AUTOMATA THEORY AND FORMAL LANGUAGES

2022-23

UNIT 4: LANGUAGES AND FORMAL GRAMMARS

José Luis Mira Peidro jmira@inf.uc3m.es
Universidad Carlos III de Madrid

Formal Grammars. Bibliography

- (AAM). Enrique Alfonseca Cubero, Manuel Alfonseca Cubero, Roberto Moriyón Salomón. Teoría de autómatas y lenguajes formales. McGraw-Hill (2007). Chapter 5.

- (HMU). John E. Hopcroft, Rajeev Motwani, Jeffrey D.Ullman. Introducción a la teoría de autómatas, lenguajes y computación (3rd edition). Ed. Pearson Addison Wesley.

- (AAM). Manuel Alfonseca, Justo Sancho, Miguel Martínez Orga. Teoría de lenguajes, gramáticas y autómatas. Publicaciones R.A.E.C. 1997. Chapter 3.

OUTLINE

- Definition of a grammar (notation)
  - Sentential form. Sentence
  - Equivalent grammars
  - Sentences and handles
  - Recursion
- Chomsky Hierarchy
  - Regular grammars (Type 3, G3) and equivalences
- Parse trees
  - Ambiguity
- Context-free grammars languages(Type 2, G2)
  - Language generated by a Type-2 Grammar
  - Well-formed grammars
  - Chomsky. Normal Form. Greibach Normal Form

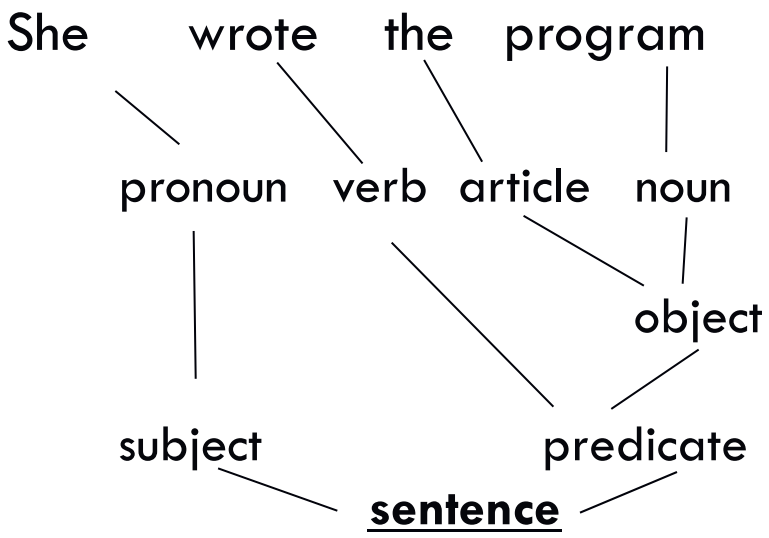
Formal Grammars. Introduction

- We need a **representation for languages**:
  - Valid for a finite or infinite number of elements.
  - Denote the structure of the words and the sentences in the language.
  - Valid for natural languages and programming languages.

Solution: GRAMMARS

Formal Grammars. Introduction

- Example: Syntax analysis of sentences in natural language



There are rules to define each one of the parts of valid sentences

Production rules

Formal Grammars. Introduction

- **Grammar:** Description of the structure of sentences and words in a language. → Rules to verify if a sentence is correct or not.
  - Sentence = Subject and Predicate
  - Subject = Noun Phrase
  - Noun Phrase = Nominal Group and Optional Descriptor
  - Nominal group: Optional Determiner + Noun
  - Descriptor: Adjective or Relative Pronoun + Subordinate Sentence
  - Predicate = Verb + Objects
  - Object = Direct, Indirect, Adverbial
  - Direct Object = Noun Phrase
  - Indirect Object: “a” Preposition + Noun Phrase
  - Adverbial Object: Preposition + Noun Phrase

