

Formal Languages and Compilers

Matteo Secco

March 3, 2021

Contents

1	Formal Language Theory	3
1.1	Operations on strings	3
1.2	Operations on Languages	4
2	Regular Expressions and Languages	6
2.1	Algebraic definition	6
2.2	Language Families	7
2.3	Derivation	7
2.4	Ambiguity of Regular Expressions	7
2.5	Extended Regular Expressions	8
3	Context Free Grammars	9

1 Formal Language Theory

Alphabet Σ : any finite set of symbols $\Sigma = \{a_1, a_2, \dots, a_k\}$

String: a sequence of alphabeth elements

Language: a set (possibly infinite) of strings

$$\Sigma = \{a, b, c\} \quad L_1 = \{ab, ac\} \quad L_2 = \{ab, aab, aaab, aaaab, \dots\}$$

Sentences/Phrases: strings belonging to a language

Language cardinality: number of sentences of the language

$$|L_1| = |\{ab, ab\}| = 2 \quad |L_2| = |\{ab, aab, aaab, aaaab, \dots\}| = \infty$$

Number of occurrences of a symbol in a string: $|bbc|_b = 2, |bbc|_a = 0$

Length of a string: number of its elements

$$|bbc| = 3 \quad |abbc| = 4$$

String equality: two strings $x = a_1a_2\dots a_h$ and $y = a_1a_2\dots a_k$ are equal \iff

- have same length: $|x| = |y| \iff h = k$
- elements from left to right coincide: $a_i = b_i \quad \forall i \in \{1..h\}$

1.1 Operations on strings

Concatenation $x = a_1a_2\dots a_h \wedge y = b_1b_2\dots b_k \implies x \cdot y = a_1a_2\dots a_hb_1b_2\dots b_k$

- associative: $(xy)z = x(yz)$
- length: $|xy| = |x| + |y|$

Empty string ϵ is the neutral element for concatenation: $x\epsilon = \epsilon x = x \forall x$.

- length: $|\epsilon| = 0$
- **NB:** $\epsilon \neq \emptyset$

Substrings: if $x = uyv$ then

- y is a substring of x
- y is a proper substring of $x \iff u \neq \epsilon \vee v \neq \epsilon$
- u is a prefix of x
- v is a suffix of y

Reflection: if $x = a_1a_2\dots a_h$ then $x^R = a_ha_{h-1}\dots a_1$

- $(x^R)^R = x$
- $(xy)^R = y^R x^R$
- $\epsilon^R = \epsilon$

Repetition: $x^m = \underbrace{xxx\dots x}_{m \text{ times}}$. Inductive definition:

- $x^0 = \epsilon$
- $x^m = x^{m-1}x$ if $m > 0$

1.2 Operations on Languages

Reflection: $L^R = \{x | \exists y (y \in L \wedge x = y^R)\}$

Prefixes(L): $\{y | y \neq \epsilon \wedge \exists x \exists z (x \in L \wedge z \neq \epsilon \wedge x = yz)\}$

- **Prefix-free language:** $L \cap \text{Prefixes}(L) = \emptyset$

Concatenation: $L'L'' = \{xy | x \in L' \wedge y \in L''\}$

Power: inductive definition:

- $L^0 = \{\epsilon\}$
- $L^m = L^{m-1}L$ for $m > 0$
- Consequences:
 - $\emptyset^0 = \{\epsilon\}$
 - $L \cdot \emptyset = \emptyset \cdot L = \emptyset$
 - $L \cdot \{\epsilon\} = \{\epsilon\} \cdot L = L$

Universal language: over alphabet Σ : $L_{\text{universal}} = \Sigma^0 \cup \Sigma^1 \cup \dots$

Complement: of L over Σ : $\neg L = L_{\text{universal}} \setminus L$

Star: formally called **reflexive and transitive closure** or **Kleene star**

$$L^* = \bigcup_{h=0}^{\infty} L^h = L^0 \cup L^1 \cup \dots = \epsilon \cup L^1 \cup L^2$$

$$\Sigma^* = L_{\text{universal}}$$

Monotonic: $L \subseteq L^*$

Close under concatenation: $x \in L^* \wedge y \in L^* \implies xy \in L^*$

Idempotent: $(L^*)^* = L^*$

Commutative with reflection: $(L^*)^R = (L^R)^*$

$$\begin{aligned} \emptyset^* &= \{\epsilon\} \\ \{\epsilon\}^* &= \{\epsilon\} \end{aligned}$$

Cross: $L^+ = L \cdot L^*$

Quotient: $L_1/L_2 = \{y | \exists x \in L_1 \exists z \in L_2 (x = yz)\}$

- **Not set quotient!**
- Removes from L_1 suffixes contained in L_2

2 Regular Expressions and Languages

Regular languages are the simplest family of languages.

They can be defined in three ways:

- Algebraically
- Using generative grammars
- Using recognizer automata

2.1 Algebraic definition

Regular expressions are expression on languages that composes languages operations.

