Theory of Computation

MIEIC, 2nd Year

João M. P. Cardoso
Email: jmpc@acm.org

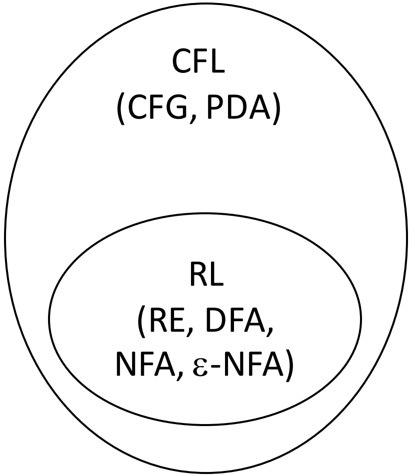
U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO
DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Outline

- Context-Free Grammars (CFGs)

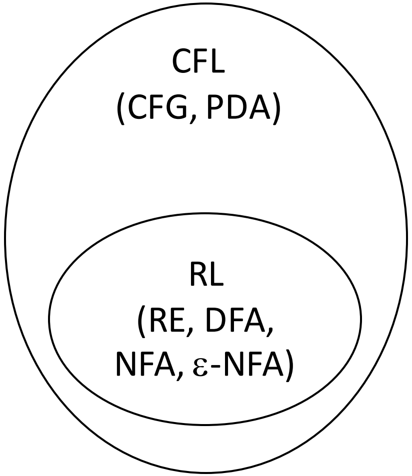
Context-Free Grammars (CFGs)

- A notation able to specify more general languages than the regular languages
  - Used for programming languages and compilers (since the 60's)
  - And in many text processing systems, e.g., see the use of DTDs (Document Type Definition) in XML
- Example: palindrome
  - $L=\{w \mid w=w^R \text{ and } w\in \{0,1\}^*\}$
    - Examples of strings in L: 0110, 11011, ε
  - It is not a regular language
    - Pumping Lemma
    - Selected n, be $w=0^n10^n=xyz$
    - Let y equal to one or more zeros of the first part of w, xz is also recognized by the automaton, but as x has less 0s than z it is not a palindrome and contradicts the hypothesis to be an automaton of the language



Context-Free Grammars (CFGs)

- $L=\{w \mid w=w^R \text{ and } w\in \{0,1\}^*\}$ is not a regular language
- Other languages of palindromes:
  - $L=\{w \mid w=w^R \text{ and } w\in \{0\}^*\}$
    - Is a regular language?
  - $L=\{0^n0^n \mid n\geq 0\}$ a regular language?
    - Is a regular language?



Example of the Palindrome

$L=\{w \mid w=w^R \text{ and } w\in \{0,1\}^*\}$

- Inductive (recursive) definition
  - Basis: 0, 1 and ε are palindromes
  - Induction: if *w* is a palindrome, 0*w*0 and 1*w*1 are also palindromes; nothing more is a palindrome
- Supposing a variable P:
  - Basis: P = {0, 1, ε}
  - Induction: P= 0P0 OR 1P1

Example of the Palindrome

$L=\{w \mid w=w^R \text{ and } w\in \{0,1\}^*\}$

- Inductive (recursive) definition
  - Basis: 0, 1 and ε are palindromes
  - Induction: if *w* is a palindrome, 0*w*0 and 1*w*1 are also palindromes; nothing more is a palindrome
- Alternative notation:
  - productions for P, variable that represents the language of the palindromes

1. $P \to \varepsilon$
2. $P \to 0$
3. $P \to 1$
4. $P \to 0P0$
5. $P \to 1P1$

=

$P \to \varepsilon \mid 0 \mid 1$

$P \to 0P0 \mid 1P1$

- Productions 1,2,3 constitute the basis; 4,5 are recursive
  - Interpretation of rule 4: if *w* is in P then 0*w*0 is also in P

Regular Expressions (REs) are not enough!

- **Example of parenthesis:** If for a given program, we remove every symbol that is not parenthesis, we obtain strings like (()())(). For example, we never obtain, (() or ())(, because the parenthesis must be paired

  a) Show that the language of these strings is not regular

  b) Show a CFG for the language

- **Answer:**

  a) The language with strings (((…()…))) of length 2n is homomorphic of the language 0^n1^n (already proved as non-regular)

  b) B $\rightarrow$ BB | (B) | ε

    BB: the concatenation of two paired strings is paired

    (B): parenthesis embracing a paired string form strings that belong to the language

    ε Is the basis case

Definition of CFG

- CFG $G=(V, T, P, S)$
  - T are the terminals, symbols used in the strings of the language
  - V are the variables (of the language), the non-terminals or syntactic categories
  - S is a start symbol, the variable of the defined language (the other variables are auxiliary variables)
  - P is a finite set of productions or rules of the form
    - $H \rightarrow B_1B_2\ldots B_n$
    - Partial definition of H, the head, being $B_1B_2\ldots B_n$, the body, a sequence of terminals and non-terminals
    - The strings of the language are the ones we obtain substituting the non-terminals B_i by strings that we now belong to the language B_i

