LANGUAGE PROCESSORS

UNIT 3: FORMAL GRAMMARS

David Griol Barres
dgriol@inf.uc3m.es
Computer Science Department
Carlos III University of Madrid
Leganés (Spain)

OUTLINE

- Motivation
- Definitions
- Formal grammars
  - BNF notation
  - Examples
  - Chomsky Hierarchy
  - Definitions
  - Parse representation
  - Derivations

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

OUTLINE

- Issues in parsing context-free grammars
  - Ambiguity
  - Recursive rules
  - Left-factoring
- Push-Down Automata

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Motivation

- A grammar is a powerful tool for describing and analyzing languages.
- A grammar is a set of rules (*productions*) by which valid sentences in a language are constructed.

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Definitions (I)

- Alphabet Σ
  - Finite, non-empty set of symbols (for example, letters and characters).
- Word w
  - Finite sequence of symbols drawn from an alphabet (sentence and string are also use as synonyms).
- Empty word λ, ε
  - word of length 0.
- Universe $W(\Sigma)$
  - All words that can be formed with elements from Σ.
- Language $L(\Sigma)$
  - Any subset of $W(\Sigma)$.

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Definitions (II)

- Grammar
  - Set of rules by which valid sentences in a language are constructed.
- Nonterminal
  - Grammar symbol that can be replaced to a sequence of symbols.
- Terminal
  - Actual word in the language (not further expansion is possible).
- Production
  - Grammar rule that defines how to replace/exchange symbols
- Derivation
  - A sequence of applications of the rules of a grammar that produces a finished string of terminals.

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Formal grammar

- $G=\{\Sigma_T, \Sigma_N, S, P\}$
  - Σ_T terminal symbols alphabet
  - Σ_N nonterminal symbols alphabet
    - $\Sigma_V = \Sigma_T \cup \Sigma_N$
    - $\Sigma_T \cap \Sigma_N = \varnothing$
  - S axiom or start symbol ($S \in \Sigma_N$)
  - P finite set of productions:
    - $w \rightarrow z$
    - $w \rightarrow^* z$
    - $w \rightarrow^+ z$

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

BNF notation (extended)

- Formal notation to describe the syntax of a given language
  - ::= meaning "is defined as"
  - | "or"

- Nonterminals are usually represented by uppercase letters
- Terminals by lowercase letters

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Example grammar

- G={Σ_T, Σ_N, S, P}

Σ_T ={x,y,z}

Σ_N ={S,A,B}

P ={S ::= AB,

A::= Ax | y,

B ::= z}

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

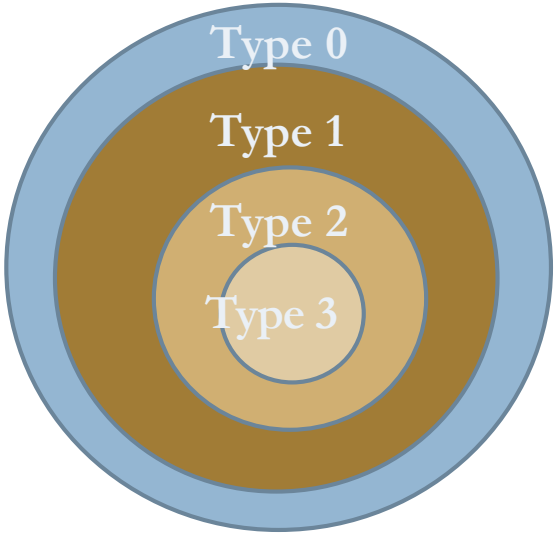
Grammar Hierarchy (Chomsky)

- **Type 0**: unrestricted grammars
  - u → v (u not null)
- **Type 1**: context-sensitive grammars
  - $P=\{xAy::=xvy \text{ where } x,y\in\Sigma_V{}^* \wedge A\in\Sigma_N \wedge v\in\Sigma_V{}^+\}$
  - uXw → uvw (v not null, X single nonterminal)
- **Type 2**: context-free grammars
  - $P=\{A::=v \text{ where } A\in\Sigma_N \wedge v\in\Sigma_V{}^+\}$
  - X → v (X single nonterminal)

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Grammar Hierarchy (Chomsky)

- **Type 3**: regular or linear grammars
  - X →a, X → aY, X → ε
  - linear on the left
    - $P = \{(A::=Ba) \cup (A::=a) \mid A,B \in \Sigma_N \wedge a \in \Sigma_T^*\}$
  - linear on the right
    - $P = \{(A::=aB) \cup (A::=a) \mid A,B \in \Sigma_N \wedge a \in \Sigma_T^*\}$



David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Definitions (II)

- Sentential form:
  - x is a sentential form if $S \rightarrow_* x$
- Sentence:
  - x is a sentence if it is a sentential form and $x \in \Sigma_T^*$
- Language generated by a grammar *G*:
  - $L(G)=\{x \text{ where } S \rightarrow_* x,\ x \in \Sigma_T^*\}$
- Grammar equivalence:
  - G_1 and G_2 are equivalent if $L(G_1)=L(G_2)$

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Parse representation

- We can represent the application of rules to derive a sentence in two ways:
  - A derivation.
  - A parse tree.