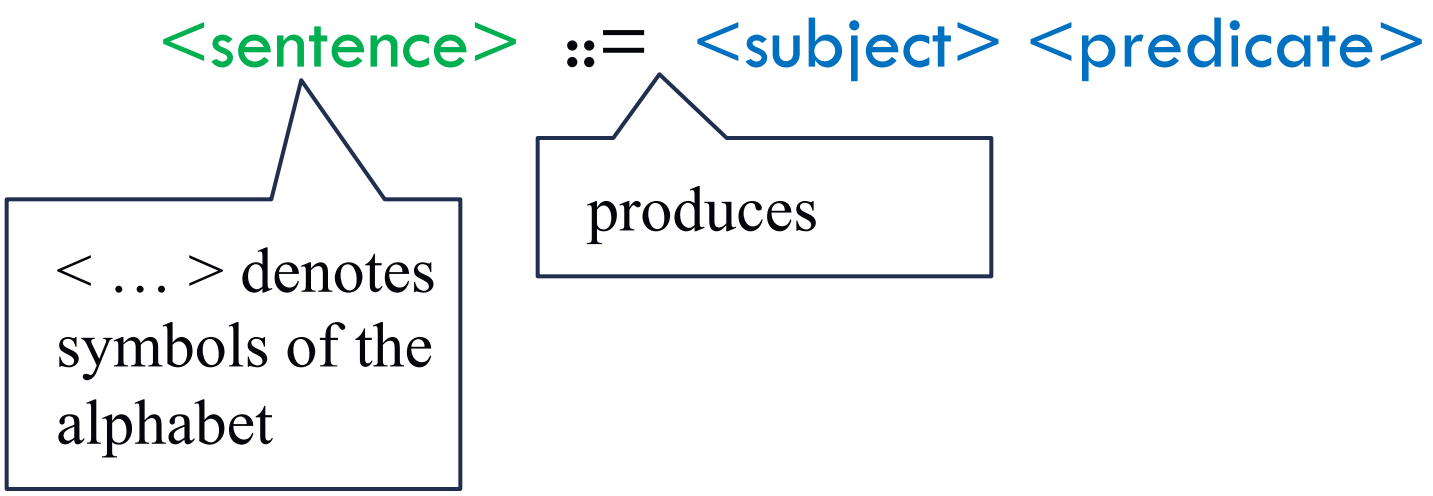
GRAMMAR
Set of structural rules governing the composition of clauses, phrases, and words in any given language

Formal Grammars. Introduction

- Production rule:

  The term to define: left side. Its definition is on the right side (of the rule).

  Example:



Formal Grammars. Notation

- **< element > :** Symbols of the grammar.
- Notation for **possible choices**:

  < noun phrase>::= <noum group> <descriptor>
  <noun phrase>::= <noun group>

  < noun phrase >::= <noun group> <descriptor> | <noun group>

- **Values**:

  <noun> ::= John | Mary” | “house” | “student” | “thing”

- **Recursive rules**:
  - <objects> ::= <object> <objects>

Formal Languages. Derivations rules

- **Productions, writing rules or derivation rules:**
  - Let Σ be an alphabet.
  - A production is an **ordered pair** (x,y) where $x, y \in \Sigma^*$
  - x = left side of the production
  - y = right side of the production
  - It is represented by: x::= y

Formal Languages. Derivations rules

- **Direct derivation:**
  - Let Σ be an alphabet and P a set of productions over Σ
  - Let v and w be two words over Σ, v, w $\in \Sigma^*$
  - We can say { "w is direct derivation of v" / "v directly produces w" / "w reduces directly to v" } $v \rightarrow w$

If there are two words z, u $\in \Sigma^*$ that fulfill:

$v = z \cdot x \cdot u \qquad w = z \cdot y \cdot u$

(x::=y) $\in$ P

17.I

Formal Languages. Derivations rules

- **Direct derivation, examples:**
  - Let Σ be the Spanish alphabet of uppercase letters and ME::=BA is a production over Σ
  - CABALLO is a direct derivation of CAMELLO (CAMELLO directly generates CABALLO)
  - Using the production CA::=PE over Σ, PERA is a direct derivation of CARA

