Free Grammars - A

Translated and adapted by L. Breveglieri

CONTEXT-FREE GRAMMAR (FREE GRAMMAR)

CONTEXT-FREE LANGUAGE (FREE LANGUAGE)

LIMITATIONS OF REGULAR LANGUAGES

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

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

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

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

$abba \ bbaabb \ aa$

$$lista \to frase \ lista \qquad frase \to \varepsilon$$

$$lista \to frase \qquad frase \to a \ frase \ a$$

$$frase \to b \ frase \ b$$

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

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 ∈ *V* is a particular non-terminal, called *axiom*

each rule of P is an ordered pair

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

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

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

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

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

In order to avoid confusion, the metasymbols “ → “, “ | “, “∪ “ 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

DERIVATION AND GENERATED LANGUAGE

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

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 ∈ V, is a string α ∈ (V ∪ Σ)* such that

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

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

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

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)

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

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

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

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

G_{l2}

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

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

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

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

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

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

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

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

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

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

Then the following transformation is applied repeatedly, until it converges

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

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

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 …)

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

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

EXAMPLES - NOT REDUCED

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

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

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

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

$$\begin{array}{llll} 1) & S \to aASb & 4) & A \to c \\ 2) & S \to aBSb & 5) & B \to c \\ 3) & S \to \varepsilon & & \end{array}$$

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

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

but since G is in the reduced form,
it follows that A is reachable from S

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

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

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

A GRAMMAR DOES NOT 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)

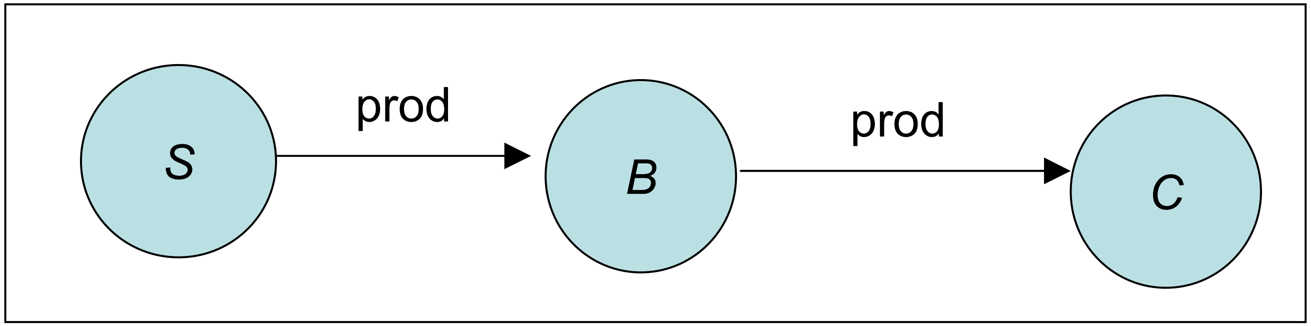
G

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

EXAMPLE (for an infinite language)

G generates the following finite language

$$\{aabc, acac\}$$



EXAMPLE (arithmetic expression)

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

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

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

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

SYNTAX TREE AND CANONICAL DERIVATION

SYNTAX TREE: directed acyclic graph, such that for every pair of nodes there exists one and only one path (not necessarily directed) that connects them

