Formal Languages – Basic Terminology

Peter Bauer

Version 01.02.02

Outline

Alphabet

Definition

An *Alphabet* (sometimes called a *Vocabulary*) is a non-empty and finite set of elements.

Remark

Alphabets are often denoted by upper-case letters A or V and sometimes as upper-case greek letters like Σ.

Alphabet

Definition

An *Alphabet* (sometimes called a *Vocabulary*) is a non-empty and finite set of elements.

Remark

Alphabets are often denoted by upper-case letters A or V and sometimes as upper-case greek letters like Σ.

Example

- $\{0, 1\}$: Binary alphabet.
- ASCII: Machine text alphabet.
- $\{a, b\}$: Small alphabet but enough for many examples.

String

Definition

A *String* is a finite sequence of symbols of an alphabet.

String

Definition

A *String* is a finite sequence of symbols of an alphabet.

Remark

- Strings are often denoted by lower greek letters like $\alpha, \beta, \varphi, \omega, \ldots$
- A string over the alphabet Σ means a string all of whose symbols are in Σ.

String

Definition
A *String* is a finite sequence of symbols of an alphabet.

Remark
- Strings are often denoted by lower greek letters like $\alpha, \beta, \varphi, \omega, \ldots$
- A string over the alphabet Σ means a string all of whose symbols are in Σ.

Definition
The *length* of a string ω denoted as $|\,\omega\,|$ is the number of symbols in the string.

String

Definition

A *String* is a finite sequence of symbols of an alphabet.

Remark

- Strings are often denoted by lower greek letters like $\alpha, \beta, \varphi, \omega, \ldots$
- A string over the alphabet Σ means a string all of whose symbols are in Σ.

Definition

The *length* of a string ω denoted as $|\omega|$ is the number of symbols in the string.

Definition

A string ω is called *empty* if $|\omega| = 0$. It is often denoted as ε or λ.

Concatenation

Definition

Let ω_1 and ω_2 be two strings. The *concatenation* (sometimes also called *catenation*) of ω_1 and ω_2 makes a new string α containing all the symbols of ω_1 in order followed by all symbols of ω_2 in order. This is usually written as $\alpha = \omega_1\omega_2$.

Concatenation

Definition

Let ω_1 and ω_2 be two strings. The *concatenation* (sometimes also called *catenation*) of ω_1 and ω_2 makes a new string α containing all the symbols of ω_1 in order followed by all symbols of ω_2 in order. This is usually written as $\alpha = \omega_1\omega_2$.

Example

Given two strings $\omega_1 = \texttt{abc}$ and $\omega_2 = \texttt{cde}$ then the concatenation α of ω_1 and ω_2 is $\alpha = \omega_1\omega_2 = \texttt{abccde}$.

Concatenation

Definition

Let ω_1 and ω_2 be two strings. The *concatenation* (sometimes also called *catenation*) of ω_1 and ω_2 makes a new string α containing all the symbols of ω_1 in order followed by all symbols of ω_2 in order. This is usually written as $\alpha = \omega_1\omega_2$.

Example

Given two strings $\omega_1 = \mathtt{abc}$ and $\omega_2 = \mathtt{cde}$ then the concatenation α of ω_1 and ω_2 is $\alpha = \omega_1\omega_2 = \mathtt{abccde}$.

Remark

For any string ω the relation $\varepsilon\omega = \omega\varepsilon = \omega$ holds.

Production Rule

Definition

A *production rule* (sometimes called a *re-writing rule* or simply *rule*) is a tuple (A, α), where A is a symbol and α is a string of symbols.

Production Rule

Definition

A *production rule* (sometimes called a *re-writing rule* or simply *rule*) is a tuple (A, α), where A is a symbol and α is a string of symbols.

Remark

A rule is often denoted as $A \to \alpha$ and can be understood as "A can be re-written (or substituted) by α" or "A is defined as α".

Example

- F $\to$I
- I $\to$a

Grammar

Definition

A *grammar* $G(S)$ is a finite, non-empty set of production rules. S is called the start symbol and appears on at least one left side of the production rules. All symbols on the left and right sides are the Vocabulary.

