Context Free Grammars - IV

Prof. A. Morzenti

FREE GRAMMARS EXTENDED WITH REGULAR EXPRESSIONS
(Extended BNF or EBNF or Regular Right Part Grammars-RPPG)

MORE READABLE thanks to the star and cross (iteration) and choice (union) operators

EBNF's allow for the definition of SYNTAX DIAGRAMS which can be viewed as a blueprint of the of the syntax analyzer flowchart

NB: The ***CF*** family is closed under all regular operations, therefore the generative power of EBNF is the same as that of BNF

EBNF GRAMMAR $G = \{ V, \Sigma, P, S \}$
Exactly $|V|$ rules, each in the form $A \rightarrow \eta$, with η r.e. over alphabet $V \cup \Sigma$

Example: Algol-like Language
B: block; D: declaration; I=Imperative part;
F=phrase; a=assignment; c=char; i=int;
r=real; v=variable; b=begin; e=end

$$B \rightarrow b[D]Ie$$
$$D \rightarrow ((c \mid i \mid r)v(,v)^*;)^+$$
$$I \rightarrow F(;F)^*$$
$$F \rightarrow a \mid B$$

Example of a Declaration section

char text1, text2; *real* temp, result;
int alpha, beta2, gamma;

Alternative BNF grammar for **D** : $D \rightarrow DE \mid E \quad E \rightarrow AF; \quad A \rightarrow c \mid i \mid r \quad F \rightarrow v, F \mid v$

The BNF grammar is longer and obviously less readable
it conceals the existence of two hierarchically nested lists
Furthermore, the choice of nonterminal symbol names (A, E, F) can be arbitrary

DERIVATIONS AND TREES IN EXTENDED FREE GRAMMARS

Derivation in EBNF G defined by considering an equivalent BNF G' with infinite rules

G includes $A \to (aB)^+$

G' includes $A \to aB \mid aBaB \mid aBaBaB \mid ....$

DERIVATION RELATION FOR EBNF G:

Given strings η_1 and $\eta_2 \in (\Sigma \bigcup V)^*$

η_2 is said to be *derived* immediately in G from η_1

$\eta_1 \Rightarrow \eta_2$, if the two strings can be factorized as:

$\eta_1 = \alpha A \gamma$, $\eta_2 = \alpha \vartheta \gamma$ and there exists a rule

$A \to e$ such that the r.e. e admits the derivation $e \overset{*}{\Rightarrow} \vartheta$

Notice that η_1 and η_2 do not contain

r.e. operators nor parenthesis.

Only string e is a r.e. but it does not appear in the derivation if it is not terminal

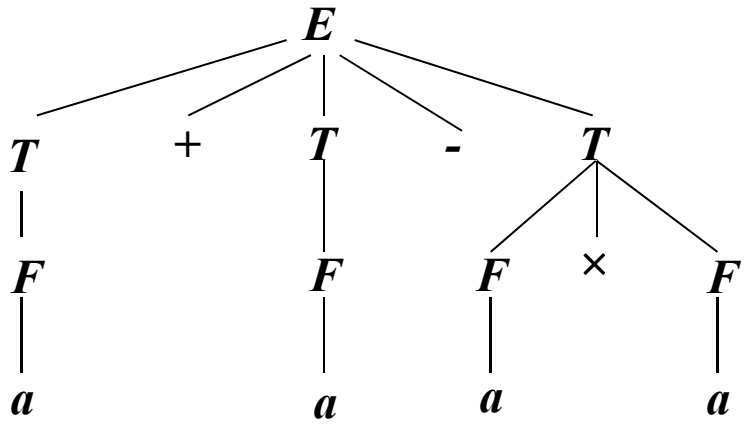
Example: Extended derivation for arithmetic expressions
Extended grammar for arithmetic expressions with four infix operators, parentheses and variable *a*.