THE SYNTAX TREE

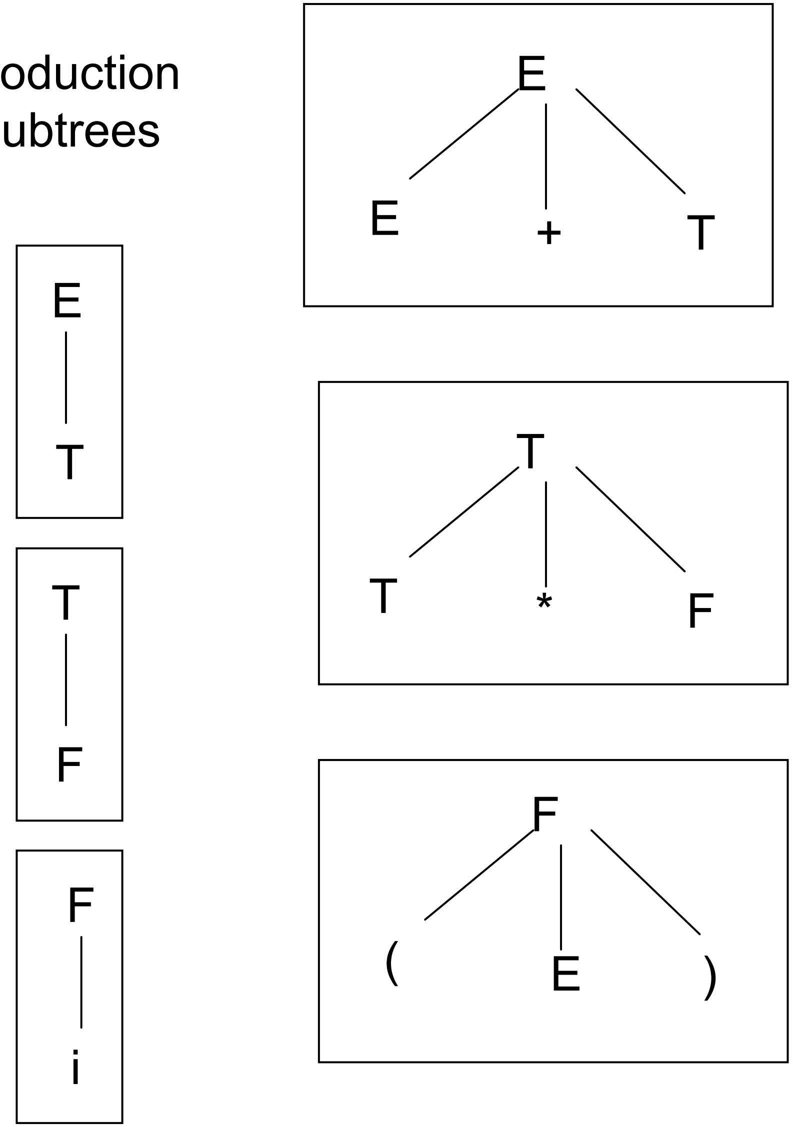
represents graphically the derivation process
gives a *parent-child* relation or a *root-node-leaf* relation
the sequence of leaves, scanned from left to right, is the so-called *frontier*
the *degree* of a node (so-called node *arity*) is the length of the rule

SUBTREE with root N: the tree that has root N and includes all the descendants of N (the immediate siblings of N, the siblings of the siblings, and so on)

SYNTAX TREE: the root is the axiom and the frontier is the generated phrase grammar

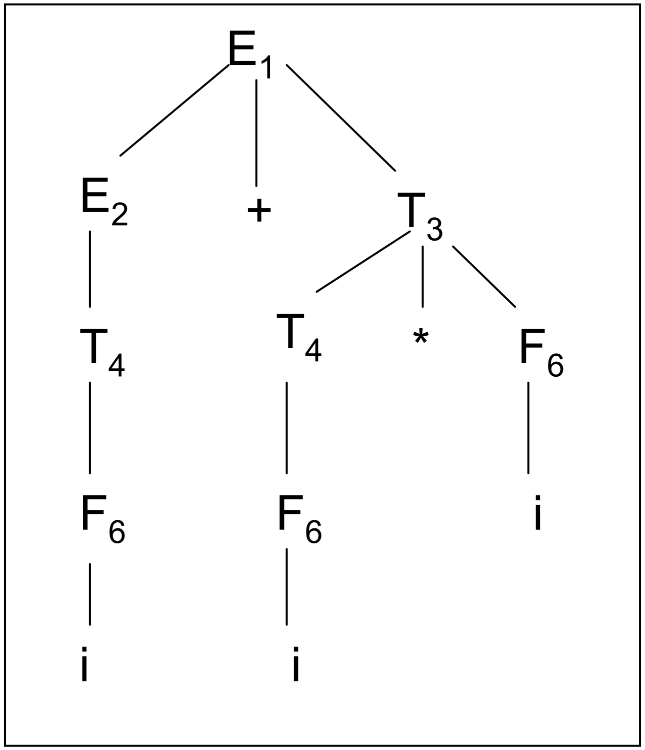
1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T * F$
4. $T \rightarrow F$
5. $F \rightarrow (E)$
6. $F \rightarrow i$