Definition of CFG

- CFG Example:
  1. $P \rightarrow \varepsilon$
  2. $P \rightarrow 0$
  3. $P \rightarrow 1$
  4. $P \rightarrow 0P0$
  5. $P \rightarrow 1P1$
- Formal definition:
  - G = ({P}, {0,1}, A, P), where A represents the 5 productions of P
    - A = $\{P \rightarrow \varepsilon, P \rightarrow 0, P \rightarrow 1, P \rightarrow 0P0, P \rightarrow 1P1\}$
  - I.e.:
    - G = $(\{P\}, \{0,1\}, \{P \rightarrow \varepsilon, P \rightarrow 0, P \rightarrow 1, P \rightarrow 0P0, P \rightarrow 1P1\})$

Example of Expressions

- Represent the arithmetic expressions with +, ×, parenthesis and identifiers
  - Alphabet of identifiers: ‘a’, ‘b’, ‘0’, ‘1’
  - Identifiers: begin with a letter followed by any number of letters and digits
  - Other terminals: ‘(‘, ‘)’, ‘+’, ‘×’
  - RE for the identifiers: (a+b)(a+b+0+1)*
- Use of a variable E for the expressions and of a variable I for the identifiers

1. $E \to I$
2. $E \to E+E$
3. $E \to E \times E$
4. $E \to (E)$
5. $I \to a$
6. $I \to b$
7. $I \to Ia$
8. $I \to Ib$
9. $I \to I0$
10. $I \to I1$

More compact form:

$E \to I \mid E+E \mid E \times E \mid (E)$

$I \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

Inference

- Recursive Inference
  - From the basis up: from the bodies to the headers of the rules
  - Start with the known rules for the symbols in the input string and then apply rules; in the end we reach all using a horizontal search
  - Example: **a×(a+b00)**

1. E → I
2. E → E+E
3. E → E×E
4. E → (E)
5. I → a
6. I → b
7. I → Ia
8. I → Ib
9. I → I0
10. I → I1

	String	From	Used production	Used chains
i	a	I	5	
ii	b	I	6	
iii	b0	I	9	ii
iv	b00	I	9	iii
v	a	E	1	i
vi	b00	E	1	iv
vii	a+b00	E	2	v, vi
viii	(a+b00)	E	4	vii
ix	**a×(a+b00)**	E	3	v, viii

Derivation

- From top to bottom; from the headers to the bodies of the rules
- Start with the goal, the target string, and apply the rules, substituting the variables by the respective bodies until we only have a chain of terminals; reach all using a vertical search
- Derivation step: $\Rightarrow$
  - CFG $G=(V,T,P,S)$ $\alpha, \beta \in (V \cup T)^*$ $A \in V$ $A \rightarrow \gamma \in P$
  - $\alpha A \beta \Rightarrow_G \alpha\gamma\beta$
  - $\Rightarrow^*$ means derivation in 0 or more steps
- $E \Rightarrow E \times E \Rightarrow I \times E \Rightarrow a \times E \Rightarrow$

$a \times (E) \Rightarrow a \times (E+E) \Rightarrow a \times (I+E) \Rightarrow a \times (a+E) \Rightarrow$

$\Rightarrow a \times (a+I) \Rightarrow a \times (a+I0) \Rightarrow a \times (a+I00) \Rightarrow a \times (a+b00)$

$E \rightarrow I$
$E \rightarrow E+E$
$E \rightarrow E \times E$
$E \rightarrow (E)$
$I \rightarrow a$
$I \rightarrow b$
$I \rightarrow Ia$
$I \rightarrow Ib$
$I \rightarrow I0$
$I \rightarrow I1$

Derivation

- In the previous example we selected in each step the leftmost variable:
  - Leftmost derivation $\overset{*}{\underset{lm}{\Rightarrow}}$
- Rightmost derivation $\overset{*}{\underset{rm}{\Rightarrow}}$
- Example:

$$E \underset{rm}{\Rightarrow} E \times E \underset{rm}{\Rightarrow} E \times (E) \underset{rm}{\Rightarrow} E \times (E+E) \underset{rm}{\Rightarrow}$$

$$\underset{rm}{\Rightarrow} E \times (E+I) \underset{rm}{\Rightarrow} E \times (E+I0) \underset{rm}{\Rightarrow} E \times (E+I00) \underset{rm}{\Rightarrow} E \times (E+b00) \underset{rm}{\Rightarrow}$$

$$E \times (I+b00) \underset{rm}{\Rightarrow} E \times (a+b00) \underset{rm}{\Rightarrow} I \times (a+b00) \underset{rm}{\Rightarrow} a \times (a+b00)$$

$$E \overset{*}{\underset{rm}{\Rightarrow}} a \times (a+b00)$$

E → I
E → E+E
E → E×E
E → (E)
I → a
I → b
I → Ia
I → Ib
I → I0
I → I1

Language of a Grammar

- The language of a CFG G=(V,T,P,S) is the set of strings (chains of terminal symbols) which have derivation from the start variable S:

$$L(G) = \{w \in T^* \mid S \overset{*}{\underset{G}{\Longrightarrow}} w\}$$

Language of a Grammar

w is palindrome $\rightarrow$ w $\in$ {0,1}* is in L(G_{pal})

- Theorem: L(G_{pal}) is the set of palindromes over {0,1}
  - Proof: w $\in$ {0,1}* is in L(G_{pal}) **if and only if (iff)** w is palindrome, i.e., $w=w^R$
  - [**if**] hypothesis: w is palindrome; induction in |w|
    - **Basis:** |w|=0 or |w|=1, i.e., w=ε, w=0, w=1 as there exist the productions (and P→ε, P→0, P→1 then

$$P \overset{*}{\Longrightarrow} w$$

    - **Induction :** suppose |w|≥2, as $w=w^R$, w must begin and end with the same symbol, w=0x0 or w=1x1. In addition, $x=x^R$. By hypothesis, $P \Longrightarrow 0P0 \overset{*}{\Longrightarrow} 0x0 = w$

Then $P \overset{*}{\Longrightarrow} x$

And similarly for 1x1. w is in L(G_{pal}). *qed (if)*

G_{pal}:

P → ε

P → 0

P → 1

P → 0P0

P → 1P1

Proof (cont.)