$E \to [\,+\,|\,-\,]\; T\,(\,(\,+\,|\,-\,)\; T\,)^*$ $\quad$ $T \to F\,(\,(\,\times\,|\,/\,)\; F\,)^*$ $\quad$ $F \to\; a \;|\; \text{‘(’}\; E\; \text{‘)’}$

left derivation:

$$E \Rightarrow T+T-T \Rightarrow F+T-T \Rightarrow a+T-T \Rightarrow a+F-T \Rightarrow$$

$$\Rightarrow a+a-T \Rightarrow a+a-F\times F \Rightarrow a+a-a\times F \Rightarrow a+a-a\times a$$



unbounded node degree

the tree is in general **wider** …

… and **reduced in depth**

GRAMMARS OF REGULAR LANGUAGES

Regular languages are a special case of free languages

(reminder: we use the term «free» as an abbreviation for «context-free»)

They are generated by grammars with strong constraints on the form of rules

Due to these constraints the sentences of regular languages present «inevitable» repetitions

FROM REGULAR EXPRESSIONS TO CONTEXT-FREE GRAMMARS

Define the form of the rules depending on that of the regular expression

Notice that recursive rules match iterative operators (star '*' and cross '+')

REGULAR EXPRESSION	**FORM OF E IN RULE $R \rightarrow E$**
1. $r = r_1.r_2....r_k$	1. $E = E_1E_2...E_k$
2. $r = r_1 \cup r_2 \cup .... \cup r_k$	2. $E = E_1 \bigcup E_2 \bigcup ... \bigcup E_k$
3. $r = (r_1)^*$	3. $E = EE_1 \mid \varepsilon$ or $E = E_1E \mid \varepsilon$
4. $r = (r_1)^+$	4. $E = EE_1 \mid E_1$ or $E = E_1E \mid E_1$
5. $r = b \in \Sigma$	5. $E = b$
6. $r = \varepsilon$	6. $E = \varepsilon$

Example

$$E = (abc)^* \cup (ff)^+$$

$E = E_1 \cup E_2$	hence	$E \rightarrow E_1 \mid E_2$
$E_1 = (E_3)^*$	hence	$E_1 \rightarrow E_1\, E_3 \mid \varepsilon$
$E_3 = abc$	hence	$E_3 \rightarrow abc$
$E_2 = (E_4)^+$	hence	$E_2 \rightarrow E_2\, E_4 \mid E_4$
$E_4 = ff$	hence	$E_4 \rightarrow ff$

We can therefore conclude that every regular language is free

But there are free languages that are not regular (e.g. palindromes)

REG $\subset$ CF (REG $\subseteq$ CF and REG $\neq$ CF)

Let us look for the subclass of free grammars that are equivalent to regular expressions

a first candidate: LINEAR GRAMMARS

A linear grammar has ***at most one nonterm.*** in its right part

$$A \to uBv \quad \text{with} \quad u, v \in \Sigma^*, B \in (V \bigcup \varepsilon)$$

The family of linear grammars is however still more powerful than regular languages.

Example: non regular linear language

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

$S \to aSc \mid ac$

UNILINEAR GRAMMARS (called «of type 3» in the Chomsky hierarchy)

RIGHT-LINEAR RULE: $A \to uB \quad \text{with} \quad u \in \Sigma^*, \quad B \in (V \bigcup \varepsilon)$

LEFT-LINEAR RULE: $A \to Bv \quad \text{with} \quad v \in \Sigma^*, \; B \in (V \bigcup \varepsilon)$

A grammar is ***unilinear*** iff its rules are either **all right-linear** or **all left-linear**

Syntax trees grow totally unbalanced (resp. toward the right or left)

Without loss of generality one can require that a unilinear grammar has

- ***STRICTLY unilinear rules***: with at most one terminal $\boldsymbol{A \to aB, \quad a \in (\Sigma \cup \varepsilon) \quad B \in (V \cup \varepsilon)}$
- all **terminal rules are empty**: e.g., rule $B \to b$ replaced by rules $B \to bB'$ and $B' \to \varepsilon$

Therefore we can assume (for the right case) just rules of type $A \to aB | \varepsilon$ with $a \in \Sigma, B \in V$
(for the left case rules of type $A \to Ba | \varepsilon$ with $a \in \Sigma, B \in V$)

Regular expressions can be translated into (strictly) unilinear grammars
(not proven here: it can be shown using finite state automata, see Chap.3)

Therefore **REG $\subseteq$ UNILIN**

Example: strings with substring *aa* and ending with *b* defined by (ambiguous) r.e.

$$(a \mid b)^* aa(a \mid b)^* b$$

1. right-linear grammar G_r

$S \to aS \mid bS \mid aaA \quad A \to aA \mid bA \mid b$

2. left-linear gramm. G_l

$S \to Ab \quad A \to Aa \mid Ab \mid Baa$

$B \to Ba \mid Bb \mid \varepsilon$

3. non-unilinear equiv. gramm.