production subtrees



parenthesized representation of the syntax tree

$$\boxed{[[[[i]_F\,]_T\,]_E + [[[i]_F\,]_T * [i]_F\,]_T\,]_E}$$



syntax tree

LEFTMOST DERIVATION

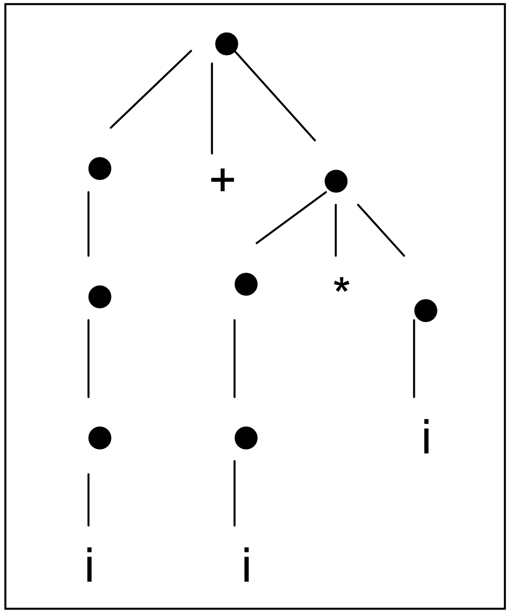
$$E \underset{1}{\Rightarrow} E+T \underset{2}{\Rightarrow} T+T \underset{4}{\Rightarrow} F+T \underset{6}{\Rightarrow} i+T \underset{3}{\Rightarrow} i+T*F \underset{4}{\Rightarrow}$$
$$\underset{4}{\Rightarrow} i+F*F \underset{6}{\Rightarrow} i+i*F \underset{6}{\Rightarrow} i+i*i$$

RIGHTMOST DERIVATION

$$E \underset{1}{\Rightarrow} E+T \underset{3}{\Rightarrow} E+T*F \underset{6}{\Rightarrow} E+T*i \underset{4}{\Rightarrow} E+F*i \underset{6}{\Rightarrow} E+i*i \underset{2}{\Rightarrow}$$
$$\underset{2}{\Rightarrow} T+i*i \underset{4}{\Rightarrow} F+i*i \underset{6}{\Rightarrow} i+i*i$$

SKELETON TREE (only the frontier and the connections)

skeleton syntax tree

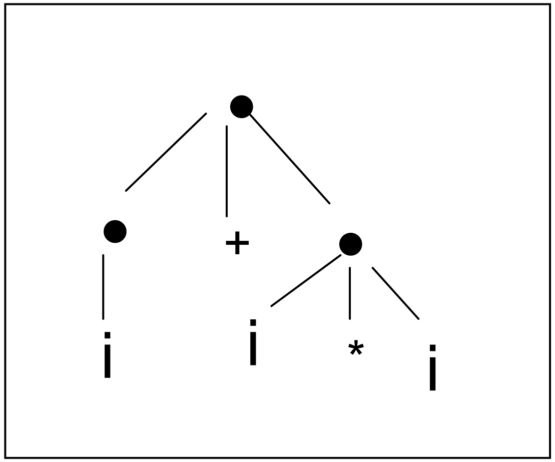


parenthesized representation of the syntax tree

$$\boxed{[[[[i]]]+[[[i]]*[i]]]}$$

CONDENSED SKELETON TREE (merge into one all the internal nodes aligned on a linear path, i.e., a path without branch points)

condensed tree



$$\boxed{[[i]+[[i]*[i]]]}$$

condensed representation of the syntax tree

LEFTMOST AND RIGHTMOST DERIVATION / FREE GRAMMAR

$$\boxed{E \overset{+}{\underset{s}{\Rightarrow}} i + i * i}$$

$$\boxed{E \overset{+}{\underset{d}{\Rightarrow}} i + i * i}$$

legenda

s: left
d: right

$$\boxed{E \overset{+}{\underset{s,d}{\Rightarrow}} i + i * i}$$

$$\boxed{\begin{array}{l} E \underset{s,d}{\Rightarrow} E + T \underset{d}{\Rightarrow} E + T * F \underset{s}{\Rightarrow} T + T * F \Rightarrow T + F * F \underset{d}{\Rightarrow} T + F * i \underset{s}{\Rightarrow} \\ \qquad \underset{s}{\Rightarrow} F + F * i \underset{d}{\Rightarrow} F + i * i \underset{d}{\Rightarrow} i + i * i \end{array}}$$

EVERY PHRASE OF A FREE GRAMMAR CAN BE GENERATED BY MEANS OF A LEFTMOST DERIVATION (and by a RIGHTMOST DERIVATION as well).

