\alpha \cup \beta)) = L(\alpha) \cup L(\beta)$ and $L(\alpha\beta) = L(\alpha) \cdot L(\beta)$ are also regular.
- If $L(\alpha)$ is regular, then $L(\alpha^*) = L(\alpha)^*$ is also regular.

Therefore it generally holds: If α does not contain any occurrences of concatenation, union or iteration, then $L(\alpha)$ is regular. It also holds: if $L(\alpha)$ is regular for all regular expressions α that contain at most n occurrences of concatenation, union or iteration, then $L(\alpha)$ is also regular for all α that contain $n + 1$ occurrences of these operations. The theorem hence follows via complete induction.

Regular languages and regular expressions

Theorem

If L is a regular language, then there is a regular expression α such that $L(\alpha) = L$.

The proof for this theorem is too complex to be discussed in this course. It is based on a construction that transforms a DFA into an equivalent regular expression.

Regular expressions, grammars and automata

Theorem

Regular expressions, type-3 grammars, deterministic finite automata and non-deterministic finite automata all describe the same class of languages.