$E_1 \to E_2 aaE_2 b \quad E_2 \to E_2 a \mid E_2 b \mid \varepsilon$

FROM UNILINEAR GRAMMAR TO THE REGULAR EXPRESSION: LINEAR LANGUAGE EQUATIONS

We show that from any unilinear grammar one can obtain an equivalent regular expression

Therefore **UNILIN** $\subseteq$ **REG**

hence (by the previous -still unproved- property **REG** $\subseteq$ **UNILIN**, p.10) **UNILIN = REG**

Rules of the unilinear right grammar can be **seen as equations**

unknowns: the languages generated by every nonterminal : ex. L_S, L_A, L_B, …

Let ***G*** be - strictly unilinear right (e.g.) and
- (with no loss of generality) with all terminal rules empty ($A \to \varepsilon$)

$x \in \Sigma^*$ is in L_A in the following cases:

1. x is the empty string (rule in P: $A \to \varepsilon$);
2. $x = ay$ (rule in P: $A \to aB$ and $y \in L_B$)

for every n.t. A_0 defined by $A_0 \to a_1 A_1 \mid a_2 A_2 \mid ... \mid a_k A_k \mid \varepsilon$

$L_{A_0} = a_1 L_{A_1} \cup a_2 L_{A_2} \cup ... \cup a_k L_{A_k} \cup \varepsilon$

therefore we obtain a system of $n = \mid V \mid$ equations in n unknowns

to be solved with the method of substitution + applying the *Arden identity*

Arden Identity

Equation $X = KX \cup L$ (with K nonempty lang. and L any language)

has exactly one solution $X = K^* L$ $(K^* L = KK^* L \cup L)$

1

2

NB: we do not prove unicity

2 is a solution: this expr. is obtained by substituting 2 in 1

Example: Language Set Equations

Grammar for a list of (possibly missing) elements ***e*** divided by separator *s*

$$S \to sS \mid eA \quad A \to sS \mid \varepsilon$$

is transformed into a system of equations:

$$\begin{cases} L_S = sL_S \cup eL_A \\ L_A = sL_S \cup \varepsilon \end{cases}$$

subst. second eq. in the first one

$$\begin{cases} L_S = sL_S \cup e(sL_S \cup \varepsilon) \\ L_A = sL_S \cup \varepsilon \end{cases}$$

distrib. propr. of concat. over $\cup$ then factorize L_s as a common suffix

$$\begin{cases} L_S = (s \cup es)L_S \cup e \\ L_A = sL_S \cup \varepsilon \end{cases}$$

applying Arden identity

$$\begin{cases} L_S = (s \cup es)^* e \\ L_A = sL_S \cup \varepsilon \end{cases} \qquad L_A = s(s \cup es)^* e \cup \varepsilon$$

COMPARISON OF REGULAR AND CONTEXT-FREE LANGUAGES

We introduce some properties useful to show that some languages are (not) regular

regular languages (and therefore unilinear grammars) exhibit ***inevitable repetitions***

PROPERTY: Let G be a unilinear grammar

Every sufficiently long sentence x (i.e., longer than a grammar-dependent constant k)

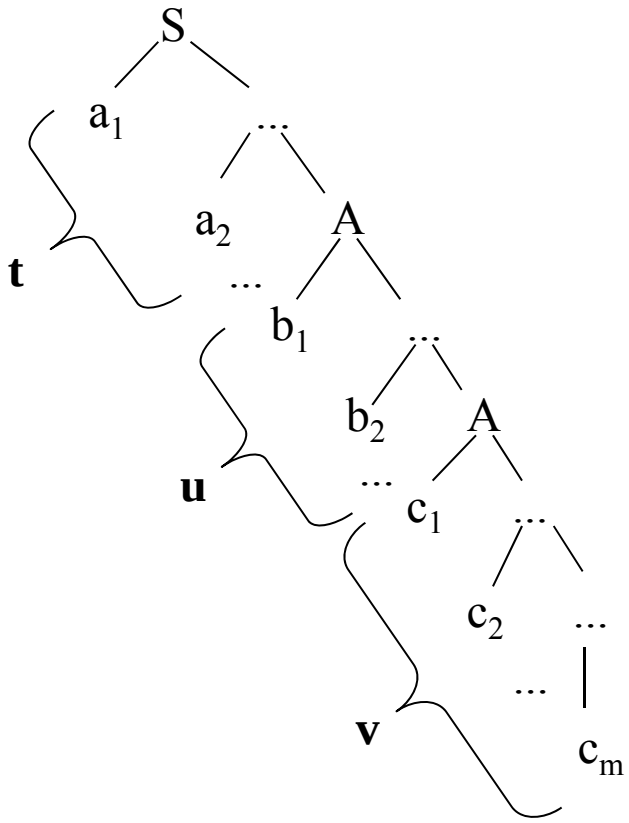
Can be factorized as $x = t\ u\ v$ - with **nonempty *u***

so that, $\forall\ n \geq 1$, the string $t\ u^n\ v \in L(G)$

(the sentence can be «pumped» by injecting string u an arbitrary number of times)

