Formal Languages and Compilers

Matteo Secco

March 7, 2021

Contents

1 Formal Language Theory

Alphabet Σ: any <u>finite</u> set of symbols $\Sigma = \{a_1, a_2, ..., a_k\}$

String: a sequence of alphabeth elements

Language: a set (possibly infinite) of strings

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

Sentences/Phrases: strings belonging to a language

Language cardinality: number of sentences of the language

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

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 $x = a_1a_2...a_h$ and $y = a_1a_2...a_k$ 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 $x = a_1a_2...a_h \wedge y = b_1b_2...b_k \implies x \cdot y = a_1a_2...a_hb_1b_2...b_k$

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

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

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

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

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

1.2 Operations on Languages

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

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

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

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

Power: inductive definition:

- $L^0 = \{\epsilon\}$
- $L^m = L^{m-1}L \quad$ 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: $L_{\text{universal}} = \Sigma^0 \cup \Sigma^1 \cup ...$

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

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

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

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

Monotonic: $L \subseteq L^*$

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

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

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

$\emptyset^* = \{\epsilon\}$

$\{\epsilon\}^* = \{\epsilon\}$

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 laguages.
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 $\Sigma = \{a_1, a_2, ..., a_n\} \cup \{\emptyset, \cup, \cdot, *\}$

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
$e_1 \cup ... \cup e_k$	$e_i \quad \forall i \in \{1, 2, ..., k\}$
e^*	ϵ 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 $\stackrel{n}{\Rightarrow}$ (deriving n times, $\stackrel{*}{\Rightarrow}$ (0 or more times), $\stackrel{+}{\Rightarrow}$ (1 or more times).

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

Equivalent r.e. defines the same language

2.4 Ambiguity of Regular Expressions

Numbered subexpressions of a R.E

- Add all passible 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: $a^n = \underbrace{aa...a}_{\text{n times}}$. NB: n is an actual number, cannot be a parameter.

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

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 represanted 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 hrough repeated application of the rules.

Context Free Grammar is defined by 4 entities:

Non-terminal aplhabet V

Terminal alphabet Σ, alphabeth of the resulting language

Rules/Productions P

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

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

$$
\begin{gathered}
X \to \alpha_1 \\
X \to \alpha_2 \\
... \\
X \to \alpha_k \\
\text{can be rewritten as} \\
X \to \alpha_1 | \alpha_2 | ... | \alpha_k
\end{gathered}
$$

.

Safety conventions:

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

Notation conventions: V elements can be distinguished using:

- <Angle brackets> surrounding elements of V
- Elements of Σ in **bold**, elements of V in *italic*
- Elements of Σ 'quoted'
- Elements of V in UPPERCASE

3.1 Types of rules

Terminal $\rightarrow u|\epsilon$

Empty/Null $\rightarrow \epsilon$

Initial/Axiomatic $S \rightarrow$

Recursive $A \rightarrow \alpha A\beta$

Left-Recursive $A \rightarrow A\beta$

Right-Recursive $A \rightarrow \alpha A$

Left-and-Right-Recursive $A \rightarrow A\beta A$

Copy/Categorization $A \rightarrow B$

Linear $\rightarrow uBv|w$

Right-linear $\rightarrow uB|w$

Left-Linear $\rightarrow Bv|w$

Homogeneous normal $\rightarrow A_1...A_n|a$

Chomsky normal $\rightarrow BC|a$

Greibach normal $\rightarrow a\sigma|b$ where $\sigma \in V^*$

Operator normal $\rightarrow AaB$

3.2 Derivation

Derivation $\implies$ Let $\beta, \gamma \in (V \cup \Sigma)^*$. Then $\beta \implies \gamma$ for grammar $G =< V, \Sigma, P, S >$ iff

$$\begin{aligned} &\beta = \delta A\eta && \wedge \\ &A \rightarrow \alpha \quad \in V && \wedge \\ &\gamma = \delta\alpha\eta \end{aligned}$$

Power, star and cross operators apply to derivation as usual

3.3 Erroneous Grammars and Useless Rules

Clean grammar $G =< V, \Sigma, P, S >$ is clean iff $\forall A \in V$

A is reachable: $S \stackrel{*}{\Rightarrow} \alpha A\beta$ where $\alpha, \beta \in (V \cap \Sigma)^*$

