Context free grammars - I

Prof. A. Morzenti

NB: many parts are well known from previous courses
I will run quickly through them and discuss more deeply the really new ones

LIMITS OF REGULAR LANGUAGES

Simple languages such as $L = \{ a^n b^n \mid n > 0 \}$

representing basic syntactic structures like

begin begin ... begin ... end ... end end

are ***not*** regular

e.g., $b^+ e^+$ does not satisfy the constraint $\#b = \#e$

$(be)^+$ does not ensure nesting

GRAMMARS – a more powerful means to define languages

through ***rewriting rules***

language phrases generated through repeated application of rules

The grammar is characterized by its set of rules

Example – Language of ***palindromes***

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

a palindrome is…

$P \to \varepsilon$	an empty palindrome
$P \to a\ P\ a$	a palindrome surrounded by two *a*'s
$P \to b\ P\ b$	a palindrome surrounded by two *b*'s

A chain of ***derivation steps***:

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

look out: distinguish the two *metasymbols*

$\to$ separates the left and right part of a rule

$\Rightarrow$ derivation ***relation*** (rewriting)

Example: a non-empty ***list of*** palindromes, ex: *abba bbaabb aa*

$L \to P\ L$

$L \to P$

$P \to \varepsilon \quad P \to a\ P\ a \qquad P \to b\ P\ b$

non terminal symbols:

- L (axiom, or start symbol)
- P (defines the component palindrome substrings)

derivation of the string *abba bbaabb aa*
(spaces added for readability, nonterm to be expanded underlined)

$L \Rightarrow P\ \underline{L} \Rightarrow P\ P\ \underline{L} \Rightarrow \underline{P}\ P\ P \Rightarrow a\ \underline{P}\ a\ P\ P$

$\Rightarrow ab\ \underline{P}\ ba\ P\ P \Rightarrow abba\ \underline{P}\ P \Rightarrow abba\ b\ \underline{P}\ b\ P$

$\Rightarrow abba\ bb\ \underline{P}\ bb\ P \Rightarrow abba\ bba\ \underline{P}\ abb\ P$

$\Rightarrow abba\ bbaabb\ \underline{P}$

$\Rightarrow abba\ bbaabb\ a\ \underline{P}\ a \Rightarrow abba\ bbaabb\ aa$

CONTEXT-FREE GRAMMAR
(BNF - Backus Normal Form – TYPE 2 – FREE GRAMMAR)

defined by four entities

1. *V, **non terminal alphabet,*** is the set of nonterminal symbols
2. *Σ, **terminal alphabet,*** is the set of the symbols of which phrases/sentences are made
3. *P,* is the set of ***rules*** or ***productions***
4. *S* ∈ *V,* is the specific nonterminal, called the ***axiom*** (***Start***), from which derivations start

a rule is written as $X \rightarrow \alpha$, with $X \in V$ and $\alpha \in (V \cup \Sigma)^*$

rules with the same nonterminal X: $X \rightarrow \alpha_1$, $X \rightarrow \alpha_2$, … $X \rightarrow \alpha_n$

can be written in brief as $X \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$ (or $X \rightarrow \alpha_1 \cup \alpha_2 \cup \ldots \cup \alpha_n$)

$\alpha_1, \alpha_2, \ldots, \alpha_n$ are called the ***alternatives*** of X

To avoid confusion,

the metasymbols '→', '|', '∪', 'ε' cannot be terminal symbols,

terminal and nonterminal alphabets must be disjointed

NOTATIONS to distinguish terminal and nonterminal symbols

- angle brackets:

  <if-phrase> $\rightarrow$ *if <cond> then <if-phrase> else <if-phrase>*

- ***bold italic***:

  if-phrase $\rightarrow$ ***if*** *cond* ***then*** *if-phrase* ***else*** *if-phrase*

- quotation marks ‘ ’

  if-phrase $\rightarrow$ *‘if’ cond ‘then’ if-phrase ‘else’ if-phrase*

- upper- versus lower-case:

  $F \rightarrow$ *if* C *then* D *else* D

WE USUALLY ADOPT THESE CONVENTIONS:

