Context free grammars - II

Prof. A. Morzenti

SYNTAX TREES AND CANONICAL DERIVATIONS

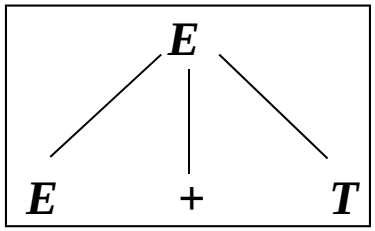
SYNTAX TREE: An oriented, sorted graph (children sorted from left to right) with no cycles, such that, for each pair of nodes, there is only one path connecting them

- it represents graphically the derivation process
- father-child relation / descendants / root node / leaf (or terminal) nodes
- degree of a node: number of its children
- root contains the axiom *S*
- frontier of the tree (leaf sequence from left to right) contains the generated phrase

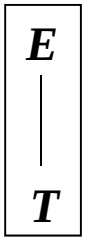
SUBTREE with root *N*: the tree having *N* as its root; it includes *N and* all its descendants

Espressions with sums and products: *E* expression, *T* term, *F* factor

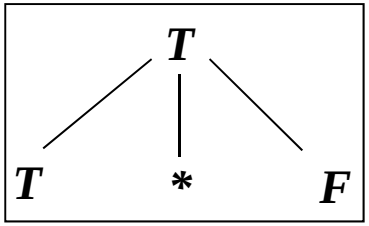
$1.\ E \rightarrow E + T$



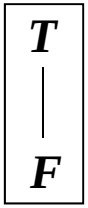
$2.\ E \rightarrow T$



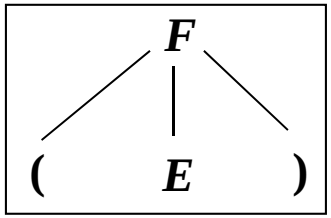
$3.\ T \rightarrow T * F$



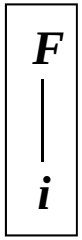
$4.\ T \rightarrow F$



$5.\ F \rightarrow (E)$

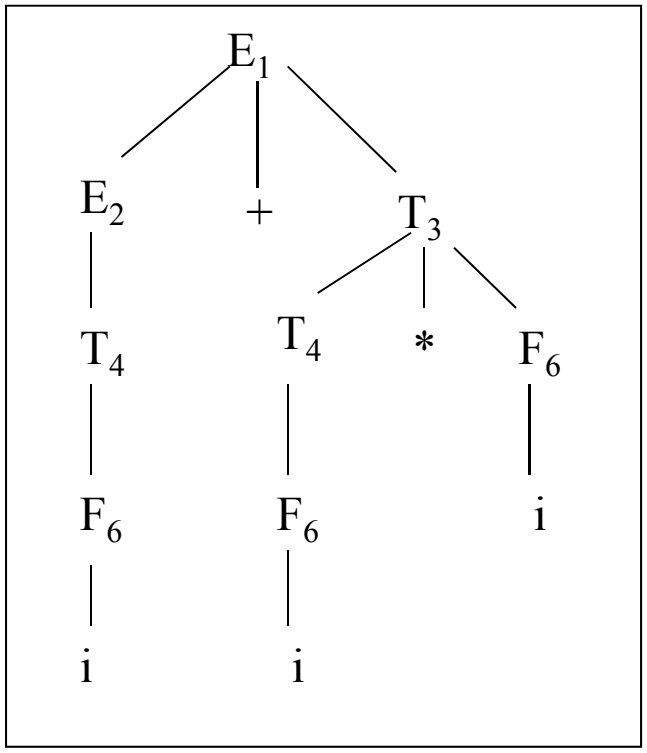


$6.\ F \rightarrow i$



Syntax tree for sentence i + i * i

(n.t. labelled with the applied rule)



Linear representation of the tree (subscripts indicate the n.t. in the subtree root)

$$[[[[i]_F]_T]_E + [[[i]_F]_T * [i]_F]_T]_E$$

LEFT DERIVATION
(numbers denote the rule, expanded nonterm. is underlined)

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

RIGHT DERIVATION (numbers denote the rule, expanded nonterm. is underlined)