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Derivation

- A derivation is a sequence of applications of the rules of a grammar that produces a word (a string of terminals)
- Leftmost derivation
  - the leftmost nonterminal symbol is always replaced when the rules are applied
- Rightmost
  - the rightmost nonterminal symbol is always replaced when the rules are applied

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Example derivation

S ::= AB

A::= Ax | y

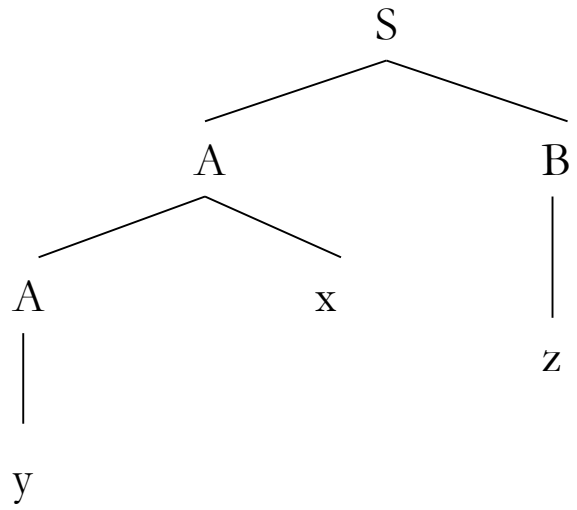
B ::= z

S -> AB -> AxB -> yxB -> yxz

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Parse tree

- It shows how each symbol derives from other symbols in a hierarchical manner. It does not encode the order the productions are applied



David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Issues in parsing context-free grammars

- Ambiguity
- Recursive rules
- Left-factoring

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Ambiguity (I)

- If a grammar permits more than one parse tree for the same sentence, then it is said to be ambiguous

- Example:

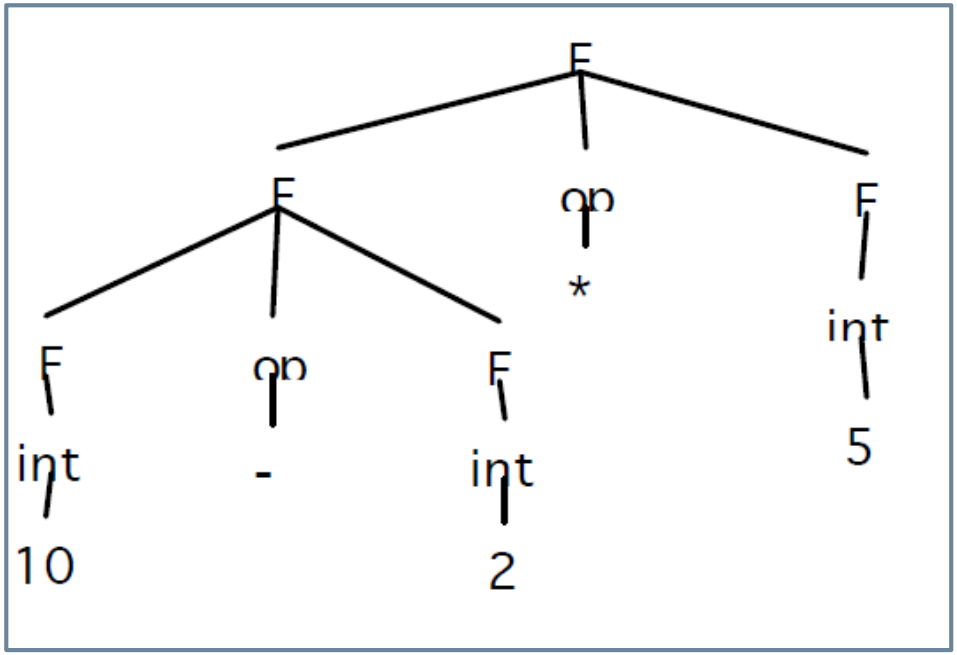
  E ::= E Op E | (E) | int

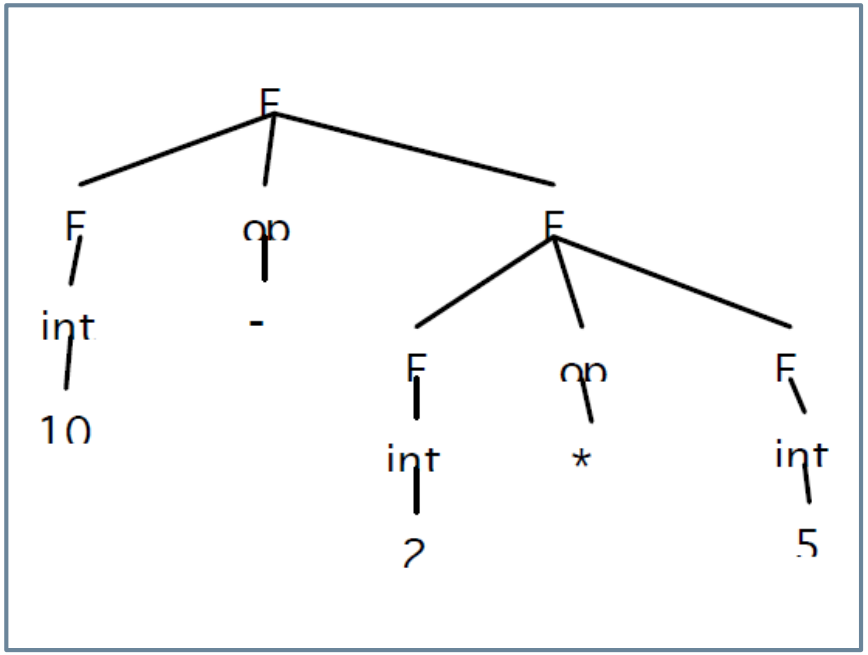
  Op ::= + | - | * | /

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Ambiguity (II)

- E.g. 10-2 *5





David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Ambiguity (III)

- Methods to handle ambiguity:
  - Rewrite the grammar unambiguously →only allow the one tree that correctly reflects our intention and eliminate the others.
  - Use the ambiguous grammar along with some disambiguating declarations.
- Example:

E ::= E T_Op E | T

T_Op ::= + | -

T ::= T F_Op T | F

F_Op ::= * | /

F ::= (E) | int

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Ambiguity (IV)

- There is no magical technique that can be used to resolve all varieties of ambiguity:
  - It is an undecidable problem to determine whether any grammar is ambiguous
  - To attempt to mechanically remove all ambiguity.

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Recursivity

- Productions are often defined in terms of themselves:

  variable_list –> variable | variable_list , variable

- G is recursive if $\exists\, A \in \Sigma_N \mid (A ::= xAy) \in P$
  - Left-recursive: $x=\lambda$ (A –> u | A v)
  - Right-recursive: $y=\lambda$ (A –> u | v A)

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Left Recursion (I)

- Some parsing methods (top-down) cannot handle left-recursive grammars
- To build this type of parsers, we need to transform the grammar to eliminate left-recursion

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Left Recursion (II)

- Eliminating left recursion (one production)

$P=(A::=A\cdot\alpha|\beta)$

1. $\Sigma_N = \Sigma_N \cup \{A'\}$
2. $P' = \{A::= \beta \cdot A', A'::= \alpha \cdot A' \mid \lambda\}$

- Example:
  - $P=\{ E ::= E * T \mid T\}$
  - Solution: $P' :=\{E ::= T E', E'::= * T E' \mid \lambda\}$

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Left Recursion (III)

- Eliminating the inmediate left recursion

$P=(A::=A\cdot\alpha_1|A\cdot\alpha_2|\ldots|A\cdot\alpha_n|\beta_1|\beta_2|\ldots|\beta_m|)$

1. $\Sigma_N = \Sigma_N \cup \{A'\}$
2. $P' = P \cup \{A::= \beta_1 \cdot A' | \beta_2 \cdot A' | \ldots | \beta_m \cdot A' | ,$
   $A'::= \alpha_1 \cdot A' | \alpha_2 \cdot A' | \alpha_n \cdot A' | \lambda\}$

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Left Recursion (IV)

- Example:
  - $P : E ::= E * T \mid E + T \mid T$
- Solution:
  - $P' : E ::= T\,E'$

    $E' ::= * T\,E' \mid + T\,E' \mid \lambda$

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Left Recursion (V)

