# Formal Languages and Compilers

Matteo Secco

March 22, 2021

# Contents

# 1 Formal Language Theory

**Alphabet $\Sigma$:** 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** $\epsilon$ 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:

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

## 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:

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

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

**Complement:** of $L$ over $\Sigma$: $\neg L = L_{\text{universal}} \backslash L$

**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 $\Sigma$ according to the table:

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

**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** |
|---|---|
| $e_1 \cup ... \cup e_k$ | $e_i \quad \forall i \in \{1, 2, ..., k\}$ |
| $e^*$ | $\epsilon$ or $e^n \quad \forall n \geq 1$ |
| $e^+$ | $e^n \quad \forall n \geq 1$ |

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

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

where $\gamma$ is a choice of $\beta$.
Derivation can be applied repeatedly, leading to $\stackrel{n}{\Longrightarrow}$ (deriving $n$ times, $\stackrel{*}{\Longrightarrow}$ (0 or more times), $\stackrel{+}{\Longrightarrow}$ (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 $\Sigma$
- 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:

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

## 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** $\Sigma$, 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:

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

.

**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 $\Sigma$ in **bold**, elements of $V$ in *italic*
- Elements of $\Sigma$ '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** $\Longrightarrow$ Let $\beta, \gamma \in (V \cup \Sigma)^*$. Then $\beta \Longrightarrow \gamma$ 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:** $S \xRightarrow{*} \alpha A \beta$ 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 derivation** $A \overset{n}{\Longrightarrow} xA$

**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 $n_1 \leftrightarrow n_2$

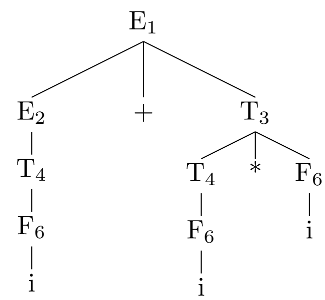
**Subtree** with root $N$ is the tree having $N$ as root, includes $N$ and all its descendant

**Example grammar**

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

**Example sentence** $i + i * i$

**Example tree**

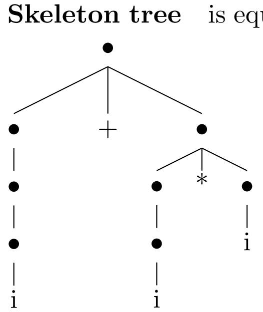



**Left derivation** the left-most rule is applied first

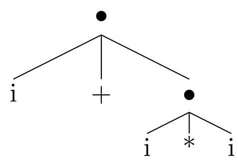
**Right derivation** the right-most rule is applied first

**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

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

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** $G_1 \cup G_2 = (\Sigma_1 \cup \Sigma_2, \quad V_{N_1} \cup V_{N_2} \cup \{S\}, \quad P_1 \cup P_2 \cup \underbrace{\{S \to S_1 | S_2\}}_{\text{execute concatenation at the beginning}}, \quad S)$

**Concatenation** $G_1 G_2 = (\Sigma_1 \cup \Sigma_2, \quad \{S\} \cup V_{N_1} \cup V_{N_2}, \quad P_1 \cup P_2 \cup \underbrace{\{S \to S_1 | S_2\}}_{\text{choose one of the languages at the beginning}}, \quad S)$

**Kleen star** $G_1^* = (\Sigma_1, \{S\} \cup V_{N_1}, \quad P_1 \cup \underbrace{\{S \to SS_1 | \epsilon\}}_{\text{Perform repetition}}, \quad S$

**Cross** $G_1^* = (\Sigma_1, \{S\} \cup V_{N_1}, \quad P_1 \cup \underbrace{\{S \to SS_1 | S_1\}}_{\text{Perform repetition}}, \quad S$

## 3.7 Ambiguity

**Syntactic ambiguity** A sentence $x$ of a grammar $G$ is ambiguous if it admits multiple distinct syntax trees

#### Degree of ambiguity DOA

**of a sentence** $x$ of a language $L(G)$: $DOA(x)$ =number of distinct trees for $x$ compatible with $G$

**of a grammar** $DOA(G) = max(\{DOA(x) | x \in L(G)\})$. It may happen that $DOA(G) = \infty$

**Determining if a grammar is ambiguous** is a semi-decidable problem. Can be proven only if the grammar is ambiguous.

### 3.7.1 Ambiguous forms and remedies

**Bilateral recursion** $S \to SxS|y$ where $x, y \in \Sigma \cup V$

[Right-recursive] $S \to yS|y$

[Left-recursive] $S \to Sy|y$

**Left and right recursion in different rules** $S \to Sa|bS|c$

[Separate] $S \to AcB$, $A \to Aa$, $B \to bB$

[Enforce order] $S \to aS|B$, $B \to Xb|c$

**Union** If $G = G_1 \cup G_2$ and $L(G_1) \cap L(G_2) \neq \emptyset$ then some sentences in $G$ can be derived using both the rules of $G_1$ or the rules of $G_2$

[Disjoint] provide disjointed set of rules: $G = (G_1 \cap G_2) \cup (G_1 \setminus G_2) \cup (G_2 \setminus G_1)$ and the rules of these subsets are disjointed

**Concatenation** $G = G_1G_2$ is ambiguous if

$$\exists x_1, u \in L_1 \exists x_2, z \in L_2 \exists v \neq \epsilon | x_1 = uv \wedge x_2 = vz$$

$$S \Rightarrow S_1S_2 \stackrel{+}{\Longrightarrow} uS_2 \stackrel{+}{\Longrightarrow} uvz \qquad \wedge \qquad S \Rightarrow S_1S_2 \stackrel{+}{\Longrightarrow} uvS_2 \stackrel{+}{\Longrightarrow} uvz$$