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

Derivations may be neither right nor left

$$E \underset{l,r}{\Rightarrow} E + \underline{T} \underset{r}{\Rightarrow} \underline{E} + T * F \underset{l}{\Rightarrow} T + \underline{T} * F \Rightarrow T + F * \underline{F} \underset{r}{\Rightarrow} \underline{T} + F * i \underset{l}{\Rightarrow}$$
$$\underset{l}{\Rightarrow} F + \underline{F} * i \underset{r}{\Rightarrow} \underline{F} + i * i \underset{l,r}{\Rightarrow} i + i * i$$

However, for a **fixed** syntax tree of a sentence, there exist

a unique right derivation, and
a unique left derivation

matching that tree

Right and left derivation useful to define ***parsing*** (i.e., syntax analysis) algorithms

A different question: does a given sentence have a unique syntax tree?

This determines the ***ambiguity*** of the grammar

$$G = \left(\underbrace{\{E, T, F\}}_{non\ term.}, \underbrace{\{i, +, *,), (\}}_{term.}, \underbrace{P}_{productions}, \underbrace{E}_{axiom} \right)$$

$$P = \{E \to E + T \mid T, \quad T \to T * F \mid F, \quad F \to (E) \mid i\}$$

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

(apparently strange structure but necessary to model operator precedence, and avoid ambiguity)

Exercise: specify the lang. of arithmetic expr. using regular expressions
obviously not possible ;-)
recursion makes expr. similar to lists with unbounded nesting level

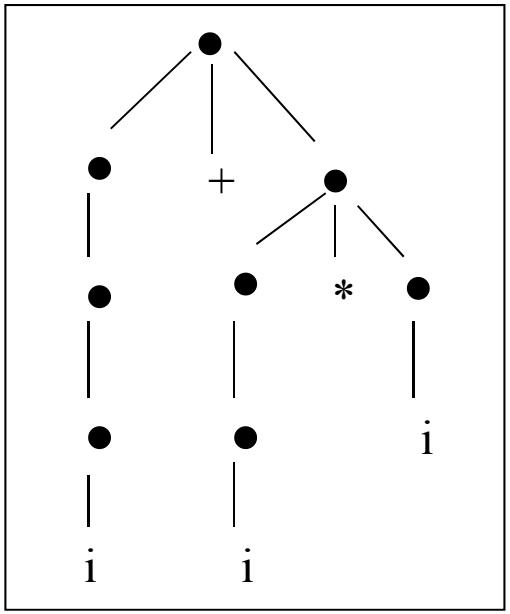
IMPORTANT: as an exercise, generate many strings, to understand how the grammar works, and check that operator precedence is necessarily respected

«Rule of thumb» for modeling precedence:
- nonterminals for ***low-precedence*** operators derived ***first***
- nonterminals for ***high-precedence*** operators derived ***later***

i.e., they are closer to of farther from the axiom in the *produce* relation

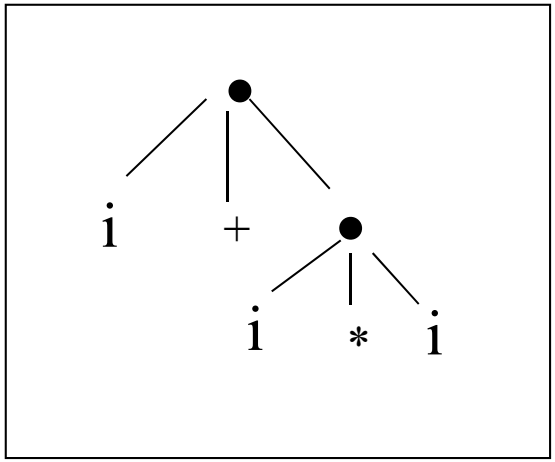
SKELETON TREE (only the frontier and the structure)

$$[[[[i]]]+[[[i]]*[i]]]$$



CONDENSED SKELETON TREE (internal nodes on a non-branching paths are merged)

$$[[i]+[[i]*[i]]]$$



PARENTHESIS LANGUAGES

structures with pairs of opening / closing marks
nested: inside a pair there can be other parenthesized structures (recursion)
(nested) structures can also be placed in sequences at the same level of nesting