$w \in \{0,1\}^*$ is in $L(G_{pal})$ $\rightarrow$ w is palindrome

- **[and only if]** hypothesis: w is in G_{pal}, $P \stackrel{*}{\Rightarrow} w$ induction in the number of steps of a derivation of w from P
  - **Basis:** derivation with a single step: use non-recursive rules. We obtain ε, 0, 1 which are all palindromes
  - **Induction:** suppose that the derivation has n+1 steps and the statement is true for all the derivations with n steps, $P \stackrel{*}{\Rightarrow} x$ then x is palindrome $x=x^R$
  - A derivation with n+1 steps can only be

$$P \Rightarrow 0P0 \stackrel{*}{\Rightarrow} 0x0 = w \quad \text{or} \quad P \Rightarrow 1P1 \stackrel{*}{\Rightarrow} 1x1 = w$$

  - As $w^R=(0x0)^R=0x^R0=0x0=w$ then w is a palindrome. *Qed (and only if)*

qed (iff)

G_{pal}:

$P \rightarrow \varepsilon$

$P \rightarrow 0$

$P \rightarrow 1$

$P \rightarrow 0P0$

$P \rightarrow 1P1$

Sentential Forms

- Sentential forms are derivations from the start symbol
  - CFG G=(V,T,P,S)
  - $\alpha \in (V \cup T)^*$ is a sentential form if $S \overset{*}{\underset{G}{\Rightarrow}} \alpha$
- Left (right) sentential form
  - Leftmost (rightmost) derivation
- L(G) consists of the sentential forms that belong to T* (i.e., only have terminals)

Exercise 1

- Define context-free grammars (CFGs) for the following non-regular languages

i) The set $\{0^n 1^n \mid n \geq 1\}$

ii) The set $\{a^i b^j c^k \mid i \neq j \text{ or } j \neq k, i,j \geq 0\}$

- Answer

Exercise 1

- Define context-free grammars (CFGs) for the following non-regular languages

i) The set $\{0^n 1^n \mid n \geq 1\}$

ii) The set $\{a^i b^j c^k \mid i \neq j \text{ or } j \neq k, i,j \geq 0\}$

- Answer

ii) $S \rightarrow AB \mid CD$

$A \rightarrow aA \mid \varepsilon$

$B \rightarrow bBc \mid E \mid cD$

$C \rightarrow aCb \mid E \mid aA$

$D \rightarrow cD \mid \varepsilon$

$E \rightarrow bE \mid b$

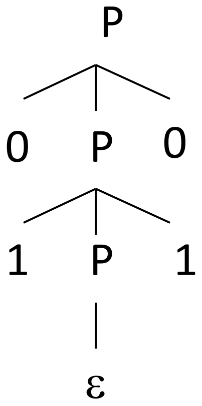
Syntax Trees (or Analysis Trees)

- Data structure most used to represent the input program in a compiler
  - Helps compiler analysis and code generation
- Consider G=(V,T,P,S); a syntax tree for G is a tree in which
  - The label of each internal node is a grammar variable
  - The label of each leaf node is a grammar variable, a terminal or ε (in this case unique child)
  - If an internal node has a label A and children labeled $X_1 \ldots X_k$ then $A \rightarrow X_1 \ldots X_k$ is a production in P

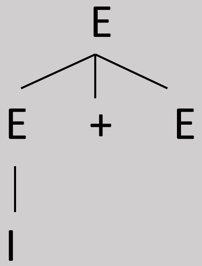
Examples of Syntax Trees

- A syntax tree represents a derivation
- Derivation $P \overset{*}{\Longrightarrow} 0110$

G_{pal}:
$P \rightarrow \varepsilon$
$P \rightarrow 0$
$P \rightarrow 1$
$P \rightarrow 0P0$
$P \rightarrow 1P1$



- A derivation can be partial $E \overset{*}{\Longrightarrow} I + E$
- Derivation



- Each internal node in the tree corresponds to the application of a production

Yield of a Syntax Tree

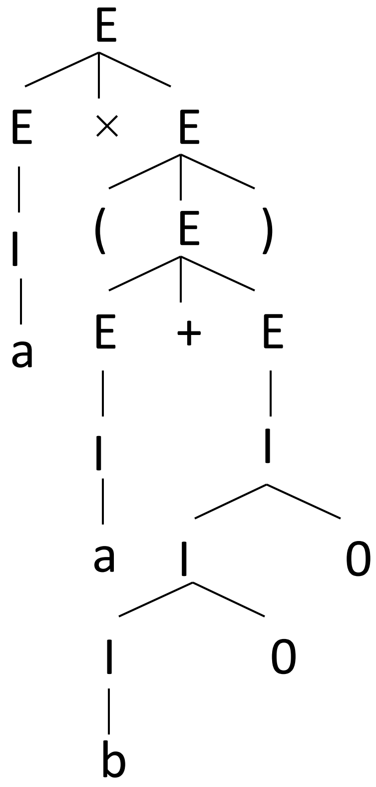
- The yield of a tree is the concatenation of the symbols in the leaves (left-to-right)
- All the yields of the trees with the start variable of G as root are sentential forms of G
- The yields of these trees that are terminals (leaves with terminals or ε) are strings of the language