- terminal characters - $\{a, b, \ldots\}$
- nonterminal characters - $\{A, B, \ldots\}$
- strings $\in \Sigma^*$ (only terminals) - $\{r, s, \ldots, z\}$
- strings $\in (V \cup \Sigma)^*$ (terminals and nonterminals) - $\{\alpha, \beta, \ldots\}$
- strings $\in V^*$ (only **nonterminals**) - σ

TYPES OF RULES (RP = right part, LP = left part)

Terminal: RP contains only terminals, or the empty string	$\to u \mid \varepsilon$
Empty (*or* ***null***): RP is empty	$\to \varepsilon$
Initial / Axiomatic: LP is the axiom	$S \to$
Recursive: LP occurs in RP	$A \to \alpha A \beta$
Left-recursive: LP is prefix of RP	$A \to A \beta$
Right-recursive: LP is suffix of RP	$A \to \alpha A$
Left- *and* ***right-recursive***: conjunction of two previous cases	$A \to A \beta A$
Copy *or* ***categorization***: RP is a single nonterminal	$A \to B$
Linear: at most one nonterminal in RP	$\to u B v \mid w$
Right-linear (type 3): linear + nonterminal is suffix	$\to u B \mid w$
Left-linear (type 3): linear + nonterminal is prefix	$\to B v \mid w$
Homogeneous normal: *n* nonterminals or just one terminal	$\to A_1 \dots A_n \mid a$
Chomsky normal (or homogeneous of degree 2): two nonterminals or just one terminal	$\to BC \mid a$
Greibach normal: one terminal possibly followed by nonterminals	$\to a \sigma \mid b$
Operator normal: two nonterminals separated by a terminal (operator); more generally, strings devoid of adjacent nonterminals	$\to A a B$

DERIVATIONS AND GENERATED LANGUAGE

Def. of ***derivation relation*** **'⇒'**

for $\delta, \eta \in (V \cup \Sigma)^*$, $A \to \alpha$ rule of G

$\delta A \eta$ derives $\delta \alpha \eta$ for grammar G, $\delta A \eta \underset{G}{\Rightarrow} \delta \alpha \eta$, or $\delta A \eta \Rightarrow \delta \alpha \eta$

the rule $A \to \alpha$ is applied in that *derivation step*, and α ***reduces to*** A

power, reflexive and transitive closure of '⇒'

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

If $A \overset{*}{\Rightarrow} \alpha$ $\alpha \in (V \cup \Sigma)^*$ called ***string form generated by G***

If $S \overset{*}{\Rightarrow} \alpha$ α called ***sentential*** or ***phrase form***

If $S \overset{*}{\Rightarrow} s$ $s \in \Sigma^*$, s is called ***phrase*** or ***sentence***

LANGUAGE GENERATED FROM NONTERMINAL ***A*** OR FROM AXIOM ***S***

$$\boxed{\begin{array}{l} L_A(G) = \left\{ x \in \Sigma^* \mid A \overset{+}{\Rightarrow} x \right\} \\ L(G) = L_S(G) = \left\{ x \in \Sigma^* \mid S \overset{+}{\Rightarrow} x \right\} \end{array}}$$

Example: Grammar G_l generates the structure of a book: it contains

- a front page (f)
- a series (denoted by the nonterm. A) of one or more chapters by means of a *left recursion*
- a chapter has a title t and a sequence B (right recursion) of one or more lines l

$$\boxed{\begin{aligned} &S \to fA \\ &A \to AtB \mid tB \\ &B \to lB \mid l \end{aligned}}$$

from A one generates the string form $tBtB$ and the phrase $tlltl \in L_A(G_l)$

from S one generates the phrase forms $fAtlB$, $ftBtB$

The language generates from B is $L_B(G_l) = l^+$

$L(G_l)$, being generated by the context free grammar G_l, is ***context free*** or ***free***

Notice: it is also regular, because it is defined also by the regular expressione $f(tl^+)^+$

Two grammars G and G' are equivalent if they generate the same language, that is, $L(G) = L(G')$