PARENTHESES LANGUAGE: artificial languages frequently contain nested structures, i.e., parentheses, where an element pair starts and ends some substructure (like for instance a substring), and where the pair may in turn contain a nested pair, and so on recursively down to arbitrary depth

Pascal:	begin ... end
C:	{ … }
XML:	< title > … < /title >
LaTeX:	\begin{equation} … \end{equation}

Do not consider the specific way the marker symbols are encoded

The paradigm of parenthesis languages is known as the *Dyck Language*

Alphabet $\Sigma = \{')', '(', ']', '['\}$ Phrases $()[[()[]]()]$

Parenthesis phrases can be equivalently defined by means of *cancellation rules*: repeatedly remove from the string any factor that consists of a pair of adjacent open and closed parentheses, as long as it is possible.
The original string is valid if and only if the final result is the *empty* string

$$[\ ] \Rightarrow \varepsilon \qquad (\ ) \Rightarrow \varepsilon$$

DYCK LANGUAGE: open parentheses *a, b, ...,* closed parentheses *a’, b’, ...*

↓

$$\Sigma = \{a, a', b, b'\}$$

$$S \to aSa'S \mid bSb'S \mid \varepsilon$$

$$\underbrace{a\,\underbrace{a\,\underbrace{a a'}\,a'}\,\underbrace{a\,\underbrace{a\,\underbrace{a a'}\,a'}\,a'}\,a'}$$

NON-REGULAR LANGUAGE

↓

$$L_1 = \{a^n c^n \mid n \geq 1\} = \{ac, aacc, ...\}$$

$$S \to aSc \mid ac$$

↓

L_1 is a subset of the Dyck language.
It does not admit more than one parenthesis nest

REGULAR COMPOSITION OF FREE LANGUAGES

The basic regular operations (union, concatenation and star), applied to free languages, still yield free languages

The family of free languaegs is closed with respect to the language operations *union*, *concatenation* and *star*

$$\boxed{\begin{array}{l} G_1=\left(\Sigma_1,V_{N_1},P_1,S_1\right)\ \text{ e }\ G_2=\left(\Sigma_2,V_{N_2},P_2,S_2\right) \\ V_{N_1}\bigcap V_{N_2}=\varnothing \qquad S\notin(V_{N_1}\bigcup V_{N_2}) \end{array}}$$

UNION $$\boxed{G=(\Sigma_1\bigcup\Sigma_2,\{S\}\bigcup V_{N_1}\bigcup V_{N_2},\{S\to S_1\,|\,S_2\}\bigcup P_1\bigcup P_2,S)}$$

CONCATENATION $$\boxed{G=(\Sigma_1\bigcup\Sigma_2,\{S\}\bigcup V_{N_1}\bigcup V_{N_2},\{S\to S_1S_2\}\bigcup P_1\bigcup P_2,S)}$$

STAR: G of $(L_1)^*$ is obtained by adding the rules $S \to S\ S_1\ |\ \varepsilon$ to G_1

CROSS: G of $(L1)^+$ is obtained by adding the rules $S \to S\ S_1\ |\ S_1$ to G_1

EXAMPLE: union of languages

$$L = \left\{ a^i b^i c^* \mid i \geq 0 \right\} \bigcup \left\{ a^* b^i c^i \mid i \geq 0 \right\} = L_1 \bigcup L_2$$

$$\begin{array}{l} G_1 \\ S_1 \rightarrow XC \\ X \rightarrow aXb \mid \varepsilon \\ C \rightarrow cC \mid \varepsilon \end{array}$$

$$\begin{array}{l} G_2 \\ S_2 \rightarrow AY \\ Y \rightarrow bYc \mid \varepsilon \\ A \rightarrow aA \mid \varepsilon \end{array}$$

$$\left\{ S \rightarrow S_1 \mid S^{"} \right\} \bigcup P_1 \bigcup P^{"}$$

$$\begin{array}{l} G^{"} \\ S^{"} \rightarrow AX \\ X \rightarrow bXc \mid \varepsilon \\ A \rightarrow aA \mid \varepsilon \end{array}$$

CAUTION: if the hypothesis that the two non-terminal alphabets are disjoint does not hold, then the construction above yields a grammar that generates a superset of the real union language. For example, replacing G_2 with G’’ would allow us to generate the invalid phrase shown aside

$$abcbc$$

The family LIB of free languages is closed with respect to STRING MIRRORING

Given the grammar G of the language, the grammar G_R that generates the mirror image of the language is obtained from G by mirroring the right member of every rule of G

REG and LIB are both closed with respect to union, concatenation and star

but later it will be proved that

they do not behave in the same way as for complement and intersection

AMBIGUITY: semantic versus syntactic
“Vedo un uomo in giardino con il cannocchiale”
(do I watch a man who has a spyglass or do I use a spyglass to watch a man ?)
“La pesca è bella” (the peach is fine or the sport of fishing is fine ?)
“half baked chicken” (a chicken that is half baked or a half chicken that is baked ?)