It does not really happen like that in most of the languages (specific roots)

Formal Languages. Derivations rules

- **Derivation of lenght n:**
  - Let Σ be an alphabet and P a set of productions over Σ
  - Let v and w be two words over Σ, v, w ∈Σ*
  - We can say { "w is a derivation of v" / "v produces w" / "w is reduced to v" } $v \overset{+}{\rightarrow} w$

In several steps

If there are a finite sequence of words, $u_0, u_1, u_2, \ldots u_n$ $(n>0) \in \Sigma^*$ that fulfill

$v = u_0$

$u_0 \rightarrow u_1$

$u_1 \rightarrow u_2$

.....

$u_{n-1} \rightarrow u_n$

$w = u_n$

Derivation of lenght n

17.II

Formal Languages. Derivations rules

- **Derivation, example:**
  - Given the alphabet Σ = {0,1,2,N,C} and the set of productions over this alphabet, P = {N::CN, N::=C, C::=0, C::=1, C::=2}
  - Indicate the lenght of the derivation

    $N \overset{+}{\rightarrow} 210$

    $N \rightarrow CN \rightarrow CCN \rightarrow CCC \rightarrow 2CC \rightarrow 21C \rightarrow 210$

COROLLARY: if $v \rightarrow w$ then $v \overset{+}{\rightarrow} w$ by means of a derivation with lenght = 1

Formal Languages. Derivations rules

- **Thue Relationship:**
  - Let Σ be an alphabet and P a set of productions over Σ
  - Let v and w be two words over Σ, v, w $\in \Sigma^*$
  - There is a Thue Relationship among v and w represented by **v $*\rightarrow$ w** if:
    - **v $+\rightarrow$ w**
    - **v = w**

    There is a derivation of lenght n or there are the same words
  - Properties:
    - Reflexion
    - Transitive

Formal Grammars. Definition

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

Formal Grammars. Definition

- Formal entity to specify (in a finite notation) the set of strings with symbols that make up a language.
  - Σ_T All the strings in the language L(G) represented by G are made with this alphabet symbols.
  - Σ_N Set of auxiliary symbols introduced for the definition of G but not included in the strings of L(G).
  - **S** Initial symbol from which the production rules P are applied.
  - **P** Set of production rules:

$$\mathbf{u::=v} \text{ where } u \in \Sigma^+ \text{ and } v \in \Sigma^* \; u = xAy$$

$$\text{where } x, y \in \Sigma^* \text{ and } A \in \Sigma_N$$

Formal Grammars. Introduction

- **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 symbol in a 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.

Formal Grammars. Examples

- $G=\{\Sigma_T, \Sigma_N, S, P\}$

$\Sigma_T = \{x,y,z\}$

$\Sigma_N = \{S,A,B\}$

$P = \{S ::= AB,$

$A ::= Ax \mid y,$

$B ::= z\}$

Formal Grammars. Backus Notation

- Formal notation to describe the syntax of a given language.

  ::= means "is defined as"

  | means "or"

  If u::=v and u::= w are two production rules included in P, then we can write u ::= v | w

- Nonterminal symbols are usually represented by uppercase letters.
- Terminal symbols by lowercase letters.

Formal Grammars. Backus Notation

Grammar not in Backus Notation:

G = ({0,1}, {N,C}, N,P)

P = {N ::= NC,

N ::= C,

C ::= 0,

C ::= 1}

Grammar in Backus Notation:

G = ({0,1}, {N,C}, N,P)

P = {N ::= NC | C,

C ::= 0 | 1}

Formal Grammars. Introduction

A Spanish grammar using Backus notation

<sentence> ::= <subject><predicate>
<subject> ::= <noun phrase>
< noun phrase >::= <noun group> <descriptor>
< noun phrase >::= <noun group>
< noun phrase > ::= <determiner><noun>
< noun phrase > ::= <noun>
<descriptor> ::= <adjective>
<descriptor> ::= <relative pronoun> <subordinate sentence>
<predicate>::= <verb> <objects>
<objects> ::= <object> <objects>
<objects> ::= λ
<object> ::= <direct>
<object> ::= <indirect>
<object> ::= <adverbial>
<direct> ::= < noun phrase>
<indirect> := “a” < noun phrase>
<adverbial> ::= <preposition> <noun phrase>