G_l

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

G_{l2}

$$S \to fA$$
$$A \to AtB \mid tB$$
$$B \to Bl \mid l$$

The first rules have the same n.t., only renamed

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

by means of *right* or *left* recursion

ERRONEOUS GRAMMARS AND USELESS RULES

A grammar G is ***clean*** (or ***reduced***) iff for every nonterminal A :

1. A is ***reachable*** from the axiom S, and hence contributes to the generation of the language; that is, there exists a derivation

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

2. A is ***defined***, that is, it generates a non-empty language (we are not interested in the language $\emptyset$)

$$L_A(G) \neq \emptyset$$

NB: $L_A(G) = \emptyset$ includes also the case when no derivation from A terminates with a terminal string s (i.e. $s \in \Sigma^*$), e.g.: $P = \{ S \to aA,\ A \to bS \}$

GRAMMAR CLEANING: two steps algorithm:

The FIRST PHASE builds the set ***UNDEF*** of undefined nonterminals

The SECOND PHASE builds the set of unreachable nonterminals

PHASE 1- We first build the ***complement*** set $DEF = V \setminus UNDEF$

DEF is ***initialized*** from the *terminal rules* (the n.t. that immediately generate a terminal string)

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

The following ***update*** is repeated until a ***fixed point*** is reached :

$$DEF := DEF \cup \{\ B \mid (B \to D_1 D_2 \ldots D_n) \in P \wedge \forall i\ (D_i \in DEF \cup \Sigma)\}$$

Every D_i is already in *DEF* or it is a terminal

In algebra, the ***fixed point*** of a transformation is an object that is transformed into itself

At each iteration, two cases can occur:

1. New nonterm. are found having the RP all with defined nonterm. or term., or
2. No new nonterm. is found, algorithm terminates (a ***fixed point*** has been reached)

nonterminals $\in UNDEF = V \setminus DEF$ are eliminated

PHASE 2 – Consider the ***produce*** relation, defined as

$$\boxed{A \text{ produce } B \quad \text{iff} \quad \left(A \to \alpha B \beta\right) \in P, \text{ with } A \neq B \quad \alpha, \beta \text{ any string}}$$

C is ***reachable*** from ***S*** iff there exists, in the graph of the produce relation, a path from ***S*** to ***C***

nonterminals that are not reachable can be eliminated often another requirement is added for cleanness condition of a grammar ***G*** :

3. ***G*** must not allow for ***circular derivations***: they are not essential and introduce ***ambiguity***

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

if the derivation $A \overset{+}{\Rightarrow} x$ is possible

then also $A \overset{+}{\Rightarrow} A \overset{+}{\Rightarrow} x$ and many other similar ones exist

NB: ***circular derivations*** must not be confused with ***recursive rules*** and ***derivations*** **!!**

EXAMPLES OF GRAMMARS THAT ARE NOT CLEAN

1) $\{ S \to aASb, A \to b \}$ (S does not generate any phrase, i.e., $L(S)=\varnothing$)

2) $\{ S \to a, A \to b \}$ (A not reachable) ($\{ S \to a\}$ equiv. clean version)

3) $\{ S \to aASb \mid A, A \to S \mid b \}$ (circular on S and A) ($\{S \to aSSb \mid b \}$ equiv. clean)

circularity can also derive from an empty rule

$$\boxed{X \to XY \mid \ldots \quad Y \to \varepsilon \mid \ldots}$$

NB: even if clean, a grammar can have ***redundant rules*** (leading to ambiguity)

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

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

RECURSION AND LANGUAGE ***INFINITY***

most interesting languages are infinite

but what determines the ability of a grammar to generate an infinite language?

infinity of the language implies unbounded phrase length

therefore the grammar must be recursive

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

if $n = 1$ it is ***immediately recursive***

A is a ***recursive nonterminal***

if $x = \varepsilon$ then it is ***left recursive*** (l.r. derivation, l.r. nonterminal)

if $y = \varepsilon$ then it is ***right recursive*** (r.r. derivation, r.r. nonterminal)