NOTE the analogy with the ***pumping lemma*** of finite state automata…

Proof:
Consider a strictly right-linear G with k n.t. symbols
In the derivation of a sentence x whose length is k or more,
there is necessarily a n.t. A that appears at least two times



$$t = a_1 a_2 \ldots, \quad u = b_1 b_2 \ldots \quad v = c_1 c_2 \ldots c_m$$

$$S \overset{+}{\Rightarrow} tA \overset{+}{\Rightarrow} tuA \overset{+}{\Rightarrow} tuv$$

then it is also possible to derive tv, $tuuv$ etc.

We use this property to show that the language with two equal exponents
$L_1 = \{ a^n c^n \mid n \geq 1 \}$ is ***not*** regular
Let us assume by contradiction that it is regular: then

NB: this ***k*** is the one of the previous proof (previous slide.)

$x = a^k c^k = tuv$ with u non empty, and there are three ways to take u

		t	u	v	*pumped string*
u made of *a*'s	⟶	1. $t = a^h$	$u = a^i$	$v = a^j c^k$	$a^h a^i a^i a^j c^k$
a's and *c's*	⟶	2. $t = a^h$	$u = a^i c^j$	$v = c^m$	$a^h a^i c^j a^i c^j c^m \notin a^+ c^+$
u made of *c*'s	⟶	3. $t = a^k c^h$	$u = c^i$	$v = c^j$	$a^k c^h c^i c^i c^j$

In all cases the pumped string (which would be in the language if this was regular), does not have the required form $a^n c^n$

THE ROLE OF SELF-NESTED DERIVATIONS

$$A \overset{+}{\Rightarrow} uAv \quad u \neq \varepsilon \wedge v \neq \varepsilon$$

Self-nested derivations are

absent from unilinear grammars of regular languages
present in grammars of free nonregular languages
(palindromes, Dyck languages, language with equal exponents, ...)

lack of self-nested derivations allows one to solve lang. equations of unilinear grams.

Hence …

GRAMMAR WITHOUT SELF-NESTED DERIVATIONS
$\Rightarrow$
REGULAR LANGUAGE

NB: the converse does not necessarily hold:
a grammar with self-nested derivations may generate a regular language

Example:
Self-nesting grammar generating a regular language

$$G = \{S \to aSa \mid \varepsilon\}$$

$$S \Rightarrow aSa \qquad L(G) = (aa)^*$$

NB:the language is regular

it is also defined by

$$G_d = \{S \to aaS \mid \varepsilon\}$$

Example:
grammar ***without self-nesting***
although ***not linear***: unilinear gramm.
is a sufficient, not necessary condition
for generating a regular language

$$G: \quad S \to AS \mid bA \quad A \to aA \mid \varepsilon$$

$$\begin{cases} L_S = L_A L_S \cup bL_A \\ L_A = aL_A \cup \varepsilon \end{cases}$$

$$\begin{cases} L_S = L_A L_S \cup bL_A \\ L_A = a^* \end{cases}$$

$$L_S = a^* L_S \cup ba^*$$

$$L_S = (a^*)^* ba^* \qquad L(G) = a^* ba^*$$

LIMITS OF CONTEXT-FREE LANGUAGES

In context-free languages all sufficiently long sentences necessarily contain

two substrings that can be repeated arbitrarily many times,

thus originating self-nested structures

This hinders the derivation of strings with ***three or more*** parts that are repeated the same number of times