Formal Grammars. Introduction

A grammar for a programming language using Backus notation

```
<statements ::= <statement> “;” <statements> | <statement>
<statement ::= <assignment> | <conditional> | <iteration>
<assignment ::= <variable> “:=“ <expresion>
<condicional ::= “if” <condition> “then” <statements>
   “else” <statements>
<iteration ::= “while” <condition> “do” <statements>
<expresion ::= <variable-number> “+” <expresion> |
               <variable-number> “*” <expresion> |
               <variable-number> “-” <expresion> |
               <variable-number> “/” <expresion> |
               <variable-number>
<variable> ::= [A-Za-z] [A-Za-z0-9]*
<variable-number> ::= <variable> | <number>
<number> ::= [0-9]+
```

Formal Grammars. Definitions

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

17.III

Formal Grammars. Definitions

- **Sentential form:**

Given a grammar $G=\{\Sigma_T, \Sigma_N, S, P\}$

A word $x \in (\Sigma_T \cup \Sigma_N)^*$ is a **sentential form** of G if:

$$S^* \rightarrow x$$

Explanation:

- There is a Thue relation between the axiom and x:
  - x is the axiom.
  - Using the production rules P, it is possible to obtain x from a set of derivations beginning with S.

Formal Grammars. Definitions

- **Sentences or Words generated by the grammar:**

  Every sentential form x of a grammar that fulfills

  $$x \in \Sigma_T^*$$

  Example N * → 210

Explanation:

- Words that are generated by the grammar.
- Sentential forms that only contain terminal symbols.

Formal Grammars. Definitions

- **Language associated to a grammar:**

Given a grammar $G=\{\Sigma_T, \Sigma_N, S, P\}$

The language generated/associated/described by G is

$L(G) = \{x \mid S \rightarrow^* x$ AND $x \in \Sigma_T^*\}$

Explanation:

- All the sentences generated by G.
- Motivation: Grammar Theory and Backus Representation

  →Notation to represent and describe languages.

Formal Grammars. Language Example

- Which is the language described by G in the following example?

G = ({0,1,2,3,4,5,6,7,8,9}, {N,C}, N,P)

P = {N ::= NC | C, C ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}

17.IV

Formal Grammars. Definitions

- **Equivalent Grammars:**

Two grammars G1 and G2 are equivalent if they describe / generate the same language:

G1 ≈ G2

If

L(G1) = L(G2)

Formal Grammars. Definitions

- **Sentence of a sentential form with regard to a nonterminal:**

Given a grammar $G=\{\Sigma_T, \Sigma_N, S, P\}$

Let v= xuy be a sentential form of G

u is a sentence of the sentential form **v** with regard to a nonterminal symbol U ($U \in \Sigma_N$) if:

$$S \xrightarrow{*} xUy$$

$$U \xrightarrow{+} u$$

Explanation:

- U can be replaced for u by means of one or several derivations in the sentential form xUy to generate the sentential form v, remaining x and y.

Formal Grammars. Definitions

- **Simple Sentence:**

Given a grammar $G=\{\Sigma_T, \Sigma_N, S, P\}$

v= xuy sentential form of G

u is a simple sentence if:

$S \xrightarrow{*} xUy$

$U \rightarrow u$ (direct derivation)

Explanation:

- U can be replaced for u by means of only one derivation.

Formal Grammars. Definitions

- **Handle:**

  The handle of a sentential form v is the simple sentence of v situated at left.

COROLLARY:

- If the grammar is not ambiguous, then there is only one handle.