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

Abstracting away from the type of parentheses, the paradigmatic language is

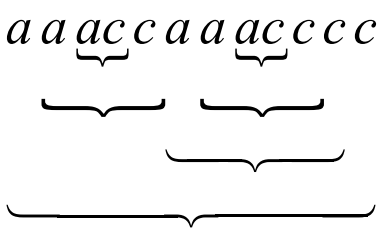
The Dyck language

Ex. alphabet: $\Sigma = \{')','(',']','['\}$

Sentence example: $()[[()[]]()]$

DYCK LANGUAGE with opening parenthesis *a,...,* closing parenthesis *c,...*
Grammar is surprisingly simple

$$\Sigma = \{a, c\}$$
$$S \to aScS \mid \varepsilon$$



The language is not linear (>1 nonterm. in the right part)

Exercise: build the syntax tree for the sentence
a a a c c a a a c c c

LINEAR NON REGULAR LANGUAGE:

$$L_1 = \{a^n c^n \mid n \geq 1\} = \{ac, aacc, \ldots\}$$
$$S \to aSc \mid ac$$

L_1 is a proper subset of the Dyck Language:
is does not admit many nested structures at the same level
(that is, strings of type ***acacac*** are ruled out)

REGULAR COMPOSITION OF FREE LANGUAGES

Applying the union, concatenation, Kleen star operations to free languages …
one obtains free languages, that is ...

The family of free languages is closed under union, concatenation, and star

$$G_1 = \left(\Sigma_1, V_{N_1}, P_1, S_1\right) \text{ and } G_2 = \left(\Sigma_2, V_{N_2}, P_2, S_2\right)$$

$$V_{N_1} \bigcap V_{N_2} = \varnothing \qquad S \notin (V_{N_1} \bigcup V_{N_2})$$

NB: one needs disjointed nonterm. sets and a new axiom

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

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

STAR: G for $(L_1)^*$ is obtained by adding to G_1 the rules $S \to SS_1 | \varepsilon$

CROSS '+' (not necessary because '+' is derived; this is added for simplicity):
G for $(L_1)^+$ is obtained by adding to G_1 the rules $S \to SS_1 | S_1$)

BTW: The ***mirror language*** of $L(G)$, $(L(G))^R$ is generated by the ***mirror grammar***, obtained by reversing the right part of the rules

EXAMPLE: Union of free languages

$$L = \{a^i b^j c^k \mid i = j \vee j = k\} = \{a^i b^i c^* \mid i \geq 0\} \cup \{a^* b^i c^i \mid i \geq 0\} = L_1 \cup L_2$$