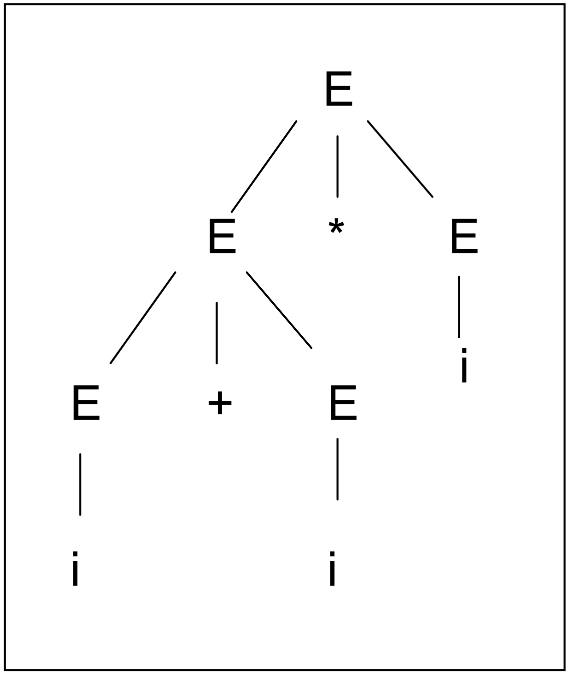
Consider SYNTACTIC AMBIGUITY. A phrase *x* of the language defined by the grammar G is said to be ambiguous if it admits two or more different syntactic trees (or, equivalently, two different derivations). A grammar G is said to be ambiguous if it generates (at least) an ambiguous string

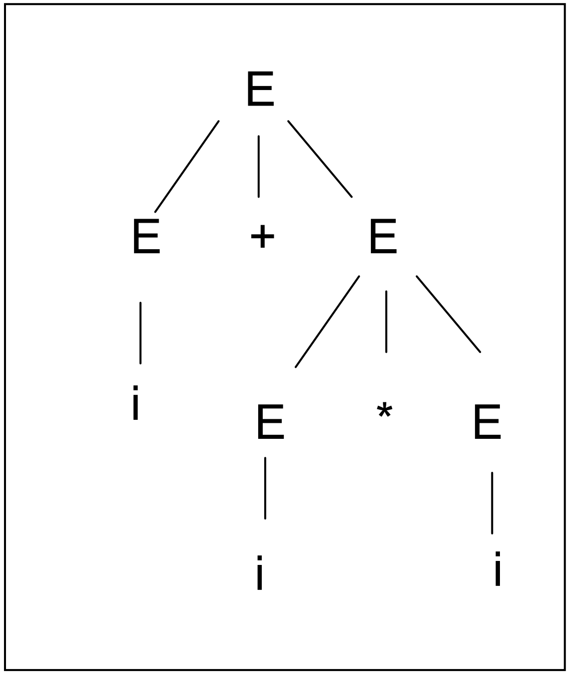
EXAMPLE – grammar G’ of the arithmetic expressions

$$E \to E + E \mid E * E \mid (E) \mid i$$

$$E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow i + E * E \Rightarrow i + i * E \Rightarrow i + i * i$$

$$E \Rightarrow E + E \Rightarrow i + E \Rightarrow i + E * E \Rightarrow i + i * E \Rightarrow i + i * i$$





The phrase *i* + *i* * *i* is ambiguous. Therefore grammar G’ is itself ambiguous

In fact, grammar G’ is not compliant with the standard convention
that multiplication must precede addition

Previously, an unambiguous grammar G that generates the arithmetic expressions has been shown. G’ is equivalent to G (that is, L(G) = L(G’)), but is smaller than G. However, grammar G’ is ambiguous. This is often the case: simplifying the grammar frequently ends up with having an ambiguous behaviour, in general

The AMBIGUITY DEGREE of a phrase *x* in the language L(G) is the number of different syntax trees that *x* admits. The ambiguity degree of a string may be infinite (only if there are nullable non-terminal symbols)
The AMBIGUITY DEGREE of a grammar G is the maximum ambiguity degree of the strings generated by G. While every string of L(G) may have a finite ambiguity degree, the ambiguity degree of G may still turn out to be unbounded

