Free Grammars - I

Translated and adapted by L. Breveglieri

CONTEXT-FREE GRAMMAR (FREE GRAMMAR)

CONTEXT-FREE LANGUAGE (FREE LANGUAGE)

LIMITATIONS OF REGULAR LANGUAGES

$begin \;\; begin \;\; begin \;\ldots\; end \;\; end \;\; end$

$L_1 = \{ x \mid x = b^n e^n, n \geq 1 \}$

L_1 is not regular $\quad b^+ e^+ \quad (be)^+$

SYNTAX – In order to define a language, one can use RULES that, after repeated application, allow to generate all and only the phrases of the language

The set of such rules constitutes a GENERATIVE GRAMMAR (or SYNTAX)

EXAMPLE – palindromes

$$L = \left\{ uu^R \mid u \in \{a,b\}^* \right\} = \{\varepsilon, aa, bb, abba, baab, ..., abbbba, ...\}$$

$frase \to \varepsilon$ – the empty string ε is a valid phrase

$frase \to a \ \ frase \ \ a$ – a phrase enclosed in *a*, *a* is a phrase

$frase \to b \ \ frase \ \ b$ – a phrase enclosed in *b*, *b* is a phrase

A derivation chain

$$frase \Rightarrow a \ \ frase \ \ a \Rightarrow ab \ \ frase \ \ ba \Rightarrow$$
$$\Rightarrow abb \ \ frase \ \ bba \Rightarrow abb\varepsilon bba = abbbba$$

The arrow "→" is a metasymbol, deserved to separate the left and right parts of a grammar rule

EXAMPLE: a list of palindromes

$abba\ bbaabb\ aa$

$lista \rightarrow frase\ \ lista$

$lista \rightarrow frase$

$frase \rightarrow \varepsilon$

$frase \rightarrow a\ \ frase\ \ a$

$frase \rightarrow b\ \ frase\ \ b$

Non-terminal symbols

lista (axiom), which defines the whole langauge phrase

frase (phrase), which defines the phrase components, i.e., the palindrome substrings that constitute the whole phrase

EXAMPLE: the metalanguage of regular expressions

Regular expressions that define the regular languages over the terminal alphabet $\Sigma = \{ a, b \}$, are a language themselves, and in particular they are strings over the alphabet $\Sigma_{r.e.} = \{a, b, \cup, *, Ø, (,)\}$

Syntax that generates $G_{r.e.}$

$$
\begin{array}{ll}
1.\ espr \to \varnothing & 4.\ espr \to (espr \cup espr) \\
2.\ espr \to a & 5.\ espr \to (espr\ espr) \\
3.\ espr \to b & 6.\ espr \to (espr)^*
\end{array}
$$

Derivation example

$$
\begin{aligned}
&espr \Rightarrow (espr \cup espr) \Rightarrow ((espr\ espr) \cup espr) \Rightarrow ((a\ espr) \cup espr) \Rightarrow \\
&\quad \Rightarrow ((a\ (espr)^*) \cup espr) \Rightarrow ((a\ ((espr \cup espr))^* \cup espr)) \Rightarrow \\
&\quad \Rightarrow ((a\ ((a \cup espr))^* \cup espr)) \Rightarrow ((a\ ((a \cup b))^* \cup espr)) \Rightarrow \\
&\quad \Rightarrow ((a\ ((a \cup b))^* \cup espr)) \Rightarrow ((a\ ((a \cup b))^* \cup b))
\end{aligned}
$$

$$L(a\{ab\}^* \cup b) = \{b, a, aa, ab, aaa, aba, \ldots\}$$

CONTEXT-FREE GRAMMAR (BNF - Backus Normal Form – of TYPE 2):
is defined by means of four entities

