Mathematics for linguists

Gerhard Jäger
gerhard.jaeger@uni-tuebingen.de

Uni Tübingen, WS 2009/2010

November 10, 2009

Theory of formal languages

Formal language:

- set of strings of symbols
- formal languages (for the time being) only model the form aspect of natural languages
- basic assumption: any string of symbols either belongs or does not belong to a given language $\Rightarrow$ idealization
- all interesting formal langauges are infinite (i.e. infinte sets of finite strings)
- formal grammar: finite description of a formal language
- (language) automata: abstract machines (computer programs) that are able to decide wehther or not a string belongs to a given formal language

Foundations

- Let a **finite** set A of symbols (called the *alphabet* or the *vocabulary*) be given
- (symbol) string over A: finite sequence of elements of A
- example:
    - $A = \{a, b, c\}$ (for instance $\{\texttt{Peter}, \texttt{Mary}, \texttt{sees}\}$)
    - strings over A:
        - $\vec{x} := abc$ (`Peter Mary sees`)
        - $\vec{y} := acbbca$ (`Peter sees Mary Mary sees Peter`)
        - $\vec{z} := bacbbca$ (`Mary Peter sees Mary Mary sees Peter`)
- *length* of a string: number of symbols that occur in the string (if the same symbol occurs more than once, it is counted more than once)
    - $l(\vec{x}) = 3$
    - $l(\vec{y}) = 6$
    - $l(\vec{z}) = 7$

Foundations

- A string of length n over the vocabulary A can be modeled set theoretically as
  - a function from $\{0, 1, \ldots, n-1\}$ to A
  - '`Peter sees Mary Mary sees Peter`' comes out as the function

    $f : \{0, 1, 2, 3, 4, 5\} \rightarrow \{\texttt{Peter}, \texttt{Mary}, \texttt{sees}\}$ with

    $$\begin{array}{lll} 0 & \mapsto & \texttt{Peter} \\ 1 & \mapsto & \texttt{sees} \\ 2 & \mapsto & \texttt{Mary} \\ 3 & \mapsto & \texttt{Mary} \\ 4 & \mapsto & \texttt{sees} \\ 5 & \mapsto & \texttt{Peter} \end{array}$$

    or, equivalently

    $$\begin{array}{lll} f(0) & = & \texttt{Peter} \\ f(1) & = & \texttt{sees} \\ f(2) & = & \texttt{Mary} \\ f(3) & = & \texttt{Mary} \\ f(4) & = & \texttt{sees} \\ f(5) & = & \texttt{Peter} \end{array}$$

Foundations

- A string of length n over the vocabulary A can be modeled set theoretically as
    - a function from $\{0, 1, \ldots, n-1\}$ to A
- Important: there is a difference between an element $a \in A$ and the string a of length 1, which only consists of the symbol a. The latter is, strictly speaking, the function $f : \{0\} \rightarrow A$ with $f(0) = a$.
- There is exactly one string of length 0, the **empty string**. It is written as ϵ. Technically, it is the (empty) mapping $\epsilon : \{\ \} \rightarrow A$ (for any arbitrary alphabet A). (sometimes written as e or as $\langle\rangle$, since it can be considered a 0-tuple).
- The set of all finite strings over A (including the empty string) is written as A^*.

Concatenation

- most important operation over strings: *concatenation* (dt. *Verkettung*), written as "$\cdot$" (or "$\frown$")
- juxtaposition of two strings:
  - $abc \cdot abc = abcabc$
  - $daaac \cdot \epsilon = daaac$
  - $\epsilon \cdot cabbba = cabbba$
- associative: for arbitrary strings $\vec{u}, \vec{v}, \vec{w} \in A^*$:

$$(\vec{u} \cdot \vec{v}) \cdot \vec{w} = \vec{u} \cdot (\vec{v} \cdot \vec{w})$$

- ϵ is a **neutral element** for concatenation:

$$\epsilon \cdot \vec{u} = \vec{u} = \vec{u} \cdot \epsilon$$

Reversal of a string

- Notation: If $\vec{u}$ is a string, $\vec{u}^R$ is the reversal of this string.
- for instance: $(acbab)^R = babca$
- for the empty string, we have: $\epsilon^R = \epsilon$
- recursive definition:

Definition

Let A be an alphabet.

1. If $\vec{v}$ is a string of length 0 (i.e. $\vec{v} = \epsilon$), then $\vec{v}^R = \vec{v}$.
2. If $\vec{v}$ is a string of length $n + 1$, then it can be written as $\vec{w}a$ (with $\vec{w} \in A^*$ and $a \in A$). It holds that: $(\vec{w}a)^R = a\vec{w}^R$.