RELEVANT PROBLEM: decide whether a grammar G is ambiguous or not.
UNFORTUNATELY, THIS PROBLEM IS UNDECIDABLE: there does not exist any algorithm to decide whether a given grammar G is ambiguous or not.
This can be done for some grammars, but in general not for every grammar

AS AMBIGUITY MAY BE DIFFICULT TO IDENTIFY A POSTERIORI
IT IS ADVISABLE TO AVOID IT BY CONSTRUCTION (A FORTIORI)

EXAMPLE: resume the previous example

The phrase *i* + *i* + *i* has ambiguity degree equal to 2

The phrase *i* + *i* * i + *i* has ambiguity degree equal to 5 (see below)

$$\underbrace{\underbrace{i+i}*\underbrace{i+i}}, \quad \underbrace{\underbrace{i+\underbrace{i*i}}+i}, \quad \underbrace{i+\underbrace{\underbrace{i*i}+i}}, \quad \underbrace{\mathrm{i}+\underbrace{\mathrm{i}*\underbrace{\mathrm{i}+\mathrm{i}}}}, \quad \underbrace{\underbrace{\underbrace{i+i}*i}+i}$$

Typically, as the derived phrase gets longer, its ambiguity degree grows as well

DIRECTORY OF AMBIGUOUS MODELS AND HOW TO REMOVE AMBIGUITY

AMBIGUITY OF TWO-SIDED RECURSION: a non-terminal symbol that is recursive both on the left and on the right (two-sided recursion) always allows two or more derivations, and thus gives rise to an ambiguous behaviour (see below)

$$\boxed{A \overset{+}{\Rightarrow} A\ldots \quad A \overset{+}{\Rightarrow} \ldots A}$$

EXAMPLE 1: grammar G_1 generates the phrase *i* + *i* + *i* in two different ways (by means of two different leftmost derivations)

ambiguous grammar

$$\boxed{G_1 : \quad E \rightarrow E + E \mid i}$$

notice that L is regular

non-ambiguous version

$$\boxed{\begin{array}{l} L(G_1) = i(+i)^* \\ E \rightarrow i + E \mid i \\ E \rightarrow E + i \mid i \end{array}}$$

EXAMPLE 2: ambiguity that originates form left and right recursion, separately

ambiguous grammar

$$G_2 : \quad A \to aA \mid Ab \mid c$$

ambiguity can be removed by separately generating the two lists

$$L(G_2) = a^* cb^*$$

$$S \to AcB$$

$$A \to aA \mid \varepsilon$$

$$B \to bB \mid \varepsilon$$

or one may choose to orderly generate the two lists (first *a* then *b*, or viceversa)

$$L(G_2) = a^* cb^*$$

$$S \to aS \mid X$$

$$X \to Xb \mid c$$

EXAMPLE 3: the grammar that generates Polish postfix expressions that contain addition and multiplication, has a left recursion (but not right) and is not ambiguous

$$S \to +SS \mid \times SS \mid i$$

AMBIGUITY OF UNION (maybe the simplest to understand)

If two languages $L_1(G_1)$ and $L_2(G_2)$ share some phrases (i.e., their intersection is not empty), the grammar G of the union language, constructed as shown before, is surely ambiguous

CAUTION: one needs to assume that the two non-terminal sets are disjoint, otherwise, as seen before, the union grammar would generate
a superlanguage that strictly contains both languages

A phrase *x* that belongs to the union of L_1 and L_2, and admits two different derivations, the first only using rules of G_1 and the second only using rules of G_2, is ambiguous for the grammar G, as G contains both sets of rules. Only the phrases that belong to $L_1 \setminus L_2$ or to $L_2 \setminus L_1$, if any, would be generated in a non-ambiguous way

EXAMPLE 1: ambiguity of union may arise when in a programming language a special case, out of a general one, is treated by means of separate rules

$$\boxed{\begin{array}{l} E \to E + 1 \\ E \to inc\ E \end{array}}$$

separate rule

$$\boxed{\begin{array}{l} E \to E + T \mid T \quad T \to V \mid C \quad V \to \quad 1 \\ C \to 0 \mid 1B \mid \ldots \mid 9B \quad B \to 0B \mid 1B \mid \ldots \mid 9B \mid \varepsilon \end{array}}$$

basic grammar

EXAMPLE 2: ambiguity of union may arise when in a programming language the same operator has two different meanings. For example, in Pascal the operator "+" can indicate both integer addition and set-theoretic union

ambiguous grammar

$$\boxed{\begin{array}{l} E \to E + T \mid T \quad T \to V \quad V \to \ldots \\ E_{set} \to E_{set} + T_{set} \mid T_{set} \quad T_{set} \to V \end{array}}$$