- Recursion:
  - P: $A ::= B\ a$
    $B ::= A\ b \mid \lambda$
- Algorithm

1. Arrange the nonterminals in some order: $A_1, A_2, .., A_n$
2. For i=1 to n
   3. For j=1 to n
      replace each production of the form: $A_i \rightarrow A_j \cdot \gamma$
      by $A_i \rightarrow \delta_1 \cdot \gamma \mid \delta_2 \cdot \gamma \mid ... \mid \delta_k \cdot \gamma$
      where $A_j \rightarrow \delta_1 \mid \delta_2 \mid ... \mid \delta_k$ are all the current A_j productions
   4. Eliminate the inmediate left recursion in A_i

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Left factoring (I)

- Left factoring is a grammar transformation that is useful for producing a grammar suitable for a type of parsers (predictive parsing).

- **Objective**: Eliminate productions that have common first symbol(s) on the right side of the productions:

  Stmt → if Cond then Stmt else Stmt | if Cond then Stmt | Other | ....

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Left Factoring (II)

- For productions of the form:
  - P: $A ::= \beta \cdot \alpha_1 \mid \beta \cdot \alpha_2$
- Algorithm

1. $\Sigma_N = \Sigma_N \cup \{A'\}$
2. $P' = P \cup \{A ::= \beta \cdot A',$

$A' ::= \alpha_1 \mid \alpha_2\}$

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Left Factoring (III)

- Example:
  - $P : S \rightarrow \textbf{\textit{if}}\ C\ \textbf{\textit{then}}\ S\ \textbf{\textit{else}}\ S$

    $S \rightarrow \textbf{\textit{if}}\ C\ \textbf{\textit{then}}\ S$

    $S \rightarrow \textbf{\textit{repeat}}\ S\ \textbf{\textit{until}}\ C$

    $S \rightarrow \textbf{\textit{repeat}}\ S\ \textbf{\textit{forever}}$
- Solution:
  - $P' : S \rightarrow \textbf{\textit{if}}\ C\ \textbf{\textit{then}}\ S\ A'$

    $S \rightarrow \textbf{\textit{repeat}}\ S\ A''$

    $A' \rightarrow \textbf{\textit{else}}\ S \mid \lambda$

    $A'' \rightarrow \textbf{\textit{until}}\ C \mid \textbf{\textit{forever}}$

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Hidden left-factors and hidden left recursion (I)

- A grammar may not appear to have left recursion or left factors:
  - A –> da | acB
    B –> abB | daA | Af
  - A –> da | acB
    B –> abB | daA | daf | acBf
    
  - A –> da | acB
    B –> aM | daN
    M –> bB | cBf
    N –> A | f

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Hidden left-factors and hidden left recursion (II)

- A grammar may not appear to have left recursion or left factors:
  - S –> Tu | wx
    T –> Sq | vvS

    S into T

  - S –> Tu | wx
    T –> Tuq | wxq | vvS

  - S –> Tu | wx
    T –> wxqT' | vvST'
    T' –> uqT' | ε

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Push-Down Automata (I)

- For syntax analysis, we have to take into account the context.

$$M = (\Sigma, \Gamma, Q, A0, q0, f, F)$$

where:

- Σ is the alphabet of input symbols.
- Γ is the stack alphabet
- *Q is the set of states*
- *$A0 \in \Gamma$ is the initial symbol in the stack*
- *$q0 \in Q$ is the initial state*
- *$F \subseteq Q$ is the set of final states*
- *f is the transition function*

$$f : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \rightarrow P(Q \times \Gamma^*)$$

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Push-Down Automata (II)

- Language accepted by a Push Down Automaton:
  - Set of words that allow the transition from the initial state to a state in which both the stack and the input are empty.
  - Set of words that allow the transition from the initial state to one of the final states of the automaton

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Push-Down Automata (III)

- E.g. Push-Down automaton necessary to accept the language of words with the same number of 0s and 1s.
  - $\delta(q_0,0,Z_o)=\{(q_o,XZ_0)\}$
  - $\delta(q_0,0,X)=\{(q_o,XX)\}$
  - $\delta(q0,1,X)=\{(q_o,\varepsilon)\}$
  - $\delta(q_0,1,zo)=\{(q_o,YZ_0)\}$
  - $\delta(q_0,1,Y)=\{(q_o,YY)\}$
  - $\delta(q_0,0,Y)=\{(q_o,\varepsilon)\}$
  - $\delta(q_0,\varepsilon,z_o)=\{(q_o,\varepsilon)\}$

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es