Foundations

- Connection between concatenation and reversal:

$$(\vec{u} \cdot \vec{v})^R = \vec{v}^R \cdot \vec{u}^R$$

- **substring**: $\vec{v}$ is a *substring* of $\vec{u} \in A^*$ iff there are $\vec{z}, \vec{w} \in A^*$ such that $\vec{u} = \vec{z} \cdot \vec{v} \cdot \vec{w}$.
- If $\vec{v}$ is a substring of $\vec{u}$ and $l(\vec{v}) < l(\vec{u})$, then $\vec{v}$ is a **proper substring** of $\vec{u}$.
- **prefix:** $\vec{v}$ is a *prefix* of $\vec{u} \in A^*$ iff ther is some $\vec{w} \in A^*$ such that $\vec{u} = \vec{v} \cdot \vec{w}$.
- **Suffix:** $\vec{v}$ ist ein *Suffix* von $\vec{u} \in A^*$ gdw. es ein $\vec{w} \in A^*$ gibt so dass $\vec{u} = \vec{w} \cdot \vec{v}$.

Languages

Formal languages

A (formal) **Language** over an alphabet A is a subset of A^*, i.e. a set of strings over A.

- Languages can be finite or infinite.
- As linguists, we are mainly interested in infinite languages.
- Not all languages have a finite description.
- Humboldt: (Natural) languages make "infinite use of finite means" $\Rightarrow$ natural languages are infinite, but they have finite descriptions (grammars)

Languages

Examples for formal languages

- $L = \{\vec{x} \in \{a, b\}^* | \vec{x}$ contains the same number of a and b (in any order)$\}$
- $L_1 = \{\vec{x} \in \{a, b\}^* | \vec{x} = a^n b^n, n \geq 0$ (i.e. a string of n times a, followed by an equal number of b) $\}$
- $L_2 = \{\vec{x} \in \{a, b\}^* | \vec{x}$ contains n times b and n^2 times a, for $n \in \mathbb{N}\}$

Grammars

(Formal) Grammars are precise descriptions of formal languages. A grammar consists of

- two alphabets, the **terminal alphabet** V_T and the **Non-terminal alphabet** V_N,
- a **start symbol** S, and
- a set of **(replacement) rules**. A replacement rule consists of two parts, the **left hand side** and the **right hand side**.

We obtain a **derivation** for a grammar by starting with the string S, and successively replacing substrings with match with the right hand side of a rule by the left hand side of the same rule.

Examples

$$
\begin{aligned}
V_T \text{ (terminal alphabet)} &= \{a, b\} \\
V_N \text{ (non-terminal alphabet)} &= \{S, A, B\} \\
S \text{ (start symbol)} & \\
R \text{ (rules)} &= \left\{ \begin{array}{lcl} S & \rightarrow & ABS \\ S & \rightarrow & \epsilon \\ AB & \rightarrow & BA \\ BA & \rightarrow & AB \\ A & \rightarrow & a \\ B & \rightarrow & b \end{array} \right\}
\end{aligned}
$$

Grammars

- Convention: terminal symbols are written as lower case letters and non-terminal symbols as upper case letters.
- **Derivation** for the grammar from the previous slide:

  $S \Rightarrow ABS \Rightarrow ABABS \Rightarrow ABAB \Rightarrow ABBA \Rightarrow ABbA \Rightarrow aBbA \Rightarrow abbA \Rightarrow abba$

- We cannot apply any replacement rules to $abba$ anymore, because it consists exclusively of terminal symboles. Such a string is called **terminal string**.
- The language that is **generated** by a grammar is defined as the set of all terminal strings that can be derived from the start symbol via (repeated) applications of the replacement rules.

Grammars

Definition ((Formal) Grammar)

A (formal) **grammar** is a 4-tuple $\langle V_T, V_N, S, R\rangle$, where V_T and V_N are finite, mutually disjoint sets (i.e. $V_T \cap V_N = \emptyset$), $S \in V_N$, and $R \subseteq (V_T \cup V_N)^* \times (V_T \cup V_N)^*$. Furthermore, the left hand side of each rule contains at least one element of V_N.

We usually write rules as $\alpha \to \beta$ rather than $\langle \alpha, \beta \rangle$.

Definition (Derivation)

Let $G = \langle V_T, V_N, S, R\rangle$ be a grammar. A **derivation** for G is a sequence of strings $\vec{x}_0, \vec{x}_1, \ldots, \vec{x}_n (n \geq 0)$, such that for every $\vec{x}_i$ with $0 \leq i < n$ it holds that

- $\vec{x}_i = \vec{u} \cdot \vec{v} \cdot \vec{w}$,
- there is a rule $\vec{v} \rightarrow \vec{z} \in R$, and
- $\vec{x}_{i+1} = \vec{u} \cdot \vec{z} \cdot \vec{w}$.

Definition (Generation)

A grammar G **generates** a string $\vec{x} \in V_T^*$ if and only if there is a derivation $\vec{x}_0, \ldots, \vec{x}_n$ for G such that $\vec{x}_0 = S$ and $\vec{x}_n = \vec{x}$.

Definition (Generated language)

The language that is **generated by** a grammar G (written as $L(G)$) is the set of all strings that are generated by G.