NB: circularity and recursiveness are (very) different notions

a grammar may be recursive (admit recursive derivations) but not circular

circular $\Rightarrow$ recursive but it is **not** true that recursive $\Rightarrow$ circular

necessary and sufficient condition for language $L(G)$ to be infinite,

assuming G clean and devoid of circular derivations,

is that G allows for recursive derivations

necessary condition: if no recursive derivation was possible,
then every derivation would have limited length hence $L(G)$ would be finite

sufficient condition:

$A \overset{n}{\Rightarrow} xAy$ implies $A \overset{+}{\Rightarrow} x^m A y^m$
for any $m \geq 1$ with $x, y \in \Sigma^*$ not both empty (because grammar is not circular)
Furthermore G clean implies
$S \overset{*}{\Rightarrow} uAv$ (A reachable from S)
and $A \overset{+}{\Rightarrow} w$ (derivation from A terminates successfully)

therefore there exist nonterminals that generate an infinite language

$$S \overset{*}{\Rightarrow} uAv \overset{+}{\Rightarrow} ux^m A y^m v \overset{+}{\Rightarrow} ux^m w y^m v, (\forall m \geq 1)$$

a grammar does not have recursive derivations

$\Leftrightarrow$ (if and only if)

the graph of the ***produce*** relation has no circuits

Example

$$\boxed{\begin{array}{l} S \to aBc \\ B \to ab \mid Ca \\ C \to c \end{array}}$$

finite language: { *aabc*, *acac* }

$$S \longrightarrow B \longrightarrow C$$

Example (arithmetic expressions)

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

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

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

F (factor) has indirect recursion (non immediate)

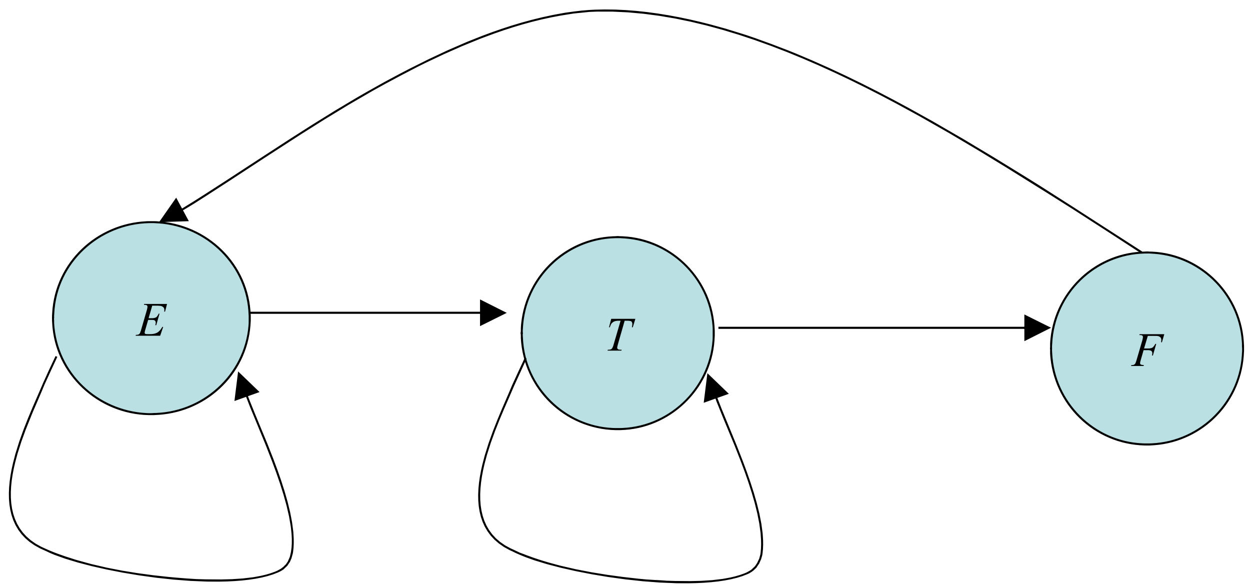
E (expression) has immediate left recursion and non immediate recursion

T (term) has immediate left recursion and non immediate recursion

G is clean, recursive, noncircular, hence the generated language is infinite grammar has recursions