Grammar

Definition

A *grammar* $G(S)$ is a finite, non-empty set of production rules. S is called the start symbol and appears on at least one left side of the production rules. All symbols on the left and right sides are the Vocabulary.

Example

$$\begin{array}{lcl} \mathrm{F} & \rightarrow & \mathrm{I} \\ \mathrm{F} & \rightarrow & \neg\mathrm{F} \\ \mathrm{F} & \rightarrow & (\mathrm{F} \wedge \mathrm{F}) \\ \mathrm{F} & \rightarrow & (\mathrm{F} \vee \mathrm{F}) \\ \mathrm{I} & \rightarrow & \mathrm{a} \\ \mathrm{I} & \rightarrow & \mathrm{b} \\ \mathrm{I} & \rightarrow & \mathrm{c} \end{array}$$

Terminal and Non-Terminal Symbols

Definition

All symbols appearing on the left side of a grammar $G(S)$ are called *non-terminals*. The set of all non-terminals of $G(S)$ is denoted by V_N. All other symbols are called *terminals*. The set of these symbols is denoted by V_T. Obviously it holds $V = V_N \cup V_T$.

Denoting Grammars — Formal Languages

- Terminals in lower-case letters
- Non-terminals in upper-case letters
- Separator: $\rightarrow$
- Alternatives: $|$

Denoting Grammars — Formal Languages

- Terminals in lower-case letters
- Non-terminals in upper-case letters
- Separator: $\rightarrow$
- Alternatives: $|$

Example

The language of the propositional logic is defined as follows using the classical notation system of formal languages.

$$
\begin{array}{lcl}
\mathsf{F} & \rightarrow & \mathsf{I}\ |\neg \mathsf{F}\ |(\mathsf{F} \wedge \mathsf{F})\ |(\mathsf{F} \vee \mathsf{F}) \\
\mathsf{I} & \rightarrow & \mathsf{a}\ |\mathsf{b}\ |\mathsf{c}
\end{array}
$$

Denoting Grammars — EBNF

- Terminals under double quotes
- Non-terminals in meaningful words written in camel case
- Separator: $=$
- Alternatives: $|$
- Each rule ends with a period.
- Options: $[A]$ means A or ε.
- Repetition: $\{A\}$ means ε or A or AA or AAA ...
- Parentheses for grouping.

Denoting Grammars — EBNF

- Terminals under double quotes
- Non-terminals in meaningful words written in camel case
- Separator: $=$
- Alternatives: $|$
- Each rule ends with a period.
- Options: $[A]$ means A or ε.
- Repetition: $\{A\}$ means ε or A or AA or AAA ...
- Parentheses for grouping.

Example

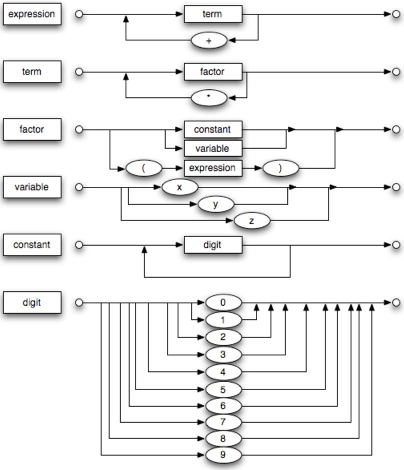
Expressions in the programming language Modula 2 written in EBNF.

Expression	=	["+" |"-"] Term {("+" |"-") Term}.
Term	=	Factor {("*" |"/") Factor}.
Factor	=	c |v |"(" Expression ")".

Denoting Grammars — Syntax Diagrams

- Each diagram defines a non-terminal
- Terminals are represented by round boxes
- Non-terminals are represented by square boxes

Syntax Diagrams – An Example



Syntax Trees or Parse Trees

Show by drawing the parse tree that -5 + 3 * (a - x) is an expression in the sense of the grammar

Expression	=	["+" |"-"] Term {("+" |"-") Term}.
Term	=	Factor {("*" |"/") Factor}.
Factor	=	c |v |"(" Expression ")".

Generating Strings

Definition

Given

1. a formal grammar G with a rule $A \to \varphi$
2. a string $\alpha = \omega_1 A \omega_2$.