V	*non-terminal alphabet,* is a set of *non-terminal symbols* (or *syntactic classes*)
Σ	*terminal alphabet*, is the set of the *characters* that constitute the phrases
P	is a set of *syntactic rules* (also called *production rules* or simply *rules*)
$S \in V$	is a particular non-terminal, called *axiom*

each rule of P is an ordered pair

$$(X, \alpha), \quad X \in V \quad \text{e} \quad \alpha \in (V \bigcup \sum)^*$$

$$(X, \alpha) \in P \qquad X \to \alpha$$

$$X \to \alpha_1, X \to \alpha_2, \ldots X \to \alpha_n$$

$$X \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$$

$$X \to \alpha_1 \bigcup \alpha_2 \bigcup \ldots \bigcup \alpha_n$$

In order to avoid confusion, the metasymbols " $\to$ ", " | ", "$\cup$ " and " ε " must not apper among the terminal and non-terminal symbols; moreover, the terminal and non-terminal alphabets must not share elements (i.e., must be disjoint sets)

1st CONVENTION

< if_phrase > → if < cond > then < phrase > else < phrase >

2nd CONVENTION

*if_phrase → **if** cond **then** phrase **else** phrase*

if_phrase → ‘ if ’ cond ‘ then ’ phrase ‘ else ’ phrase

3rd CONVENTION

F → if C then D else D

CONVENTIONS USED IN THE FOLLOWING

- terminal symbols (or characters or letters) - { *a*, *b*, … }
- non-terminal symbols - { A, B, …}
- strings over Σ that contain only terminal characters - { *r*, *s*, …, *z* }
- strings over (V U Σ) that contain terminal and non-terminal elements - { α, β,… }
- strings over V that contain only non-terminal symbols - σ

TERMINAL: the right part contains terminals or empty string	$\to u \mid \varepsilon$
EMPTY: the right part is the empty string	$\to \varepsilon$
INITIAL: the left part is the axiom	$S \to$
RECURSIVE: the left part appears in the righ part	$A \to \alpha A \beta$
LEFT RECURSIVE: the left part is prefix of the right part	$A \to A\beta$
RIGHT RECURSIVE: the left part is suffix of the right part	$A \to \beta A$
TWO-SIDED RECURSIVE: superposition of previous cases	$A \to A\beta A$
COPY (or RENAMING): simply a change of name	$A \to B$
LINEAR: the right part contains at most one nonterminal	$\to uBv \mid w$
RIGHT LINEAR: like linear, but the nonterminal is suffix	$\to uB \mid w$
LEFT LINEAR: like linear, but the nonterminal is prefix	$\to Bv \mid w$
CHOMSKY NORMAL: either one term. ot two nonterm.s	$\to BC \mid a$
GREIBACH NORMAL: always a prefix term., max length 2	$\to a\sigma \mid b$
OPERATOR NORMAL: always an infix terminal	$\to AaB$

DERIVATION AND GENERATED LANGUAGE

$\beta, \gamma \in (V \cup \Sigma)^*$

γ derives from β in the grammar G

$\beta \underset{G}{\Rightarrow} \gamma$ if $A \to \alpha$ is a rule of grammar G

and $\beta = hAd, \quad \gamma = h\alpha d$

$$\beta_0 \overset{n}{\Rightarrow} \beta_n \qquad \beta_0 \overset{*}{\Rightarrow} \beta_n \qquad \beta_0 \overset{+}{\Rightarrow} \beta_n$$

THE LANGUAGE GENERATED BY G WHEN STARTING FROM A NON-TERMINAL "A" (one usually takes "S" for "A") IS $L_A(G)$

A *form* generated by G, starting from the non-terminal A $\in$ V, is a string $\alpha \in (V \cup \Sigma)^*$ such that

$$A \overset{*}{\Rightarrow} \alpha$$

$$L_A(G) = \left\{ x \in \Sigma^* \mid A \overset{+}{\Rightarrow} x \right\}$$

$$L(G) = L_S(G) = \left\{ x \in \Sigma^* \mid S \overset{+}{\Rightarrow} x \right\}$$