NB: we do not provide a proof: see the textbook, §2.7.1

LIMITS OF CONTEXT-FREE LANGUAGES

THE LANGUAGE OF THREE EQUAL POWERS IS NOT FREE

$$L = \left\{ a^n b^n c^n \mid n \geq 1 \right\}$$

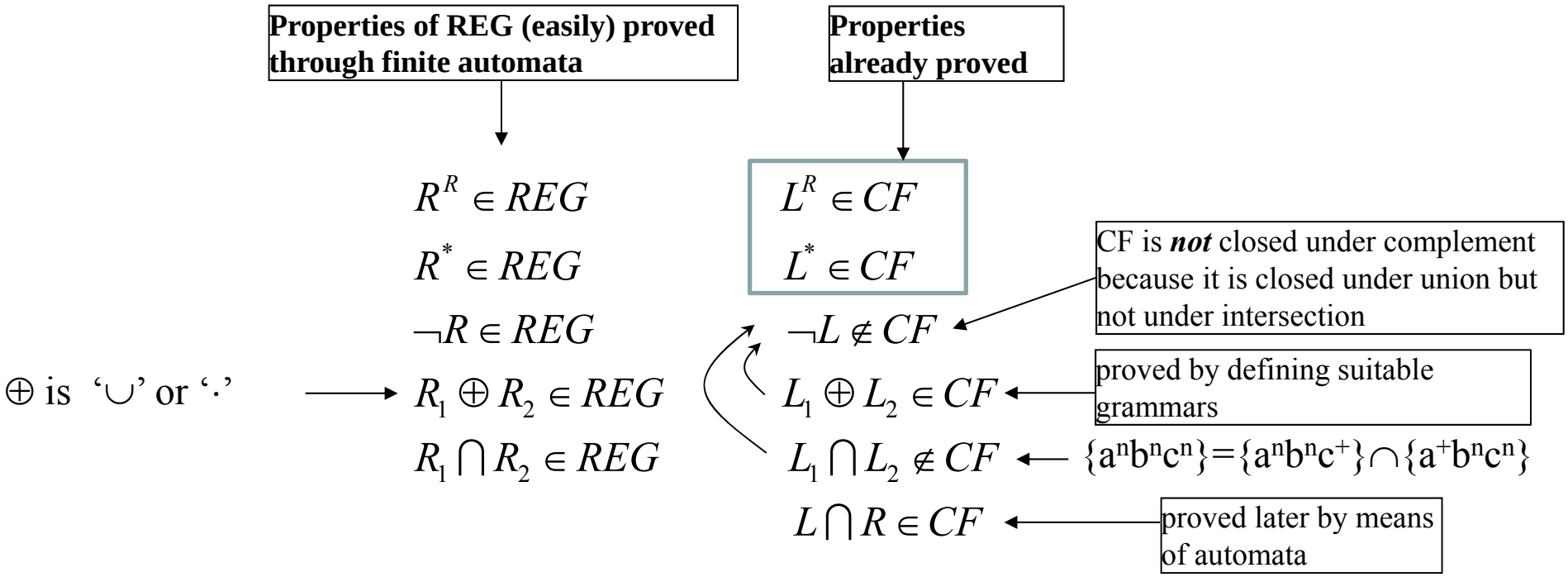
LANGUAGE OF COPIES OR REPLICA IS NOT FREE
(NB: this property in not proved here)
Very frequent in technical languages, when elements in two or more lists must be identical or be in some correspondence (ex. Formal and actual parameters)

$$L_{replica} = \left\{ uu \mid u \in \Sigma^+ \right\}$$

$$\Sigma = \left\{ a, b \right\} \quad x = abbbabbb$$

CLOSURE PROPERTIES OF **REG** (regular) AND **CF** (context free) FAMILIES

Let languages $\boldsymbol{L} \in \boldsymbol{CF}$ and $\boldsymbol{R} \in \boldsymbol{REG}$:



NB: The «non-inclusion» symbol $\notin$ means: «there exist some language not in the family»
(not: «there is no language in the family»)

INTERSECTION OF FREE LANGUAGES WITH REGULAR LINGUAGES

To make a grammar more selective (to constrain the denoted language) …

… one can ***filter*** it through (i.e., intersect it with) a regular language
(the result is always free – see slides on push-down automata and textbook, §4.3.1)

Example: Regular filters on the Dyck language L_D

sentences without substring *cc*, i.e., without nested parentheses → $L_1 = L_D \bigcap \neg(\Sigma^* cc \Sigma^*) = (ac)^*$

sentences with no more than one nest → $L_2 = L_D \bigcap \neg(\Sigma^* ca \Sigma^*) = \left\{ a^n c^n \mid n \geq 0 \right\}$

Both languages are free, the first one is also regular