Then we can obviously generate a new string $\beta = \omega_1 \varphi \omega_2$. In this case we say α *generates* β *directly*, in symbols $\alpha \Rightarrow \beta$.

Generating Strings

Definition

Given

1. a formal grammar G with a rule $A \to \varphi$
2. a string $\alpha = \omega_1 A \omega_2$.

Then we can obviously generate a new string $\beta = \omega_1 \varphi \omega_2$. In this case we say α *generates* β *directly*, in symbols $\alpha \Rightarrow \beta$.

Definition

A string α *generates* a string β (usually denoted by $\alpha \Rightarrow^+ \beta$ if there exists a sequence of direct generations

$$\alpha = \omega_0 \Rightarrow \omega_1 \Rightarrow \omega_2 \Rightarrow \ldots \Rightarrow \omega_n = \beta. \qquad (n > 0)$$

If $\alpha \Rightarrow^+ \beta$ or $\alpha = \beta$ we write $\alpha \Rightarrow^* \beta$ and say α *generates or is equal* to β.

Kleene Star

Definition

Let Σ be an alphabet. Then we define the sets Σ_i recursively as follows:

$$\begin{aligned}\Sigma_0 &= \{\varepsilon\} \\ \Sigma_{i+1} &= \{\omega\upsilon \mid \omega \in \Sigma_i \wedge \upsilon \in \Sigma\}\end{aligned}$$

The *Kleene star* is defined then by $\Sigma^* = \bigcup_{i \in \mathbb{N}_0} \Sigma_i$

Kleene Star

Definition

Let Σ be an alphabet. Then we define the sets Σ_i recursively as follows:

$$\Sigma_0 = \{\varepsilon\}$$
$$\Sigma_{i+1} = \{\omega\upsilon \mid \omega \in \Sigma_i \wedge \upsilon \in \Sigma\}$$

The *Kleene star* is defined then by $\Sigma^* = \bigcup_{i \in \mathbb{N}_0} \Sigma_i$

Example

Let $V = \{\text{"ab"}, \text{"c"}\}$ be an alphabet. Then the $V^* = \{\varepsilon, \text{"ab"}, \text{"c"}, \text{"abab"}, \text{"abc"}, \text{"cc"}, \text{"cab"}, \text{"ababab"}, \text{"ababc"}, \ldots\}$

Kleene Star

Definition

Let Σ be an alphabet. Then we define the sets Σ_i recursively as follows:

$$\Sigma_0 = \{\varepsilon\}$$
$$\Sigma_{i+1} = \{\omega\upsilon \mid \omega \in \Sigma_i \wedge \upsilon \in \Sigma\}$$

The *Kleene star* is defined then by $\Sigma^* = \bigcup_{i \in \mathbb{N}_0} \Sigma_i$

Example

Let $V = \{\texttt{"ab"}, \texttt{"c"}\}$ be an alphabet. Then the $V^* = \{\varepsilon, \texttt{"ab"}, \texttt{"c"}, \texttt{"abab"}, \texttt{"abc"}, \texttt{"cc"}, \texttt{"cab"}, \texttt{"ababab"}, \texttt{"ababc"}, \ldots\}$

Remark

Loosely interpreted we could say that the Kleene Star of an alphabet is the set of all strings that can be built out of this alphabet.

Language

Definition

Let $G(S)$ be a grammar with a start symbol S. The set

$$L(G(S)) = \{\alpha : S \Rightarrow^* \alpha \wedge \alpha \in V_T^*\}$$

is then called the *language* of $G(S)$.

Language

Definition

Let $G(S)$ be a grammar with a start symbol S. The set

$$L(G(S)) = \{\alpha : S \Rightarrow^* \alpha \wedge \alpha \in V_T^*\}$$

is then called the *language* of $G(S)$.

Example

Let $G(\text{Java})$ be the grammar defining the programming language Java. $L(G(\text{Java}))$ is then

Language

Definition

Let $G(S)$ be a grammar with a start symbol S. The set

$$L(G(S)) = \{\alpha : S \Rightarrow^* \alpha \wedge \alpha \in V_T^*\}$$

is then called the *language* of $G(S)$.

Example

Let $G(\text{Java})$ be the grammar defining the programming language Java. $L(G(\text{Java}))$ is then the set of all syntactically correct Java programs.