If the nonterminal A is the axiom, then the form is said to be *sentential*

A phrase is simply a sentential form that contains only terminal characters

EXAMPLE (the structure of a book). Grammar G_l generates the structure of a book: it contains a preface (*f*) and a series (denoted by the non-term. *A*) of one or more chapters, each one of which starts with a chapter title (*t*) and contains a series (*B*) of one or more lines (*l*)

$$S \to f\ A$$
$$A \to A\,t\,B \mid t\,B$$
$$B \to l\,B \mid l$$

Non-term. *A* generates the forms *t B t B* and eventually the string *t l l t l* $\in L_A(G_l)$

Non-term. *S* generates the sentential forms *f A t l B* and *f t B t B*

The language generated by the non-terminal *B* is $L_B(G_l) = l^+$

The language $L(G_l)$, generated by G_l, is defined by the regular expression $f\ (\ t\ l^+\ )^+$

A language is CONTEXT-FREE (or simply FREE) if and only if there exists a free grammar that generates it

In the example before, the free language $L(G_l)$ happens to be regular as well, but in the following it will be proved that in general the family LIB of context-free languages strictly contains that of regular languages

Two grammars G e G' are equivalent if and only if they generate the same language, i.e., iff the equality L(G) = L(G') holds

G_l

$$S \to fX$$
$$X \to XtY \mid tY$$
$$Y \to lY \mid l$$

G_{l2}

$$S \to f\ A$$
$$A \to At\ B \mid t\ B$$
$$B \to Bl \mid l$$

$Y \overset{n}{\underset{G_l}{\Rightarrow}} l^n$ and $B \overset{n}{\underset{G_{l2}}{\Rightarrow}} l^n$ generate the same language $L_B = l^+$

ERRONEOUS GRAMMARS AND USELESS RULES: in writing a grammar, one needs to pay attention to that every non-terminal is defined and to that each of them actually contributes to generate the strings in the language

A GRAMMAR IS IN REDUCED FORM (or simply is REDUCED) if BOTH (not either one) the following conditions hold

Every non-terminal A is reachable from the axiom, i.e., there exists a derivation as follows

$$S \overset{*}{\Rightarrow} \alpha A \beta$$

Every non-terminal A generates a non-empty set of strings

$$L_A(G) \neq \varnothing$$

REDUCTION OF THE GRAMMAR: there exists an algorithm in two phases

The first phase identifies the non-terminal symbols that are undefined

The second phase identifies those that are unreachable

PHASE 1 – construct the complement set DEF of the defined non-term. symbols

$$DEF = V \setminus UNDEF$$

Initially DEF is set to contain the non-terminal symbols that are expanded by terminal rules

$$DEF := \left\{ A \mid (A \to u) \in P, \text{ with } u \in \Sigma^* \right\}$$

Then the following transformation is applied repeatedly, until it converges

$$DEF := DEF \bigcup \left\{ B \mid (B \to D_1 D_2 \ldots D_n) \in P \right\}$$

Each symbol D_i ($1 \leq i \leq n$) either is a terminal or belongs to DEF

In every iteration of the first phase, two events are possible

1) new non-terminal symbols are identified, which have as their right hand side a string that contains terminal symbols or defined non-terminal symbols; such new symbols are added to the set DEF
2) the DEF set does not change, which means that convergence has been reached, and therefore the algorithm terminates

The non-terminal symbols that belong to the UNDEF set can be eliminated, along with the rules wherein they appear (in the left or right side)

PHASE 2 – The identification of the non-terminal symbols that are reachable starting from the axiom S can be reduced to the problem of the existence of a path in the graph associated with the following binary relation *produce*

$$A \overset{produce}{\rightarrow} B \text{ if and only if}$$

$$A \rightarrow \alpha B \beta \text{ with } A \neq B \quad \alpha, \beta \text{ are any strings}$$

The non-terminal C is reachable from the axiom S if and only if the graph of the binary relation *produce* contains a path from S to C

The unreachable non-terminal symbols can be removed from the grammar, along with the rules wherein they appear (in the left or right side)

A third property is often required for a grammar to be in the reduced form, as follows

The grammar G may not have circular derivations, which are inessential and moreover give raise to ambiguity (think of ambiguous regexps …)

$$A \overset{+}{\Rightarrow} A$$

if x is derivable as follows $A \Rightarrow A \Rightarrow x$

then x is derivable also as follows $A \Rightarrow x$

In the following, it will be shown why and how a circular derivation necessarily gives raise to an ambiguous behaviour

