Free Grammars - I

Prof. Licia Sbattella

aa 2007-08

Translated and adapted by L. Breveglieri

CONTEXT-FREE GRAMMAR (FREE GRAMMAR)

CONTEXT-FREE LANGUAGE (FREE LANGUAGE)

LIMITATIONS OF REGULAR LANGUAGES

$$\boxed{\begin{array}{l} \textit{begin begin begin} \ldots \textit{end end end} \\ L_1 = \{x \mid x = b^n e^n, n \geq 1\} \\ L_1 \quad \text{is not regular} \quad b^+ e^+ \quad (be)^+ \end{array}}$$

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 grammatical rule.

EXAMPLE: a list of palindromes

$abba\ bbaabb\ aa$

$$lista \rightarrow frase\ \ lista \qquad frase \rightarrow \varepsilon$$

$$lista \rightarrow frase \qquad frase \rightarrow a\ \ frase\ \ a$$

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

Non-terminal symbols: *lista* (axiom) and *frase* (phrase, defines some components, that is, 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 themselves a language and in particular are strings over the alphabet $\Sigma_{e.r.} = \{a, b, \cup, *, \emptyset, (,)\}$.

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

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

Derivation example:

$$espr \Rightarrow (espr \bigcup espr) \Rightarrow ((espr\ espr) \bigcup espr) \Rightarrow ((a\ espr) \bigcup espr) \Rightarrow$$

$$\Rightarrow ((a\ (espr)^*) \bigcup espr) \Rightarrow ((a\ ((espr \bigcup espr))^* \bigcup espr)) \Rightarrow$$

$$\Rightarrow ((a\ ((a \bigcup espr))^* \bigcup espr)) \Rightarrow ((a\ ((a \bigcup b))^* \bigcup espr)) \Rightarrow$$

$$\Rightarrow ((a\ ((a \bigcup b))^* \bigcup espr)) \Rightarrow ((a\ ((a \bigcup b))^* \bigcup b))$$

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

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

1. *V, non-terminal alphabet,* is a set of non-terminal symbols (or metasymbols)
2. *Σ, terminal alphabet*, is the set of the characters constituting the phrases
3. *P,* is the set of the *syntactic rules* (also called *production rules*)
4. *S* $\in$ *V,* is a particular non terminal, called *axiom*.

Each rule of P is an ordered pair:

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

$$(X,\alpha) \in P \quad X \rightarrow \alpha$$

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

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

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

In order to avoid confusion, the metasymbols ‘ $\rightarrow$ ’, ‘ | ’, ‘$\cup$ ’ 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 containing only terminal characters - { *r*, *s*, …, *z* }
- Strings over (V U Σ) containing terminal and non-terminal elements - { α, β,… }
- Strings over V containing only non-terminal symbols - σ

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

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

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

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

A *form* generated by G, starting from the non-term. A $\in$ V, is a string α $\in$ (V $\cup$ Σ)* such that:

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

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

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

If A is the axiom, 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*).

$$\boxed{\begin{array}{l} S \to f\ A \\ A \to At\ B \mid t\ B \\ B \to l\ B \mid l \end{array}}$$

A generates forms of the type $t\ B\ t\ B$ and eventually the string $t\ l\ l\ t\ l \in L_A(G_l)$
S generaters the sentential forms $f\ A\ t\ l\ B$ and $f\ t\ B\ t\ B$.

The language generated by *B* is $L_B(G_l) = l^+$.
$L(G_l)$, generated by G_l, is defined by the regular expressions $f\ (\ t\ l^+\ )^+$.

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, that is if the equality L(G) = L(G') holds.

G_l

$$\boxed{\begin{array}{l} S \to fX \\ X \to XtY \mid tY \\ Y \to lY \mid l \end{array}}$$

G_{l2}

$$\boxed{\begin{array}{l} S \to f\,A \\ A \to At\,B \mid t\,B \\ B \to Bl \mid l \end{array}}$$

$$\boxed{Y \overset{n}{\underset{G_l}{\Rightarrow}} l^n \quad \text{and} \quad B \overset{n}{\underset{G_{l2}}{\Rightarrow}} l^n \qquad \text{generate the same language} \qquad L_{\mathrm{B}} = l^+}$$

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

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

Every non-terminal A is reachable from the axiom, that is there exists a derivation as follows:

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

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

$$\boxed{L_A(G) \neq \emptyset}$$

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 those that are unreachable.

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

$$\boxed{DEF = V \setminus UNDEF}$$

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

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

Then the following transformation is applied repeatedly, as long as it converges:

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

Each symbol D_i ($1 \le i \le n$) 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, that have as their right hand side a string containing 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 convergence has been reached, and therefore the algorithm terminates.