EXAMPLE 3

ambiguous grammar

$$G: \quad S \to bS \mid cS \mid D \quad D \to bD \mid cD \mid \varepsilon$$

$$L(G) = L_S(G) \bigcup L_D(G)$$

$$L_S(G) = \{b, c\}^* = L_D(G)$$

$$S \overset{+}{\Rightarrow} bbcD \Rightarrow bbc \quad S \Rightarrow D \overset{+}{\Rightarrow} bbcD \Rightarrow bbc$$

$$\boxed{S \to bS \mid cS \mid \varepsilon}$$

non-ambiguous version

ambiguous grammar

EXAMPLE 4

$$S \to B \mid D \quad B \to bBc \mid \varepsilon$$

$$D \to dDe \mid \varepsilon$$

B generates $b^n c^n, n \geq 0$

D generates $d^n e^n, n \geq 0$

ε is the only ambiguous phrase

$$\boxed{\begin{aligned} &S \to B \mid D \mid \varepsilon \\ &B \to bBc \mid bc \\ &D \to dDe \mid de \end{aligned}}$$

non-ambiguous version

LANGUAGES THAT ARE INHERENTLY AMBIGUOUS

A language is said to be INHERENTLY AMBIGUOUS if every grammar that generates it is necessarily ambiguous, i.e., if all the equivalent grammars of the language happen to be ambiguous

EXAMPLE: here is a classical inherently ambiguous language

$$\boxed{L = \left\{ a^i b^j c^k \mid (i, j, k \geq 0) \wedge ((i = j) \vee (j = k)) \right\}}$$

$$\boxed{L = \left\{ a^i b^i c^* \mid i \geq 0 \right\} \bigcup \left\{ a^* b^i c^i \mid i \geq 0 \right\} = L_1 \bigcup L_2}$$

ambiguous
union

Whatever grammar G is designed to generate L, the phrases *ε*, *abc*, *$a^2b^2c^2$*, ... ALWAYS HAPPEN TO BE DERIVABLE IN TWO OR MORE WAYS. This behaviour is caused by the very nature of the language L and does not depend on the specific grammar G that generates L

AMBIGUITY OF CONCATENATION

Concatenating two languages may give rise to ambiguity if there exists a suffix of a phrase of the former language that is a prefix of a phrase of the latter language

$$\boxed{G = (\Sigma_1 \bigcup \Sigma_2, \left\{S\right\} \bigcup V_{N_1} \bigcup V_{N_2}, \left\{S \rightarrow S_1 S_2\right\} \bigcup P_1 \bigcup P_2, S)}$$

Assume that G_1 and G_2 are not ambiguous, then G is ambiguous if there exist two phrases x' $\in$ L_1 and x'' $\in$ L_2, and a non-empty string *v* such that

$$\boxed{\begin{array}{l} x' = u'v \wedge u' \in L_1 \quad x'' = vz'' \wedge z'' \in L_2 \\ u'vz'' \in L_1.L_2 \quad \text{and is ambiguous} \\ S \Rightarrow S_1 S_2 \overset{+}{\Rightarrow} u'S_2 \overset{+}{\Rightarrow} u'vz'' \\ S \Rightarrow S_1 S_2 \overset{+}{\Rightarrow} u'vS_2 \overset{+}{\Rightarrow} u'vz'' \end{array}}$$

EXAMPLE 1 – concatenation of Dyck languages

$$\Sigma_1 = \{a, a', b, b'\} \quad \Sigma_2 = \{b, b', c, c'\}$$

$$aa'bb'cc' \in L = L_1 L_2$$

$$G(L): \quad S \rightarrow S_1 S_2$$

$$S_1 \rightarrow aS_1a'S_1 \mid bS_1b'S_1 \mid \varepsilon$$

$$S_2 \rightarrow bS_2b'S_2 \mid cS_2c'S_2 \mid \varepsilon$$

$$\overbrace{aa'bb'}^{S_1}\overbrace{cc'}^{S_2} \qquad \overbrace{aa'}^{S_1}\overbrace{bb'cc'}^{S_2}$$