EXAMPLES - NOT REDUCED

1) $\{ S \to aASb, \quad A \to b \}$

2) $\{ S \to a, \quad A \to b \} \qquad \{ S \to a \}$

3) $\{ S \to aASb \mid A, \quad A \to S \mid c \} \qquad \{ S \to aSSb \mid c \}$

CAUTION: circularity may raise from a null rule, as follows

$$X \to XY \mid \ldots \quad Y \to \varepsilon \mid \ldots$$

CAUTION: even if properly reduced, a grammar may still exhibit redundancy, and therefore may behave ambiguously, as follows

the rule pairs (1, 4) and (2, 5) generate the same phrases

1) $S \to aASb$ 4) $A \to c$

2) $S \to aBSb$ 5) $B \to c$

3) $S \to \varepsilon$

RECURSION AND INFINITY OF THE LANGUAGE

Almost all the formal languages of practical interest are infinite (= contain infinitely many strings). What is the feature that enables a free grammar to generate infinitely many strings, i.e., an infinite language?

IN ORDER TO BE ABLE TO GENERATE INFINITELY MANY STRINGS, IT IS NECESSARY FOR THE RULES TO ALLOW DERIVATIONS OF UNBOUNDED LENGTH

<u>THIS IS TO SAY THAT THE GRAMMAR IS RECURSIVE</u>

$A \overset{n}{\Rightarrow} xAy \quad n \geq 1$ is recursive

if $n = 1$ is immediately recursive

A is a recursive nonterminal

if $x = \varepsilon$ is left recursive

if $y = \varepsilon$ is right recursive

A NECESSARY AND SUFFICIENT CONDITION for a language L(G) to be infinite, where G is a grammar in the reduced form and does not have any circular derivations, is that G admits recursive derivations

PROOF OF NECESSITY: if there were not any recursive derivation, then every derivation would be of bounded length, hence L(G) would be finite (= would contain finitely many strings)

PROOF OF SUFFICENCE:
If there were such a derivation as

$$A \overset{n}{\Rightarrow} x\,A\,y$$

this derivation would be possible but since G is in the reduced form, it follows that A is reachable from S

$$A \overset{+}{\Rightarrow} x^m\,A\,y^m \quad \text{for } m \geq 1 \text{ and } x, y \text{ not both empty}$$

$$S \overset{*}{\Rightarrow} u\,A\,v$$

hence from A at least one terminal string *w* can be derived

$$A \overset{+}{\Rightarrow} w$$

whence there exist infinitely many derivations that yield different strings

$$S \overset{*}{\Rightarrow} u\,A\,v \overset{+}{\Rightarrow} u\,x^m\,A\,y^m\,v \overset{+}{\Rightarrow} u\,x^m\,w\,y^m\,v, \quad (m \geq 1)$$

A GRAMMAR DOES NOT CONTAIN RECURSION (or is RECURSION-FREE)
IF AND ONLY IF THE GRAPH OF THE BINARY RELATION *produce* IS ACYCLIC
(= does not contain any closed path)

EXAMPLE (for an infinite language)