L_1 and L_2 are generated by the two grammars G_1 and G_2

generates $a^i b^i c^*$

G_1

$S_1 \to XC$

$X \to aXb \mid \varepsilon$

$C \to cC \mid \varepsilon$

G_2

$S_2 \to AY$

$Y \to bYc \mid \varepsilon$

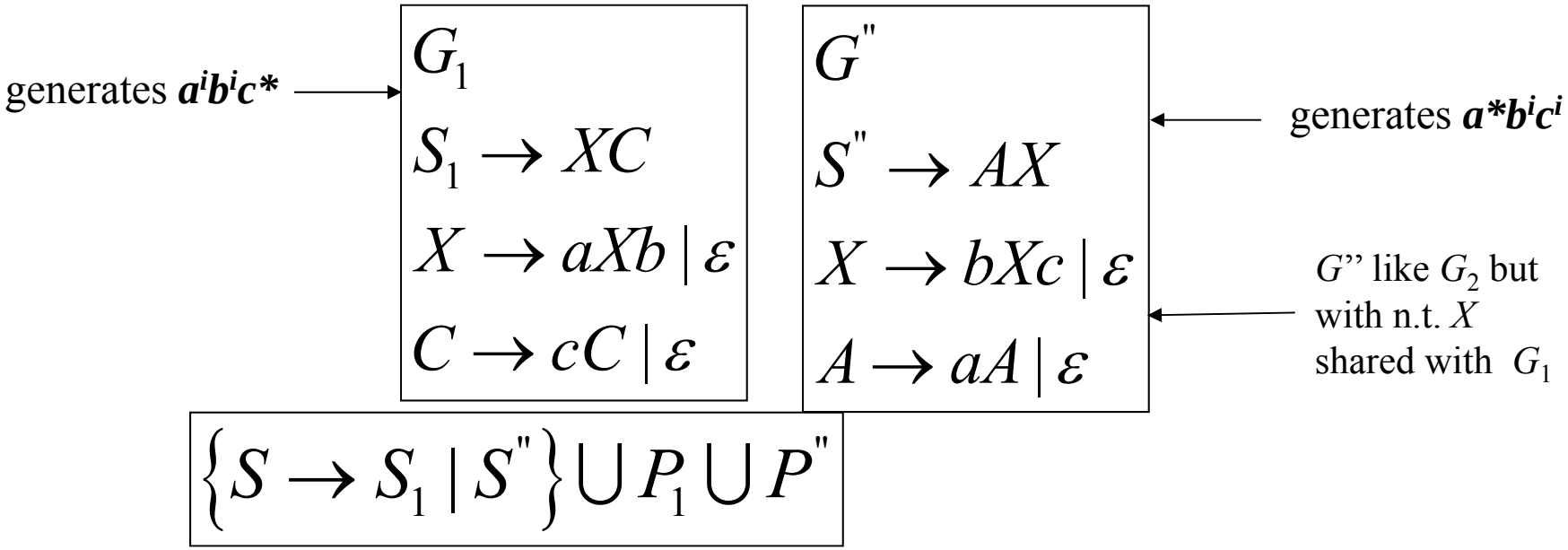
$A \to aA \mid \varepsilon$

generates $a^* b^i c^i$

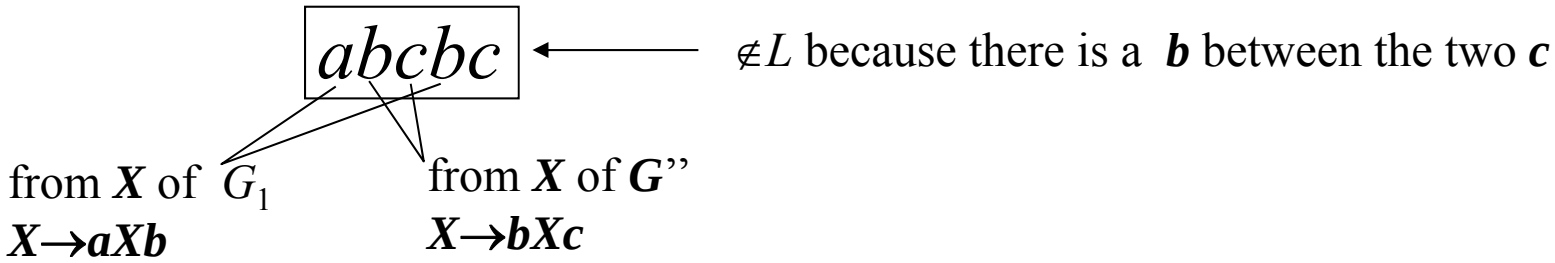
Notice that the nonterminal sets of grammars G_1 and G_2 are DISJOINTED

EXAMPLE: Union of free languages (follows)

What happens if the nonterm. sets are not disjointed? Let us use G" instead of G_2



If nonterm. are not disjoint, the grammar generates a **superset** of the union language:
spurious aditional sentences are generated



AMBIGUITY

Examples from natural language

“I saw the man with the binoculars.”
“La pesca è bella”
“half baked chicken”
“Questa è una rapina, dateci i soldi altrimenti ***spariamo***.” “OK, allora ***sparite***”...

In artificial, technical languages ambiguity must be ruled out (in the natural ones…).

We only consider SYNTACTIC AMBIGUITY:

A sentence *x* of a grammar *G* is ambiguous if it admits several distinct syntax trees

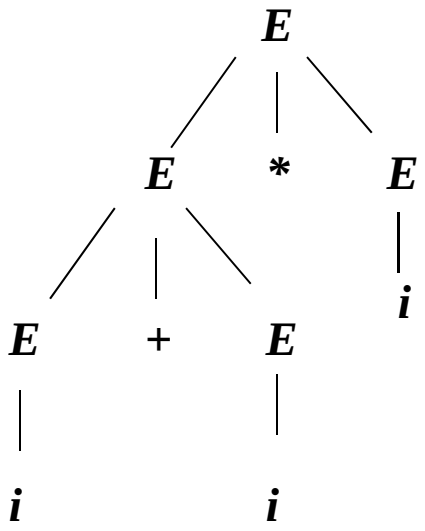
In such a case we say that the grammar *G* is ambiguous