$\Leftrightarrow$

the graph of the produce relation has circuits



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

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

SYNTAX TREES AND CANONICAL DERIVATIONS

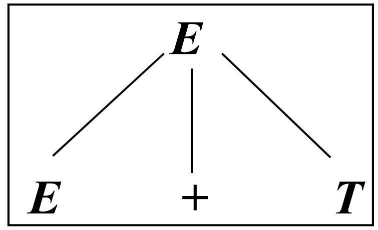
SYNTAX TREE: An oriented, sorted graph (children sorted from left to right) with no cycles, such that, for each pair of nodes, there is only one path connecting them

- 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

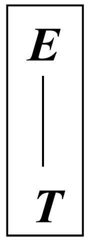
SUBTREE with root N: the tree having N as its root; it includes N *and* all its descendants

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

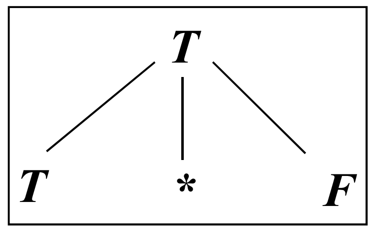
$$1.\ E \to E + T$$



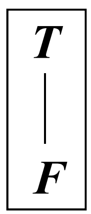
$$2.\ E \to T$$



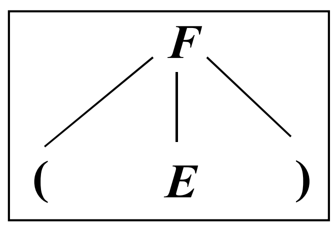
$$3.\ T \to T * F$$



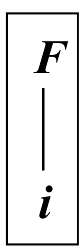
$$4.\ T \to F$$



$$5.\ F \to (\ E\ )$$



$$6.\ F \to i$$



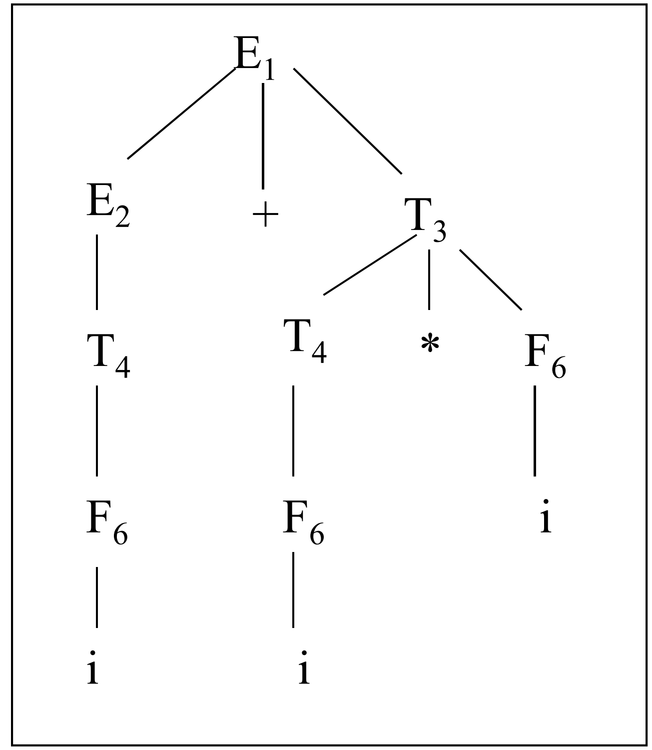
every rule application

generates a tree fragment

father + its children

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)

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

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 are 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

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

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

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

(this rule structure is necessary to model precedence, associativity, and avoid ambiguity)

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

VERY IMPORTANT

«Rule of thumb» for modeling mutual precedence of operators:

- terminals for ***low-precedence*** operators derived ***first*** Ex: ‘+’
- terminals for ***high-precedence*** operators derived ***later*** Ex: ‘*’

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

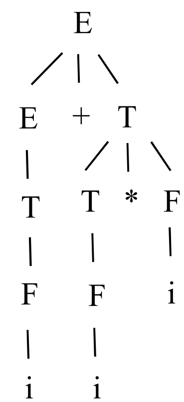
«Rule of thumb» for modeling associativity of operators:

- left-recursive rule $\Rightarrow$ left-associative operator Ex: ‘+’ and ‘*’ are left assoc.
- right-recursive rule $\Rightarrow$ right-associative operator

With P = { E → E + T | T, T → T * F | F, F → (E) | i }

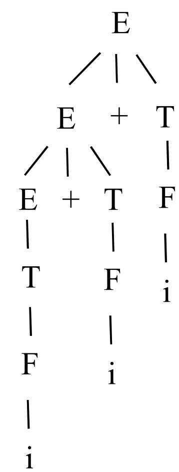
i* + *i* * *i has syntax tree

‘*’ higher precedence than ‘+’



i* + *i* + *i has syntax tree

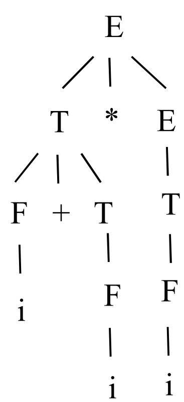
‘+’ left associative



With P = { E → T * E | T, T → F + T | F, F → (E) | i }

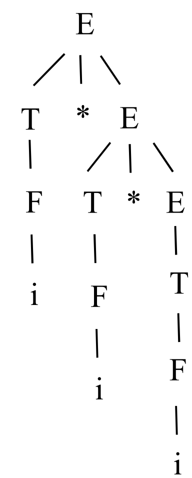
i* + *i* * *i has syntax tree

‘+’ higher precedence than ‘*’



i* * *i* * *i has syntax tree

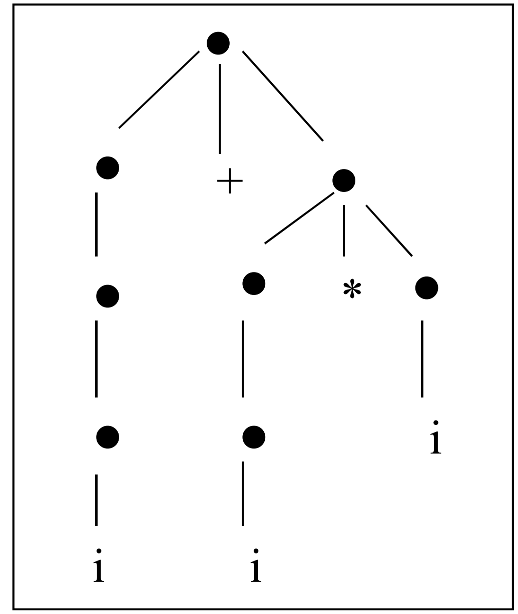
‘*’ right associative



SKELETON TREE (only the frontier and the structure)

linear representation

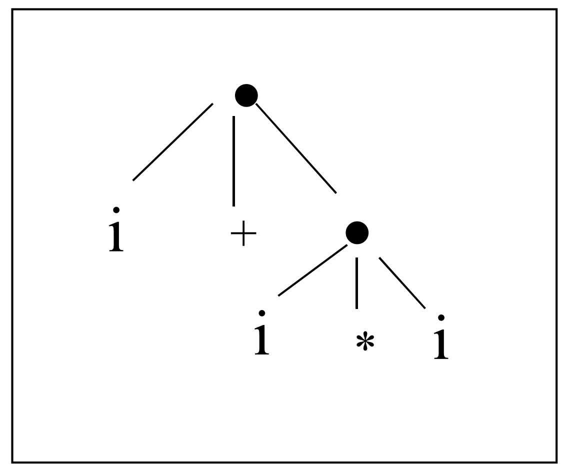
$$[[[[i]]]+[[[i]]*[i]]]$$



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

linear representation

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



STRONG (STRUCTURAL) AND WEAK EQUIVALENCE

WEAK EQUIVALENCE: two grammars G and G' are weakly equivalent if they generate the same language: $L(G) = L(G')$.

G and G' might assign different structures (syntax trees) to the same sentence

the structure assigned to a sentence is important:
it is used by translators and interpreters

STRONG or STRUCTURAL EQUIVALENCE of two grammars G and G'
$L(G) = L(G')$ (weak eq.) **and**
G and G' have the same ***condensed skeleton trees***

strong eq. $\rightarrow$ weak eq. **but** strong eq. $\neq$ weak eq.