In order to eliminate ambiguity, it is necessary to exclude the case when a suffix moves from the former language to the latter one. A solution consists of inserting a separator between the two languages. Such a separator may not belong to the alphabets of the two languages. The concatenation language $L_1 \# L_2$ is generated by the additional axiomatic rule $S \rightarrow S_1 \# S_2$

EXAMPLE 2 – encoding – the relationship between ambiguity and the uniqueness of encoding in information theory

A message is a sequence of symbols out of the set Γ = { A, B,…, Z }, with possible repetitions. Such symbols are then encoded into strings of letters out of the set of terminal symbols Σ. Most frequently Σ is the binary alphabet, Σ = { 0, 1 }

$$\Gamma = \left\{ \overbrace{A}^{01}, \overbrace{C}^{10}, \overbrace{E}^{11}, \overbrace{R}^{001}, \right\}$$

$$ARRECA: 01\ 001\ 001\ 11\ 10\ 01$$

$$Testo \rightarrow ATesto \mid CTesto \mid ETesto \mid RTesto \mid A \mid C \mid E \mid R$$

$$A \rightarrow 01 \quad C \rightarrow 10 \quad \mathrm{E} \rightarrow 11 \quad \mathrm{R} \rightarrow 001$$

unambiguous grammar

$$\overbrace{\overbrace{01}^{A}\overbrace{10}^{\overset{Testo}{\overbrace{C}}}}^{Testo}$$

This grammar is not ambiguous: every message encoded onto Σ admits only one syntax tree, and therefore it can be decoded in a unique way

A bad choice of the set of encoding strings causes the grammar to be ambiguous

ambiguous grammar

$$\Gamma = \left\{ \overbrace{A}^{00}, \overbrace{C}^{01}, \overbrace{E}^{10}, \overbrace{R}^{010}, \right\}$$

$$Testo \rightarrow ATesto \,|\, CTesto \,|\, ETesto \,|\, RTesto \,|\, A \,|\, C \,|\, E \,|\, R$$

$$A \rightarrow 00 \quad C \rightarrow 01 \quad \mathrm{E} \rightarrow 10 \quad \mathrm{R} \rightarrow 010$$

$$ARRECA = 00\ 010\ 010\ 10\ 01\ 00$$

$$ACAEECA = 00\ 01\ 00\ 10\ 10\ 01\ 00$$

The *theory of codes* studies these and also other aspects, to identify conditions ensuring that a code (= a set of encoding strings) is decodable in a unique way

OTHER AMBIGUOUS SITUATIONS – regexps may be themselves ambiguous

ambiguous grammar

EXAMPLE 1

$$S \to DcD \quad D \to bD \mid cD \mid \varepsilon$$

$$\{b, c\}^* c \{b, c\}^*$$

Every phase that contains two or more *c* is ambiguous. To remove ambiguity, one can impose that the mandatory *c* is the leftmost one

unambiguous version

$$\boxed{S \to BcD \quad D \to bD \mid cD \mid \varepsilon \quad B \to bB \mid \varepsilon}$$

EXAMPLE 2 – Fixing the order of application of the rules – The rule to generate two *b* can be applied before or after the rule that generates one *b*

$$S \to bSc \mid bbSc \mid \varepsilon$$

$$S \Rightarrow bbSc \Rightarrow bbbScc \Rightarrow bbbcc$$

$$S \Rightarrow bSc \Rightarrow bbbScc \Rightarrow bbbcc$$

ambiguous grammar

$$\boxed{S \to bSc \mid D \quad D \to bbDc \mid \varepsilon}$$

unambiguous version

AMBIGUITY IN CONDITIONAL PHRASES

$$S \to if\ \ b\ then\ S\ else\ S \mid if\ \ b\ then\ S \mid a$$

$$\overbrace{if\ \ b\ then\ \overbrace{if\ \ b\ then\ a\ else\ a}}$$

$$\overbrace{if\ \ b\ then\ \ \overbrace{if\ \ b\ then\ a}\ \ else\ a}$$

so-called problem
of the
“dangling else”

$$S \to S_E \mid S_T \quad S_E \to if\ \ b\ then\ S_E\ \ else\ S_E \mid a$$
$$S_T \to if\ \ b\ then\ S_E\ \ else\ S_T \mid if\ \ b\ then\ S$$

Only S$_E$ can precede *else*

unambiguous version 1

$$S \to if\ \ b\ then\ S\ else\ S\ end\_if \mid$$
$$\mid if\ \ b\ then\ S\ end\_if \mid a$$

unambiguous version 2

Use the additional keyword *end_if* to mark the end of the conditional phrase