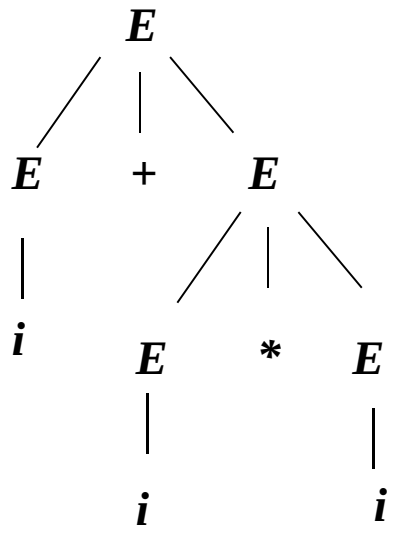
EXAMPLE: a "simple" grammar *G*' for arithmetic expressions

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

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

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





the sentence $i+i+i$ is also ambiguous

G' is ambiguous and **does not enforce** the usual **precedence of product over sum**

NB: *G*' is smaller than the previous *G* (p.3): it has only 1 n.t. and 4 rules
in grammar design there is ofted a **trade-off** between size and ambiguity

The ***degree of ambiguity of a sentence*** x of a language $L(G)$

is the number of distinct trees of x compatible with G

for a ***grammar*** the degree of ambiguity

is the maximum among the degree of ambiguity of its sentences

Notice that the degree of ambiguity of sentences of a grammar can be unlimited

IMPORTANT PROBLEM: determine if a grammar is ambiguous

The ***problem*** is ***undecidable***:
there does not exist a general algorithm that, given any free grammar,
terminates (after a finite number of steps) with the correct answer

The ***absence of ambiguity*** in a specific grammar can be shown

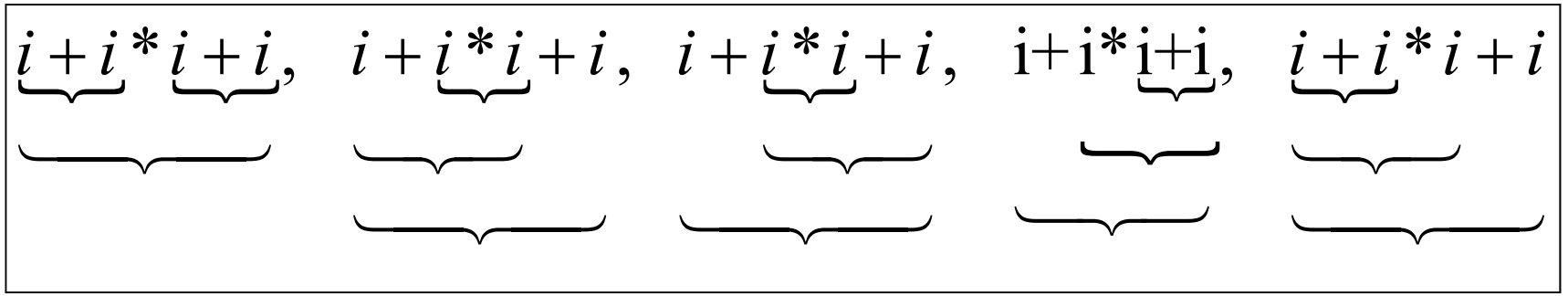
on a case by case basis, by hand, through ***inductive reasonings***,
hence by analyzing a finite number of cases

Instead, to show ambiguity one can exhibit a ***witness***: an ambiguous sentence

BEST APPROACH: AVOID AMBIGUITY IN THE GRAMMAR DESIGN PHASE

Let us reconsider the arithmetic expression example

The degree of ambiguity is 2 for $i + i + i$ and 5 for $i + i * i + i$



For longer sentences the degree of ambiguity increases with no limit

CATALOG OF AMBIGUOUS FORMS AND REMEDIES

1) AMBIGUITY FROM BILATERAL RECURSION: $A \rightarrow A \ \ldots \ A$

Example 1: grammar G_1 generates $i + i + i$ in two different ways (check!).