(hence $\neg$(weak eq. $\rightarrow$ strong eq.))

strong eq. is **decidable**

weak eq. is **not decidable**

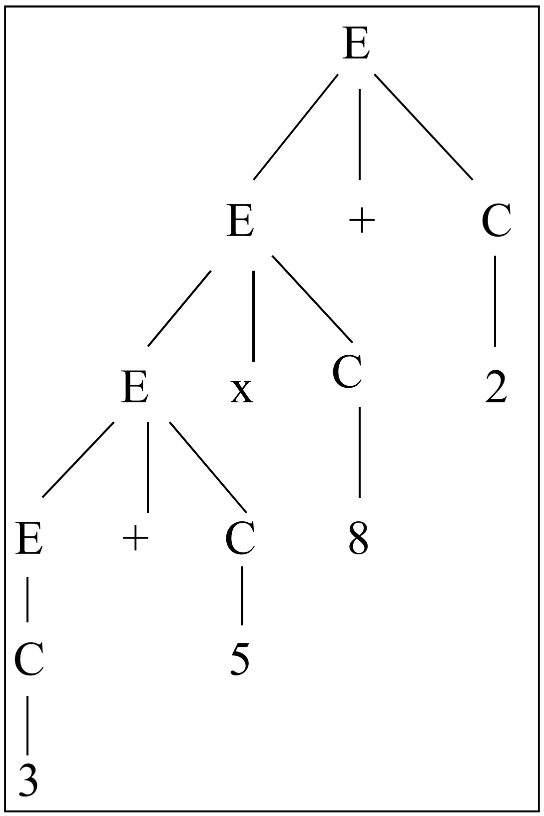
(it can happen that one can establish that G_1 and G_2 **are not** strongly equivalent

but is unable to establish whether they are weakly equivalent or not)

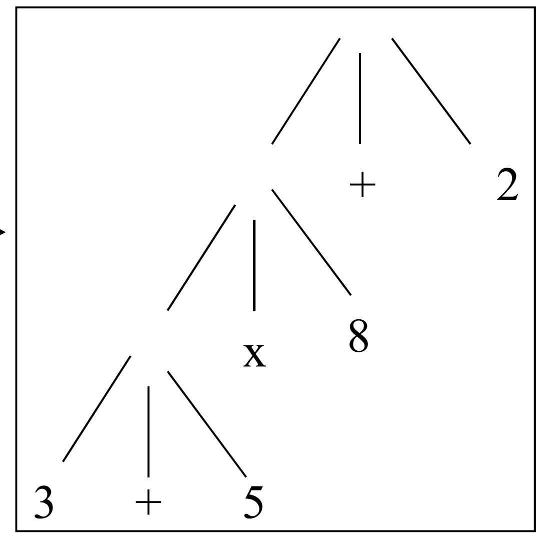
Example: Structural equivalence of arithmetic expressions: $3 + 5 \times 8 + 2$

$$\boxed{\begin{array}{l} G_1:\quad E \rightarrow E + C \quad E \rightarrow E \times C \quad E \rightarrow C \\ C \rightarrow 0\,|\,1\,|\,2\,|\,3\,|\,4\,|\,5\,|\,6\,|\,7\,|\,8\,|\,9 \end{array}}$$

NB: the grammar has ***left recursion*** hence the operators are ***left-associative***



condensed skeleton

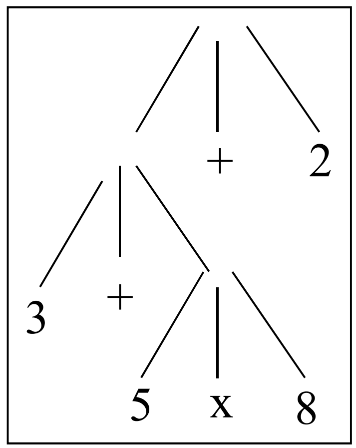


Semantic interpretation of G_1: (((3 + 5) x 8) + 2)