A Syntax Tree for a×(a+b00)

Grammar G:

E → I | E+E | E×E | (E)

I → a | b | Ia | Ib | I0 | I1



$E \Rightarrow E{\times}E \Rightarrow I{\times}E \Rightarrow a{\times}E \Rightarrow a{\times}(E) \Rightarrow a{\times}(E+E)$
$\Rightarrow a{\times}(I+E) \Rightarrow a{\times}(a+E) \Rightarrow a{\times}(a+I) \Rightarrow$
$a{\times}(a+I0) \Rightarrow a{\times}(a+I00) \Rightarrow a{\times}(a+b00)$

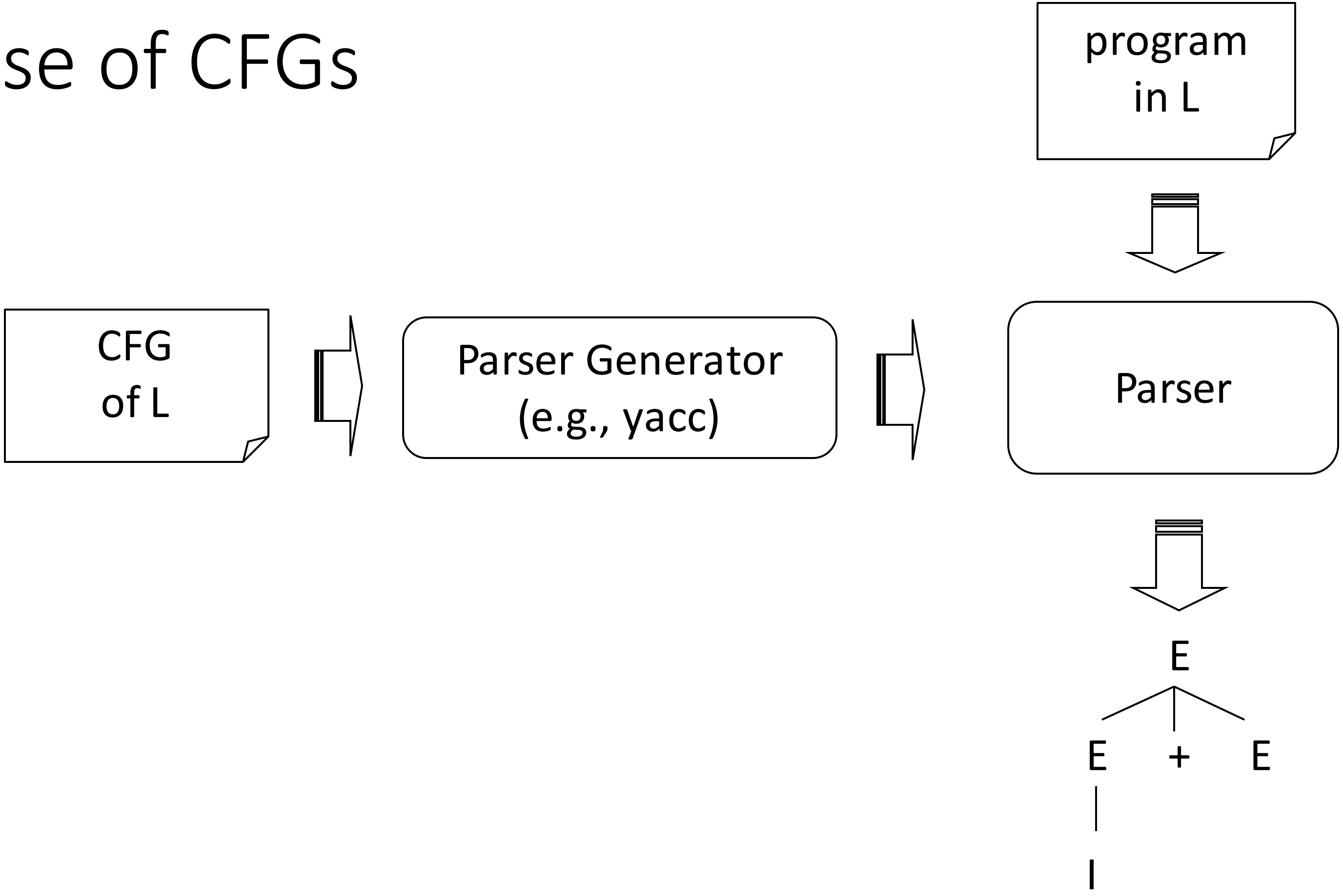
Equivalences

- Given a grammar G=(V,T,P,S), the following are equivalent:
  - The recursive inference procedure determines that the terminal string *w* is in the language of variable A
  - $A \overset{*}{\Rightarrow} w$
  - $A \underset{lm}{\overset{*}{\Rightarrow}} w$
  - $A \underset{rm}{\overset{*}{\Rightarrow}} w$
  - There is a syntax tree with root *A* and yield w
- The equivalences are true even if *w* contains variables