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

But $L(G_1) = i\ (+i)^*$ is a regular language: other, simpler grammars are possible

Nonambiguous right-recursive grammar : $\boldsymbol{E \rightarrow i + E \mid i}$

Nonambiguous left-recursive grammar : $\boldsymbol{E \rightarrow E + i \mid i}$

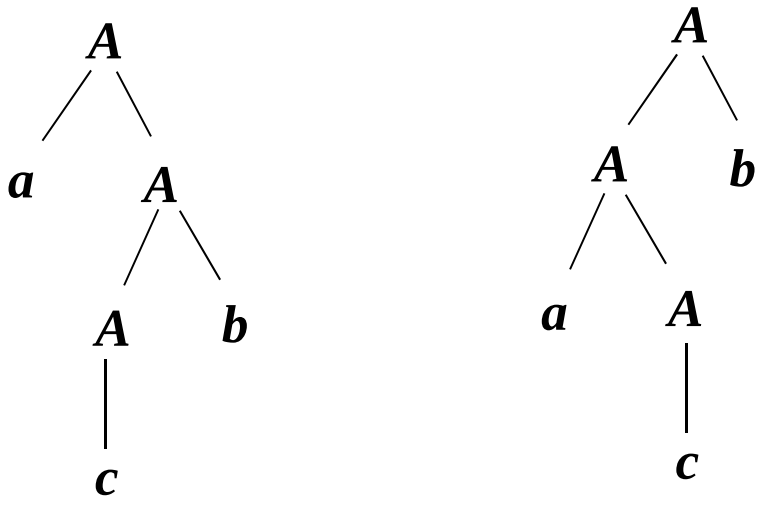
Example 2: Left and right recursion in different rules

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

G_2 admits derivations where the ***a*** and ***b*** in a given sentence are obtained in any order

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

Simplest example: two syntax trees for sentence *acb*



Example 2 (follows): Left and right recursion in different rules

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

$$S \to AcB$$

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

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

First method to eliminate ambiguity:
Generate the two lists by distinct rules

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

$$S \to aS \mid X$$

$$X \to Xb \mid c$$

Second method: force an ***order in the derivations***:
ex., first generate the *a*'s then the *b*'s

2) AMBIGUITY FROM LANGUAGE UNION

If L_1=$L(G_1)$ and L_2=$L(G_2)$ share some sentences (nonempty intersection)

The grammar G for the union language (slide n 10), is ambiguous (slide n.11)

A sentence $x \in L_1 \cap L_2$ admits two distinct derivations,

One with the rules of G_1 and the other with the rules of G_2

It is ambiguous for a grammar G that includes all the rules

Instead the sentences that belong to $L_1 \setminus L_2$ and to $L_2 \setminus L_1$ are nonambiguous

Remedy: provide disjointed set of rules for $L_1 \cap L_2$, $L_1 \setminus L_2$, and $L_2 \setminus L_1$

Examples in the textbook and in exam exercises

3) INHERENT AMBIGUITY

A language is INHERENTLY AMBIGUOUS if all its grammars are ambiguous

EXAMPLE:

$$L = \{a^i b^j c^k \mid i = j \vee j = k\} = \{a^i b^i c^* \mid i \geq 0\} \cup \{a^* b^i c^i \mid i \geq 0\} = L_1 \cup L_2$$