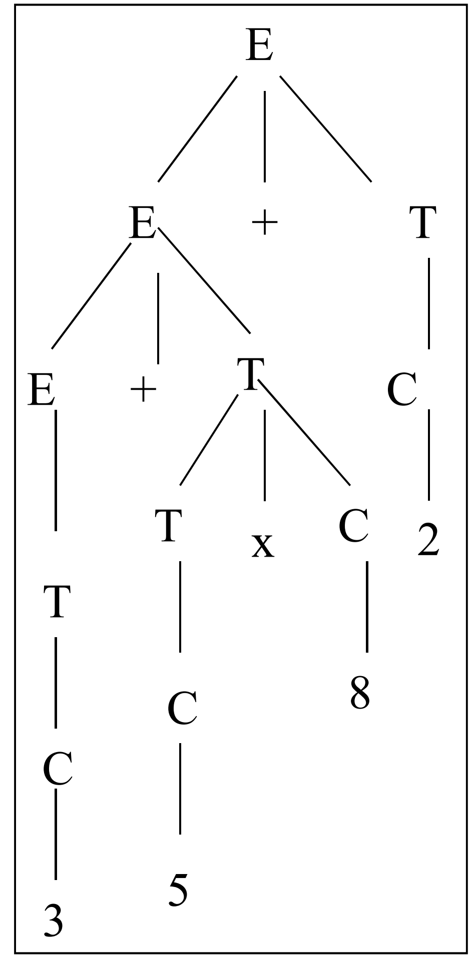
ANOTHER GRAMMAR FOR THE SAME LANGUAGE

NB: copy (or categorization) rules

$$G_2: \quad E \to E + T \quad E \to T \quad T \to T \times C \quad T \to C$$
$$C \to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$



condensed skeleton



G_1 and G_2 are not structurally equivalent

Semantic interpretations: G_1: (((3 + 5) x 8) + 2)
G_2: ((3 + (5 x 8)) + 2)

Only G_2 is **structurally adequate** w.r.t. the operator precedence rules (NB: G_2 is more complex): it forces the generation of a product from n.t. T only ***after*** E therefore it assigns a higher precedence to the product than to the sum

NB: **this is how operator priority can be enforced through grammars**

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: $\boxed{\Sigma = \{')','(',']','['\}}$ Sentence example: $\boxed{()[[()[]]()]}$

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

$$\Sigma = \{a, c\}$$

$$S \rightarrow aScS \mid \varepsilon$$

$$\underbrace{a\,\underbrace{a\,\underbrace{ac}\,c}\,\underbrace{a\,\underbrace{a\,\underbrace{ac}\,c}\,c}\,c}$$

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 c

LINEAR NON REGULAR LANGUAGE:

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

$$S \rightarrow aSc \mid ac$$

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, therefore ...

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

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

$$V_{N_1} \cap V_{N_2} = \varnothing \qquad S \notin (V_{N_1} \cup V_{N_2})$$

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

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

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

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 $S \to SS_1 | S_1$)

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

$$\boxed{\begin{array}{l} G_1 \\ S_1 \to XC \\ X \to aXb \mid \varepsilon \\ C \to cC \mid \varepsilon \end{array}} \quad \boxed{\begin{array}{l} G_2 \\ S_2 \to AY \\ Y \to bYc \mid \varepsilon \\ A \to aA \mid \varepsilon \end{array}}$$

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

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

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

$$G_1$$
$$S_1 \to XC$$
$$X \to aXb \mid \varepsilon$$
$$C \to cC \mid \varepsilon$$

$$G''$$
$$S'' \to AX$$
$$X \to bXc \mid \varepsilon$$
$$A \to aA \mid \varepsilon$$

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

G'' like G_2 but with n.t. X shared with G_1

$$\boxed{\{S \to S_1 \mid S''\} \cup P_1 \cup P''}$$

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

$\boxed{abcbc}$ $\notin L$ because there is a ***b*** between the two ***c***

from ***X*** of G_1
$\boldsymbol{X \to aXb}$

from ***X*** of ***G''***
$\boldsymbol{X \to bXc}$