Formally

- Is a string r
- Over the alphabet $\Sigma = \{a_1, a_2, \dots, a_n\} \cup \{\emptyset, \cup, \cdot, *\}$

Moreover, assuming s and t are regular expressions, then r is a regular expression if any of the following rules applies:

- $r = \emptyset$
- $r = a, \quad a \in \Sigma$
- $r = s \cup t$ (alternative notation is $s|t$)
- $r = s \cdot t$ (the \cdot can be omitted)
- $r = s^*$

The meaning of a r.e. is a **language** L_r of alphabet Σ according to the table:

Expression	Language
\emptyset	\emptyset
ϵ	$\{\epsilon\}$
$a \in \Sigma$	$\{a\}$
$s \cup t$	$L_s \cup L_t$
$s \cdot t$	$L_s \cdot L_t$
s^*	L_s^*

Regular Languages are languages denoted by a regular expression

2.2 Language Families

REG is the collection of all regular languages

FIN is the collection of all languages with finite cardinality

Every finite language is regular $FIN \subset REG$:

- $L \in FIN \implies L = \bigcup_{i=1}^{k \in \mathbb{N}} x_i \implies L \in FIN$
- $L = a^* \implies L \in REG \wedge L \notin FIN$

2.3 Derivation

Choice Union and Concatenation corresponds to possible choices. One obtains subexpressions by making a choice that identifies a sub language.

Regular expression	Choices
$e_1 \cup \dots \cup e_k$	$e_i \quad \forall i \in \{1, 2, \dots, k\}$
e^*	$\epsilon \text{ or } e^n \quad \forall n \geq 1$
e^+	$e^n \quad \forall n \geq 1$

Derivation among two r.e: $e_1 \implies e_2$ if

$$e_1 = \alpha\beta\gamma \wedge e_2 = \alpha\delta\gamma$$

where γ is a choice of β .

Derivation can be applied repeatedly, leading to $\overset{n}{\implies}$ (deriving n times, $\overset{*}{\implies}$ (0 or more times), $\overset{+}{\implies}$ (1 or more times).

Language defined by an r.e. $L(r) = \{x \in \Sigma^* | r \overset{*}{\implies} x\}$

Equivalent r.e. defines the same language

2.4 Ambiguity of Regular Expressions

Numbered subexpressions of a R.E

- Add all possible parentheses to the r.e.
- number the elements of Σ
- identify all the subexpressions

Ambiguity happens when a phrase can be obtained through distinct derivations, which differ **not only for the order**.

Sufficient condition for ambiguity of the r.e. f having numbered version f' is that $\exists x \exists y \in L(f') | x \neq y$ but $x = y$ when numbers are removed

2.5 Extended Regular Expressions

Regular expressions extended with other operators:

Power: $a^n = \underbrace{aa...a}_n$. NB: n is an actual number, cannot be a parameter.

Repetition: from k to $n > k$: $[a]_k^n = a^k \cup a^{k+1} \cup \dots \cup a^n$

Optionality: $\epsilon \cup a$ or $[a]$

Ordered interval: $(0...9)$ $(a...z)$ $(A...Z)$

Intersection

Difference

Complement

It can be shown that Extended R.E. are not more powerful than standard R.E.

Closures REG is closed under

- Concatenation
- Union
- Star (*)
- Cross (+)
- Power
- Intersection
- Complement

Lists contains an unspecified number of elements of the same type. Lists can be represented with regex:

$$ie(se) * f$$

where i, s, f are terminal symbols denoting the beginning of the list, a separator between elements, and the end of the string.

Nested lists are possible using regex if the nesting level is limited:

$$\begin{aligned} list_1 &= i_1 \cdot list_2 \cdot (s_1 \cdot list_2)^* \cdot f_1 \\ list_2 &= i_2 \cdot list_3 \cdot (s_2 \cdot list_3)^* \cdot f_2 \\ &\dots \\ list_k &= i_k \cdot e_k \cdot (s_k \cdot e_k)^* \cdot f_k \end{aligned}$$

3 Context Free Grammars

The language $L = \{a^n b^n | n > 0\}$ is **not** regular.

Grammars a tool to define language through **rewriting rules**. Phrases are generated through repeated application of the rules.

Context Free Grammar is defined by 4 entities:

Non-terminal alphabet V

Terminal alphabet Σ , alphabet of the resulting language

Rules/Productions P

Axiom/Start $S \in V$, from which derivation starts

Rules form: $X \rightarrow \alpha$ where $X \in V \wedge \alpha \in (V \cup \Sigma)^*$. Rules can be condensed:

$$X \rightarrow \alpha_1$$

$$X \rightarrow \alpha_2$$

...

$$X \rightarrow \alpha_k$$

can be rewritten as

$$X \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_k$$

Safety conventions:

- $\{\rightarrow, |, \cup, \epsilon\} \cap \Sigma = \emptyset$
- $V \cap \Sigma = \emptyset$

Notation conventions: V elements can be distinguished using:

- $\langle \rangle$ Angle brackets surrounding elements of V
- Elements of Σ in **bold**, elements of V in *italic*
- Elements of Σ 'quoted'
- Elements of V UPPERCASE