generates $\boldsymbol{a^i b^i c^*}$

G_1

$S_1 \rightarrow XC$

$X \rightarrow aXb \mid \varepsilon$

$C \rightarrow cC \mid \varepsilon$

G_2

$S_2 \rightarrow AY$

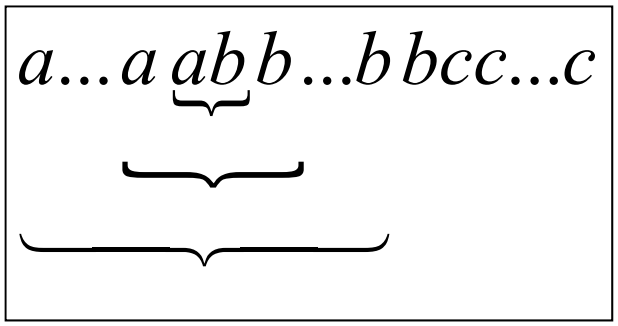
$Y \rightarrow bYc \mid \varepsilon$

$A \rightarrow aA \mid \varepsilon$

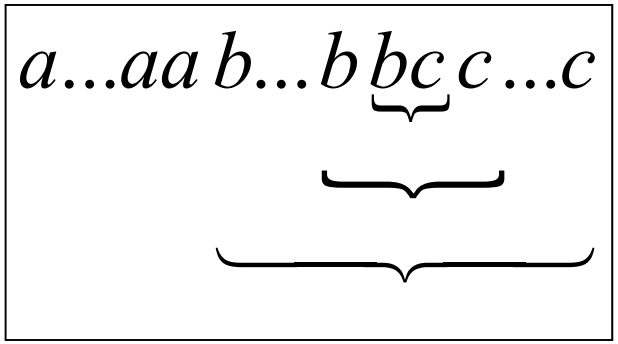
generates $\boldsymbol{a^* b^i c^i}$

The union grammar is ambiguous for the sentences ε, abc, $a^2b^2c^2$, ...
(which constitute a non-context free language)

They are generated by G_1 which has rules ensuring that $|x|_a = |x|_b$ …



… and also by G_2, with rules ensuring that $|x|_b=|x|_c$



We ***intuitively*** argue that ***any*** grammar for L is ambiguous

Luckily inherent ambiguity is rare and can be avoided in technical languages

4) AMBIGUITY FROM CONCATENATION OF LANGUAGES

when the suffix of some sentence in the first language
is also a prefix of a sentence in the second one

G for the concatenation of L_1 and L_2

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

Supposing G_1 and G_2 are not ambiguous, G is ambiguous if $\exists\ x' \in L_1$ and $x'' \in L_2$ such that there is a non empty string v such that

$$x' = u'v \wedge u' \in L_1 \quad x'' = vz'' \wedge z'' \in L_2$$

$$u'vz'' \in L_1.L_2 \quad \text{and is ambiguous:}$$

$$S \Rightarrow S_1 S_2 \overset{+}{\Rightarrow} u' S_2 \overset{+}{\Rightarrow} u'vz''$$

$$S \Rightarrow S_1 S_2 \overset{+}{\Rightarrow} u'vS_2 \overset{+}{\Rightarrow} u'vz''$$

Example – Ambiguity in the concatenation of Dyck languages

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

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

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

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

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

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

To prevent ambiguity one must avoid the shift of the substring from the suffix in the first language to the prefix in the second one

Easy solution: insert a ***new*** terminal symb. (e.g. '#') acting as a separator

***new*:** the symbol may not belong to any of the two alphabets

$L_1 \# L_2$ is generated by the «concatenation grammar» with the axiomatic rule $S \to S_1 \# S_2$.

The language is however modified

5) OTHER CASES OF AMBIGUITY

AMBIGUOUS REGULAR EXPRESSIONS

every sentence with two or more
c is ambiguous

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

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

REMEDY: identify the leftmost *c*

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

OTHER CASE: **LACK OF ORDER IN DERIVATIONS**

Example: Every phrase with
>2 *b*'s is ambiguous

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

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

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

REMEDY: Impose an order in the derivation
First the rule that generates balanced *b*'s and *c*'s
Then the one that generates eccess *b*'s

$$S \to bSc \mid D \quad D \to bbDc \mid \varepsilon$$

6) **AMBIGUITY IN CONDITIONAL PHRASES**: the (in)famous ***dangling else*** problem

$$S \rightarrow \overbrace{if\ b\ then\ S\ else\ S \mid if\ b\ then\ S}^{} \mid a$$

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

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

$$S \rightarrow S_E \mid S_T$$
$$S_E \rightarrow if\ b\ then\ S_E\ else\ S_E \mid a$$
$$S_T \rightarrow if\ b\ then\ S_E\ else\ S_T \mid if\ b\ then\ S$$

REMEDY 1 Rule out this interpretation (i.e., use the ***closest match interpretation***);
two n.t., S_E and S_T:
S_E always has the ***else*** (cannot have only the ***then***),
only for S_T it is possibile not to have the ***else***
therefore only S_E can precede the ***else***

$$S \rightarrow if\ b\ then\ S\ else\ S\ endif \mid if\ b\ then\ S\ endif \mid a$$

REMEDY 2 Modify the language,
introduce a closing mark ***endif***