**Inherent ambiguity** $L$ is inherently ambiguous if any grammar $G$ for $L$ is ambiguous.

[Avoidance] inherent ambiguity is rare and can be avoided

**Others** See slides

For practical purposes, it is also possible to modify the language (and for programming languages this may be desirable)

## 3.8 Grammar equivalence

**Weak equivalence** $G_1$ and $G_2$ are weakly equivalent if $L(G_1) = L(G_2)$. Semi decidable

**Strong equivalence** $G_1$ is strongly/structurally equivalent to $G_2$ if $L(G_1) = L(G_2) \wedge G_1$ and $G_2$ have the same **condensed skeleton tree**. Decidable

## 3.9 Normal forms and Transformations

**Expansion of a non-terminal** allows to eliminate it from the rules where it appears

$$A \to xBy \quad B \to b_1|b_2|...|b_n$$

$$A \to xb_1y|xb_2y|...|xb_ny$$

**Elimination of the axiom $S$ from right parts** obtained by introducing a replacement axiom:

$$S_{new} \to S_{old}$$

### 3.9.1 Normal form without nullable nonterminals

Grammar such that $\forall A \in V \setminus \{S\} \quad \neg A \overset{+}{\Longrightarrow} \epsilon$

**Nullable non-terminals** $A$ is nullable if $\exists A \overset{+}{\Longrightarrow} \epsilon$

**Nullables set** $Null \subseteq V$

**Algorithm 3** Compute $Null$

```
repeat
  for all A ∈ V do
    if A → ϵ ∈ P then
      A ∈ Null
    else if A → A_1 A_2 ... A_n ∈ P | A_i ∈ V \ {A} and ∀A_i (A_i ∈ Null) then
      A ∈ Null
    end if
  end for
until convergence is reached
```

**Algorithm 4** Construction of non-nullable normal form

```
Compute Null
for all R ∈ P do
  for all A ∈ Null do
    for all Occurrencies of A in the right part of R do
      Remove A in the given occurrence of R
      Add the resulting rule to P
    end for
  end for
end for
for all R = A → ϵ ∈ P | A ≠ S do
  P ← P \ R
end for
Clean the grammar
Remove circularities
```

### 3.9.2 Copy rules elimination

Copy rules reduce grammar size but increase derivation length

**Example:** $loop \to while|for|repeat$

**Copy set** $Copy(A) = \{B \in V | \exists A \stackrel{*}{\Longrightarrow} B\}$

---

**Algorithm 5** Computation of $Copy$

---

**Require:** $A \stackrel{+}{\Longrightarrow} \epsilon \implies A = S$ ▷*no empty rules*

**repeat**
  **for all** $A \in V$ **do**
    $A \in Copy(A)$
    **for all** $B, C \in V$ **do**
      **if** $B \in Copy(A)$ **and** $B \to C \in P$ **then**
        $C \in Copy(A)$
      **end if**
    **end for**
  **end for**
**until** convergence is reached

---

**Algorithm 6** Definition of equivalent grammar without copy rules

---

$P' \leftarrow P \setminus \{A \to B | A, B \in V\}$ ▷*delete copy rules*

$P' \leftarrow P' \cup \{A \to a | \exists B(B \in Copy(A) \wedge (B \to a) \in P)\}$ ▷*add compensating rules*

---

### 3.9.3 Conversion from left to right recursion

**Immediate L-recursion**

$$\begin{cases} A \to AB_1|AB_2|\dots|AB_n \\ A \to a_1|a_2|\dots|a_h \end{cases}$$

becomes

$$\begin{cases} A \to a_1A'|a_2A'|\dots|a_hA' \\ A' \to B_1A'|B_2A'|\dots|B_na' \end{cases}$$

**Non-immediate** not treated here

### 3.9.4 Chomsky normal form

only 2 types of rules allowed:

- $A \to BC$
- $A \to a$

**Algorithm 7** Convert $G$ to Chomsky's normal

$P \leftarrow \emptyset$
**if** $\epsilon \in L(G)$ **then**
  $P \leftarrow \{S \rightarrow \epsilon\}$
**end if**
**for all** $R = x_0 \rightarrow x_1x_2 \ldots x_n | x_i \in \Sigma \cup V$ **do**
  $P \leftarrow P \cup \{x_0 \rightarrow X_1X_n\}$  $\triangleright X_1, X_n$ *are newly created symbols*
  $P \leftarrow P \cup \{X_s \rightarrow x_2 \ldots x_n\}$
**end for**
**if** $x_1 \in \Sigma$ **then**
  $P \leftarrow P \cup \{X_1 \rightarrow x_1\}$
**end if**

### 3.9.5 Real-time normal form

the right part of any rule has a terminal as prefix:

$$A \rightarrow a\alpha | a \in \Sigma, \alpha \in \{\Sigma \cup V\}^*$$

### 3.9.6 Greibach normal form

special case of RT-nf: right parts are a terminal followed by 0 or more nonterminals

$$A \rightarrow a\alpha | a \in \Sigma, \alpha \in V^*$$

## 3.10 CFG extensions and subsets

### 3.10.1 ENBF grammar (CFG+RE)

$G = \{V, \Sigma, P, S\}$. $|P| = |V|$. Rules are in the form $A \rightarrow \eta$, where $\eta$ is a RE over $V \cup \Sigma$

**Derivation** Given $\eta_1, \eta_2 \in (\Sigma \cup V)^*, \eta_1 \Rightarrow \eta_2$ if

- $\eta_1 = \alpha A \gamma$
- $\eta_2 = \alpha B \gamma$
- $A \rightarrow e \in P$ (e is an RE)
- $e \stackrel{*}{\Rightarrow} B$