Parsers

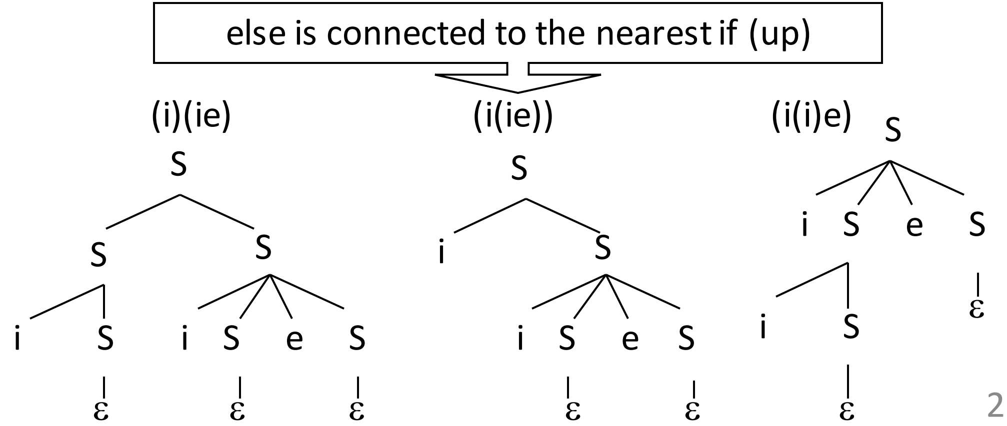
- Formal grammars proposed by Noam Chomsky
  - Limited to describe natural languages
  - Very useful to describe artificial languages
- Programming Languages
  - Some aspects can be defined by regular expressions
  - But the pairing of parenthesis and if-else structures is not a regular language – they need CFGs
- There exist programs (e.g., Lex and Yacc, Flex and Bison, JavaCC, Antlr) that from an input CFG for a given language L, automatically generate a parser (i.e., a program able to implement the CFG)
  - Useful for language processors (in compilers, interpreters, translators)

Use of CFGs



Syntax Trees Meanings

- Example of the if-else:
  - In a programming language, the construction *if* can be isolated or paired with *else*. Examples: *i, ie, ieie, iiee*. And not: *ei, iee*
  - a) define a CFG for this language
  - b) show all the syntax trees of *iie*; which one is the correct in C?
- Answer:
  - a) $S \rightarrow \varepsilon \mid SS \mid iS \mid iSeS$
  - b)



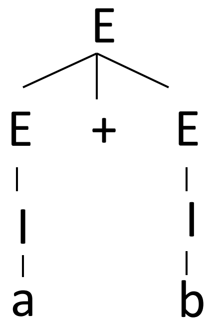
Ambiguity of a CFG

- Example of the if-else: analysis of the string *iie*
  - 3 syntax trees with different meaning
- Example of arithmetic expressions: analysis of a+b
  - Derivation 1: $E \Rightarrow E+E \Rightarrow I+E \Rightarrow a+E \Rightarrow a+I \Rightarrow a+b$
  - Derivation 2: $E \Rightarrow E+E \Rightarrow E+I \Rightarrow I+I \Rightarrow I+b \Rightarrow a+b$
  - Derivations are different but syntax tree is the same: same meaning
- A CFG G=(V,T,P,S) is **ambiguous** if
  - There exists a string w in T* to which there exist two different syntax trees with root S and leaves (yield) w
- If there is not none of those strings the CFG is **not ambiguous**

Grammar:

$E \rightarrow I \mid E+E \mid E\times E \mid (E)$

$I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

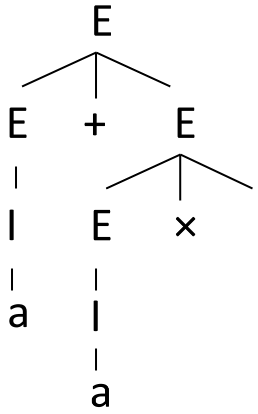


Priorities of Operators

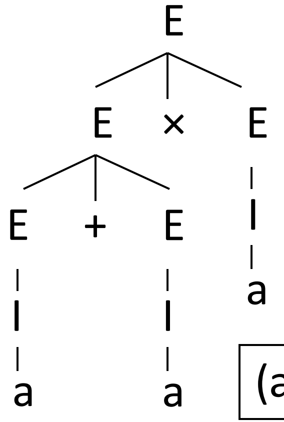
Grammar:

E → I | E+E | E×E | (E)