The non-terminal symbols belonging to the UNDEF set can be eliminated, along with the production rules wherein they appear (on the left or right sides).

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*:

$$\boxed{\begin{array}{l} A \overset{produce}{\rightarrow} B \ \text{ if and only if} \\ A \rightarrow \alpha B \beta \ \text{ with } A \neq B \quad \alpha, \beta \text{ are any strings} \end{array}}$$

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 production rules wherein they appear (on the left or right sides).

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

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

$$\boxed{\begin{array}{l} A \overset{+}{\Rightarrow} A \\ \text{if } x \text{ is derivable as follows } A \Rightarrow A \Rightarrow x \\ \text{then } x \text{ is derivable also as follows } A \Rightarrow x \end{array}}$$

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

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

CAUTION: circularity may raise from a null production 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) $S \to aASb$ $\quad$ 4) $A \to c$

2) $S \to aBSb$ $\quad$ 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, that is an infinite language?

IN ORDER TO BE ABLE TO GENERATE INFINITELY MANY STRINGS, IT IS NECESSARY FOR THE PRODUCTION RULES TO PERMIT DERIVATIONS OF UNBOUNDED LENGTH.
<u>THIS IS TO SAY THAT THE GRAMMAR NEED BE 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 reduced form and not containing circular derivations, is that G can admit recursive derivations.

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

PROOF OF SUFFICIENCE:
If there were such a derivation as:

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

this derivation would be possible:

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

but as G is in reduced form, it follows:

A is reachable from S

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

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

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

and hence, there exist infinitely many derivations yielding as many different strings:

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

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

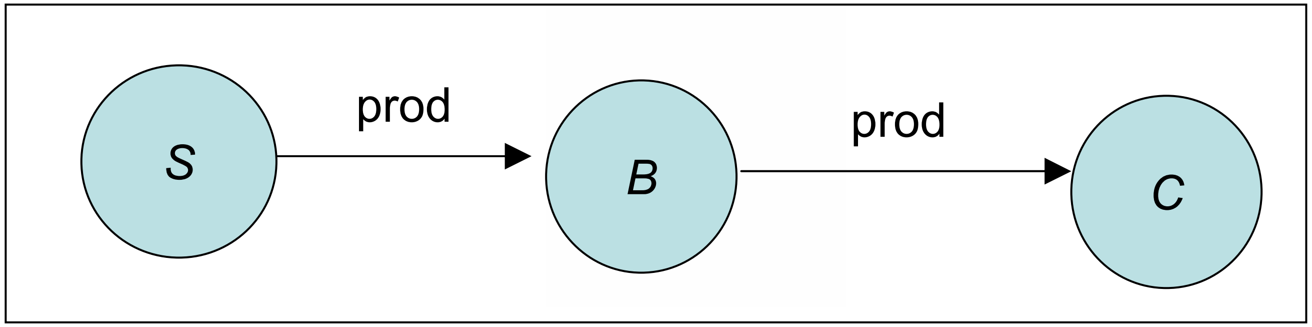
G

EXAMPLE (for an infinite language)

$$\boxed{\begin{array}{l} S \to a\,B\,c \\ B \to a\,b \mid C\,a \\ C \to c \end{array}}$$

G generates the following finite language

$$\boxed{\{aabc, acac\}}$$



EXAMPLE (arithmetic expression)

$$\boxed{\begin{array}{l} G = \left(\left\{ E,\, T,\, F \right\},\, \left\{ i,\, +,\, *,\,),\, (\right\},\, P,\, E \right) \\ E \to E + T \mid T \quad T \to T * F \mid F \quad F \to (E) \mid i \\ L(G) = \left\{ i,\, i+i+i,\, i*i,\, (i+i)*i,\, \ldots \right\} \end{array}}$$

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

$$\begin{tikzcd}
E \arrow[r, "\text{prod}"] \arrow[loop below, "\text{prod}"] & T \arrow[r, "\text{prod}"] \arrow[loop below, "\text{prod}"] & F \arrow[ll, bend right, "\text{prod}"']
\end{tikzcd}$$

Bibliography

– S. Crespi Reghizzi, *Linguaggi Formali e Compilazione*, Pitagora Editrice Bologna, 2006

– Hopcroft, Ullman, *Formal Languages and their Relation to Automata*, Addison Wesley, 1969

– A. Salomaa – *Formal Languages*, Academic Press, 1973

– D. Mandrioli, C. Ghezzi – *Theoretical Foundations of Computer Science*, John Wiley & Sons, 1987

– L. Breveglieri, S. Crespi Reghizzi, *Linguaggi Formali e Compilatori: Temi d'Esame Risolti,* web site (eng + ita)