G generates the following finite language

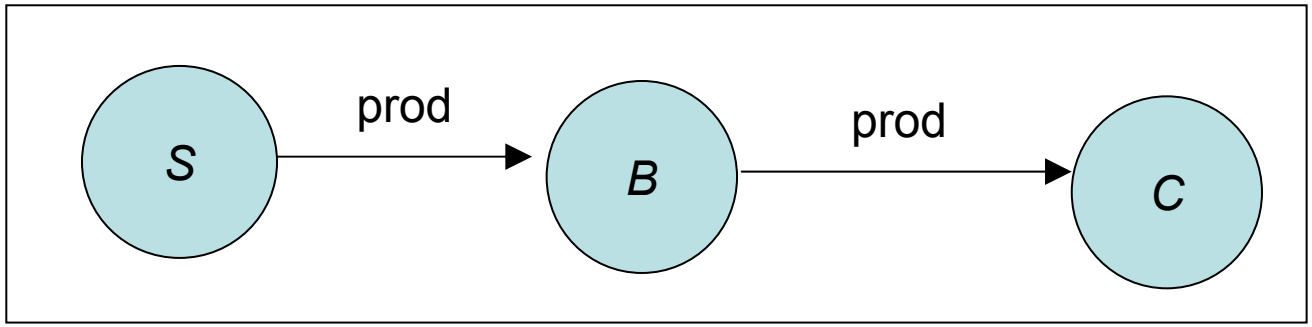
$$\{aabc, acac\}$$

G

$$S \to a\,B\,c$$

$$B \to a\,b \mid C\,a$$

$$C \to c$$



EXAMPLE (arithmetic expression)

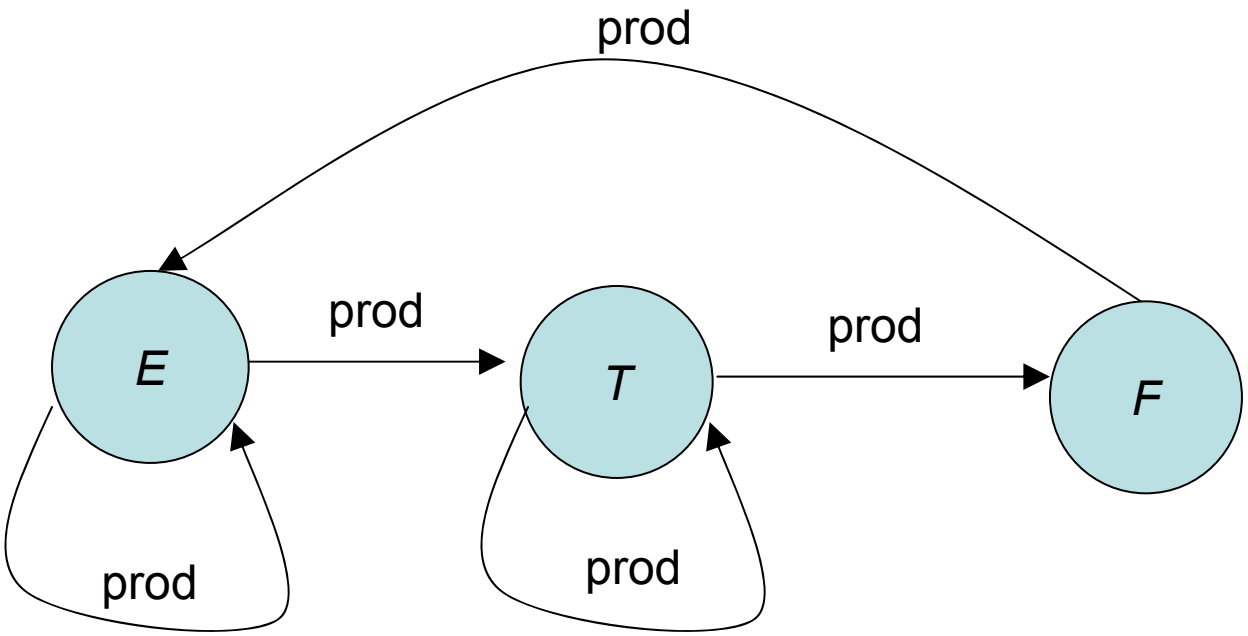
$$G = \left(\{ E, T, F \}, \{ i, +, *,), (\}, P, E \right)$$

$$E \to E+T \mid T \qquad T \to T*F \mid F \qquad F \to (E) \mid i$$

$$L(G) = \{ i, i+i+i, i*i, (i+i)*i, \ldots \}$$

F (factor)	is involved in an indirect recursion
E (expression)	is involved in an immediate left recursion
T (term)	is involved in an immediate left recursion

G is in the reduced form and does not contain any circular derivations, therefore the generated language L(G) is infinite


prod
E
prod
T
prod
F
prod
prod