I → a | b | Ia | Ib | I0 | I1



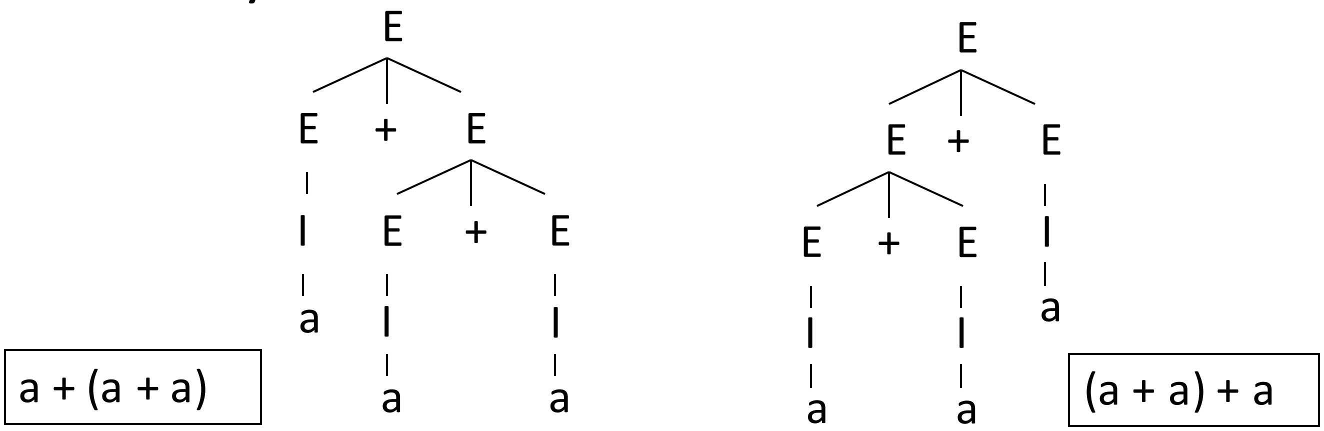
a + (a × a)



(a + a) × a

- String a + a × a has 2 syntax trees – ambiguous grammar
  - To eliminate the ambiguity we can use **rules of priority**
    - × has priority over +, applied before + left or right (1st tree respects the rule, the 2nd not)
    - a + a × a = a + (a × a) ≠ (a + a) × a

Associativity



- String a + a + a has 2 syntax trees – ambiguous grammar
  - To eliminate the ambiguity we can use the associativity rule
    - Sequences of operators with the same priority are left associative (2nd tree does not respect this rule, the 1st tree respects)
    - a + a + a = (a + a) + a
    - As the addition is associative, the results is the same in both analysis
    - And if it is the division? Ex: 8 / 4 / 2

Eliminating the Ambiguity in the CFG

- The ambiguity can be eliminated adding new variables, to distinguish levels of priority and association rules
- Concepts:
  - Factor: expression that cannot be split by adjacent operators (+ or ×) – identifiers and expressions between parenthesis
  - Term: expression that cannot be split by +
  - Expression: other expressions – an expression is thus a sum of one or more terms

Non-Ambiguous Grammar

- Original grammar (ambiguous):
  - E → I | E+E | E×E | (E)
  - I → a | b | Ia | Ib | I0 | I1
- Modified grammar:
  - E → I | E+I | E×I | (E)
  - I → a | b | Ia | Ib | I0 | I1

?

E → J | E×J

J → I | J+I

I → a | b | Ia | Ib | I0 | I1 | (E)

?

Does not respect the priority of the operators and…

- Modified grammar (respecting the priority of the operators):
  - E → T | T+E
  - T → F | F×T
  - F → I | (E)
  - I → a | b | Ia | Ib | I0 | I1
- Exercise: show the syntax tree for a+a×a

Ambiguity and Derivations

- Theorem: for each grammar and each string *w* of terminals, there are two different syntax trees **iff** there are two different leftmost *(and consequently two different rightmost)* derivations of *w* (starting in S, the start variable of the grammar)
- Example: leftmost derivations of a+a×a

Ambiguity in a Language (homework)

- A context-free language L is **ambiguous** if all the grammars for L are ambiguous
- Example: $L= \{a^n b^n c^m d^m \mid n \geq 1, m \geq 1\} \cup \{a^n b^m c^m d^n \mid n \geq 1, m \geq 1\}$
- Exercise:
  a) Define a CFG
  b) Show 2 syntax tree for w=aabbccdd

Examples of CFGs and Parsers

Markup Languages: HTML

- The strings of these languages contains marks to structure the text and to control its presentation