A is defined: $L_A(G) \neq \emptyset$ (generates a non-empty language)

(G doesn't allow for circular derivations) optional, but useful

Algorithm 1 Undefined nonterminals identification

$NEW \leftarrow \{A|(A \to u) \in P \wedge u \in \Sigma^*\}$
repeat
  $DEF \leftarrow NEW$
  $NEW \leftarrow DEF \cup \{B|(B \to D_1 D_2 ... D_n) \in P \wedge \overbrace{\forall i(D_i \in DEF \cup \Sigma)}^{D_i \text{in DEF or a terminal}}\}$
until $NEW = DEF$
$UNDEF \leftarrow V \setminus DEF$

Produce relation A produce B iff $A \to (\alpha B \beta) \in P$, where $A \neq B \wedge \alpha, \beta$ are strings

Algorithm 2 Unreachable nonterminals identification

Write the graph of the **produce** relation
Delete states that are not reachable from S

3.4 Infinite Languages and Recursion

Interesting languages are infinite. Infinite languages require the grammar generating them to be recursive

Recursive derivation $A \overset{n}{\Longrightarrow} xAy$

Immediately recursive derivation $A \overset{1}{\Longrightarrow} xAy$

Left-recursive derivation $A \overset{n}{\Longrightarrow} Ay$

Right-recursive derication $A \overset{n}{\Longrightarrow} xA$

Infinity condition $|L(G)| = \infty \iff G$ is clean $\wedge$ G avoids circular derivations $\wedge$ G allows recursive derivations

3.5 Syntax Trees and Canonical Derivations

Syntax tree A graph representing the derivation process which is

- Oriented
- Sorted (Top-down, Left-to-right)
- Acyclical
- $\forall n_1, n_2 \exists!$ a path $n_1 \leftrightarrow n_2$

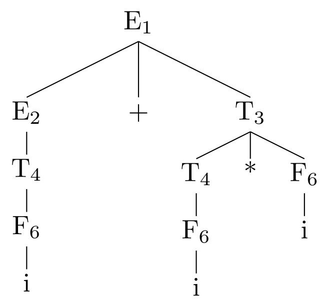
Subtree with root N is the tree having N as root, includes N and all its descendant

Example grammar

- $E \to E + T|T$
- $T \to T * F|F$
- $F \to (E)|i$

Example sentence $i + i * i$

Example tree

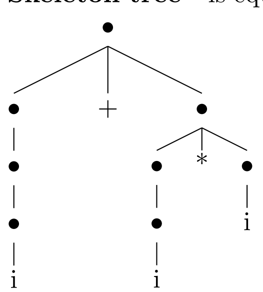


Left derivation the left-most rule is applied first

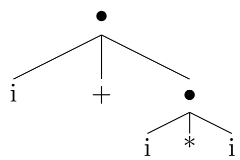
Right derivation the right-most rule is applied first

Unicity of derivations for a <u>fixed</u> syntax tree $\exists!$ the left and right derivations

Skeleton tree is equal to the syntax tree with all the non-terminals obscured



Condensed skeleton tree obtained from the skeleton tree by merging internal nodes and non-branching paths



3.5.1 Parenthesis languages

Are expressed by the Dyck language

- $\Sigma = \{a, c\}$
- $S \rightarrow aScS|\epsilon$

We can observe that $L_1 = \{a^n c^n | n \geq 1\} \subset L_{\text{DYCK}}$

3.6 Regular composition of (Context) Free Languages

The family of free languages is closed under union, concatenation, kleen star. Given $G_1 = (\Sigma_1, V_{N_1}, P_1, S_1)$ and $G_2 = (\Sigma_2, V_{N_2}, P_2, S_2$ such that $V_{N_1} \cap V_{N_2} = \emptyset \wedge S \notin (V_{N_1} \cup V_{N_2})$

Union $G = (\Sigma_1 \cup \Sigma_2, \quad V_{N_1} \cup V_{N_2} \cup \{S\}, \quad P_1 \cup P_2 \cup \{S \rightarrow S_1 S_2\}, \quad S)$

Concatenation

Kleen star