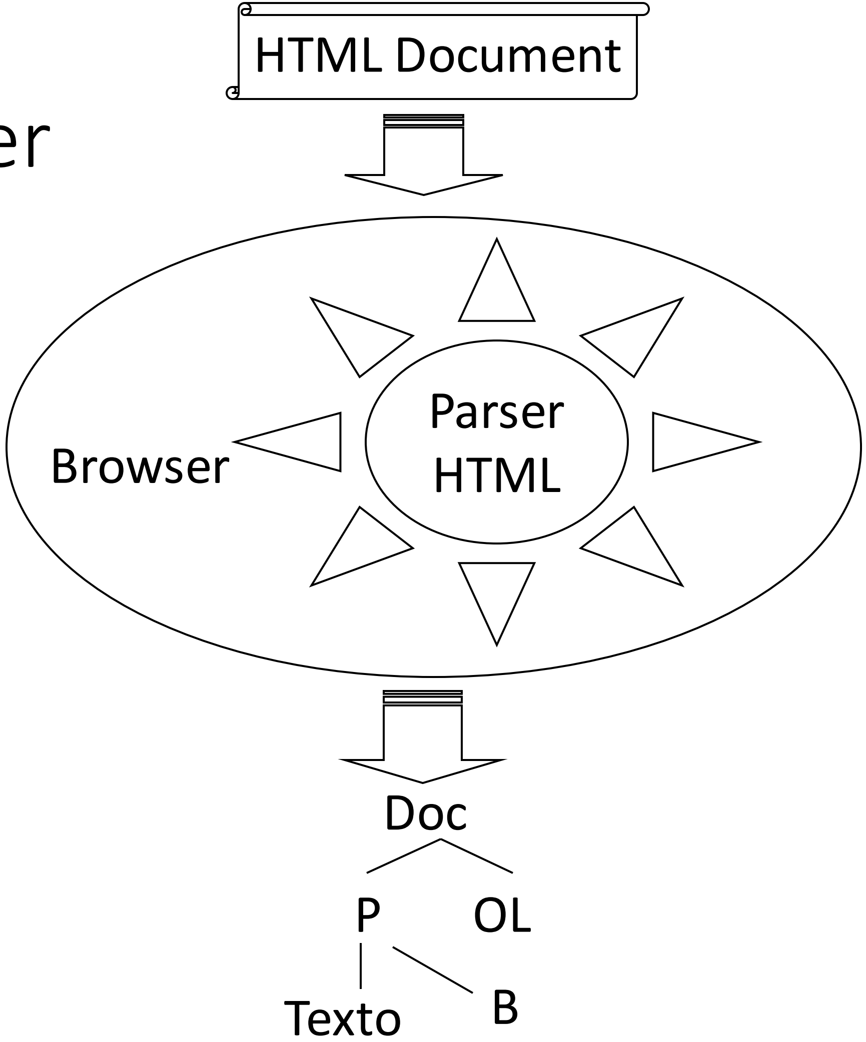
```
<P>The evaluation of <B>CLF</B> includes:
<OL><LI>Midterm Exam
        <LI>Final Exam
        <LI>Exercises
</OL>
```

Char → a | A | …	Element → Text |
Texto → ε | Char Text	<P> Doc |
Doc → ε | Element Doc	<B> Doc </B> |
	<OL> List </OL>
List → ε | ItemList List	ItemList → <LI> Doc

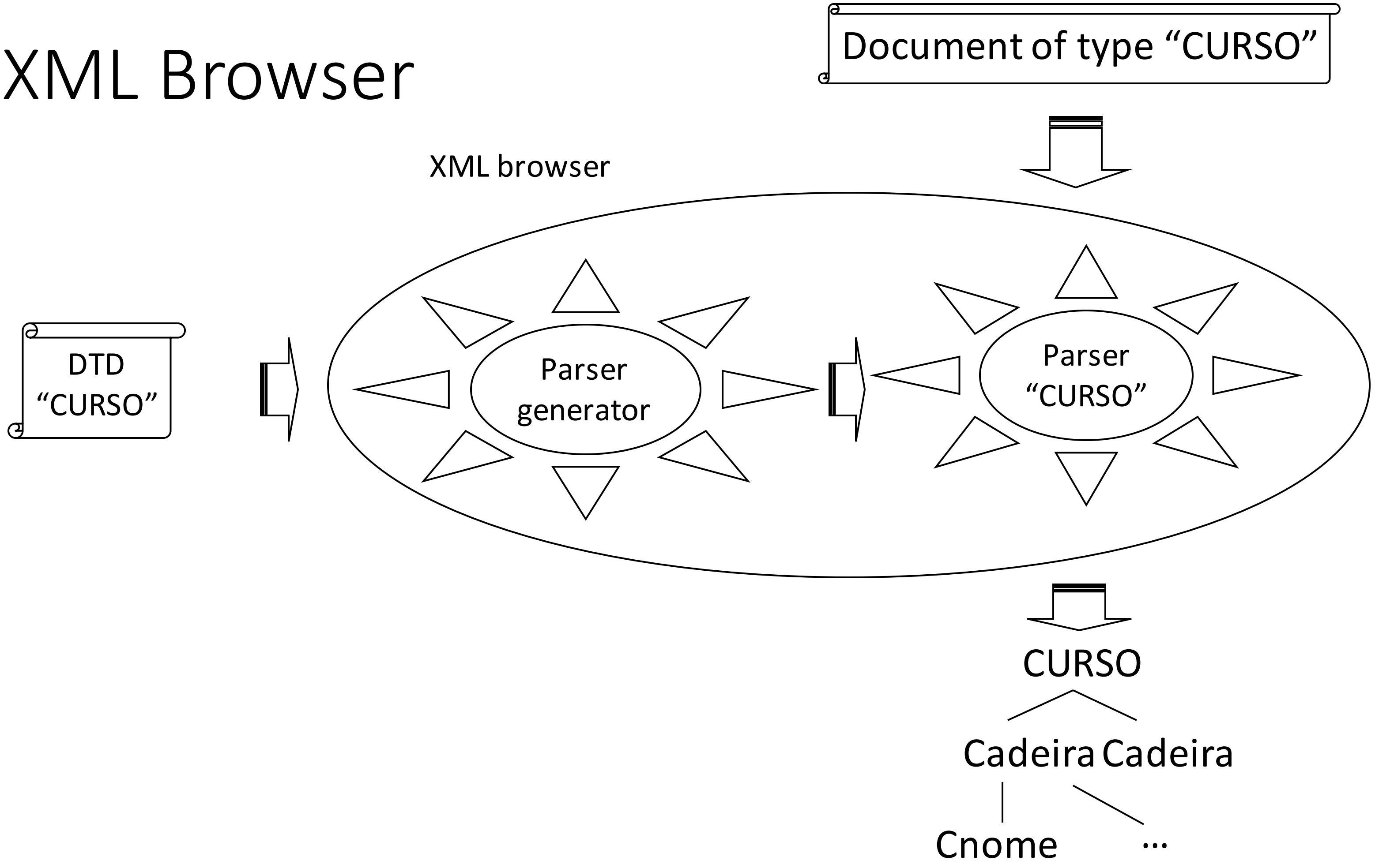
XML (*eXtensible Markup Language*)

- Some aspects of HTML do not need the power of CFGs
  - The 1st and 2nd lines define that Text can be represented by the regular language (a+A+…)*
  - The elements <B> and <OL> require a CFG
- In HTML, the grammar is predefined
  - Browsers include an HTML parser to analyze the input documents, to produce a tree and show results
- XML allows users to define their own grammar in a DTD (*Document Type Definition*)
  - Documents explicit the DTD which specifies the structure
  - Browsers need to have the capacity to process the grammar to validate the input documents

HTML Browser



XML Browser



Document of type “CURSO”
XML browser
DTD
“CURSO”
Parser
generator
Parser
“CURSO”
CURSO
Cadeira Cadeira
Cnome
…

XML Documents

```
<CURSO>
  <GRAU>L</GRAU>
  <CADEIRA>
    <CNOME>Lógica</CNOME>
    <PROF>Francisco</PROF>
  </CADEIRA>
</CURSO>
```

```
<CURSO>
  <GRAU>M</GRAU>
  <CADEIRA>
    <CNOME>Matemática</CNOME>
    <PROF>Francisco</PROF>
    <ALUNO>Zé</ALUNO>
    <ALUNO>Rui</ALUNO>
    <ASSISTENTE>Luis</ASSISTENTE>
  </CADEIRA>
  <CADEIRA>
    <CNOME>Redes</CNOME>
    <PROF>Antonio</PROF>
  </CADEIRA>
</CURSO>
```

Syntax of DTDs

```
<!DOCTYPE name-of-DTD [
   list of element definitions
]>
<!ELEMENT name-of-element (description)>
```

- Description is a regular expression
  - Basis: other elements or #PCDATA (text without marks)
  - Operators:
    - "|" union
    - "," concatenation
    - "*" closure, zero or more
    - "+" closure, one or more
    - "?" closure, zero or one
  - Parenthesis can be used

DTD “CURSO” (course)

```
<!DOCTYPE CURSO [
<!ELEMENT CURSO (GRAU, CADEIRA+)>
<!ELEMENT CADEIRA (CNOME, PROF, ALUNO*, ASSISTENTE?)>
<!ELEMENT GRAU (L | M | D)>
<!ELEMENT CNOME (#PCDATA)>
<!ELEMENT PROF (#PCDATA)>
<!ELEMENT ALUNO (#PCDATA)>
<!ELEMENT ASSISTENTE (#PCDATA)>
]>
```

XML and CFGs

- Rewrite DTD in the notation of CFGs
  - Convert CFG with regular expressions in the body of the rules to CFG forms
- Basis: if the body is a concatenation then it is already in the CFG form
- Induction: 5 cases
  - $A \rightarrow E_1, E_2$ (concatenation, E_1 and E_2, allowed in DTD)
    - $A \rightarrow BC$ (add new B and C variables, allowed in CFGs)
    - $B \rightarrow E_1$ (E_1 and E_2 may not be in a CFG form)
    - $C \rightarrow E_2$
  - $A \rightarrow E_1 \mid E_2$
    - $A \rightarrow E_1$
    - $A \rightarrow E_2$

XML and CFGs

- $A \rightarrow (E_1)^*$
  - $A \rightarrow BA$ (B is new)
  - $A \rightarrow \varepsilon$
  - $B \rightarrow E_1$
- $A \rightarrow (E_1)^+$
  - $A \rightarrow BA$
  - $A \rightarrow B$
  - $B \rightarrow E_1$
- $A \rightarrow (E_1)?$
  - $A \rightarrow \varepsilon$
  - $A \rightarrow E_1$

Exercise: Convert the DTD “CURSO”

CURSO → GRAU, CAD

CAD → CADEIRA

CAD → CADEIRA CAD

CADEIRA → CNOME PROF AL ASS

AL → B AL | ε or AL → ALUNO AL | ε

B → ALUNO

ASS → ASSISTENTE | ε

GRAU → L | M | D

CNOME → #PCDATA

PROF → #PCDATA

ALUNO → #PCDATA

ASSISTENTE → #PCDATA

```
<!DOCTYPE CURSO [
<!ELEMENT CURSO (GRAU, CADEIRA+)>
<!ELEMENT CADEIRA (CNOME, PROF, ALUNO*, ASSISTENTE?)>
<!ELEMENT GRAU (L | M | D)>
<!ELEMENT CNOME (#PCDATA)>
<!ELEMENT PROF (#PCDATA)>
<!ELEMENT ALUNO (#PCDATA)>
<!ELEMENT ASSISTENTE (#PCDATA)>
]>
```