Formal Grammars. Sentences Example

- **Exercise (Alfonseca page 50)**

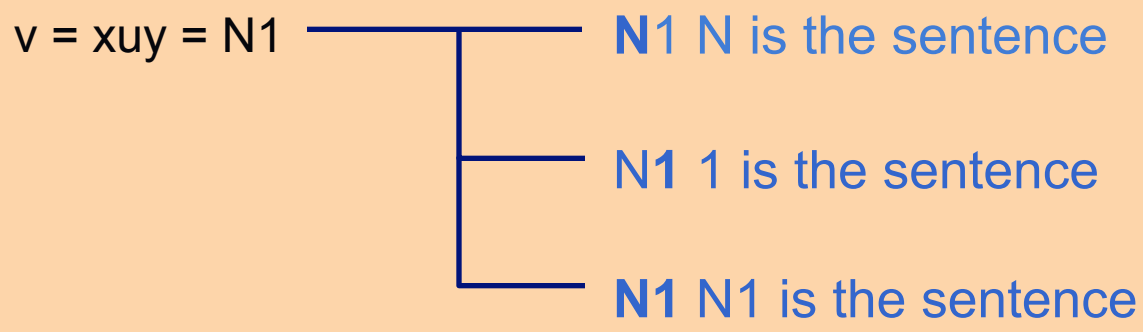
  In the Grammar describing natural numbers + 0, demonstrate that N is not a sentence of N1. Find all the sentences of N1. What are the simple sentences? What is the handle?

  G = ({0,1,2,3,4,5,6,7,8,9}, {N,C}, N,P)
  P = {N ::= NC | C, C ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}

  and in the sentential form 2C0?

Formal Grammars. Sentences Example

- **Exercise (Alfonseca page 50)**



Formal Grammars. Definitions

- **Recursion:**

A production rule is recursive if:

$$U \rightarrow x\ U\ y$$

- Left recursive production: $x = \lambda$ ($U \rightarrow U\ y$)
- Right recursive production: $y = \lambda$ ($U \rightarrow xU$)

Formal Grammars. Definitions

- **Recursion:**

Given a grammar $G=\{\Sigma_T, \Sigma_N, S, P\}$

It is recursive in the nonterminal $U \in \Sigma_N$ if

$$U \xrightarrow{+} xUy$$

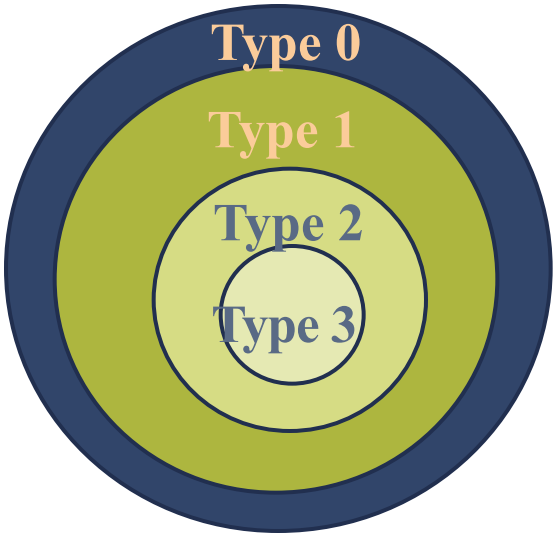
- Left recursive grammar → x = λ
- Right recursive grammar → y = λ
- If a language is infinite, then the grammar that represents it must be recursive.

Formal Grammars. Chomsky Hierarchy

- **Type 0**: unrestricted grammars
  - **u** ::= **v** (u not **λ**). At least a non-terminal in **u**
- **Type 1**: context-sensitive grammars
  - $P=\{xAy ::= xvy \text{ where } x,y\in\Sigma^{*} \wedge A\in\Sigma_N \wedge v\in\Sigma^{+}\}$ S → **λ** allowed
- **Type 2**: context-free grammars
  - $P=\{A::=v \text{ where } A\in\Sigma_N \wedge v\in\Sigma^{+}\}$ S → **λ** allowed

Formal Grammars. Chomsky Hierarchy

- **Type 3**: regular or linear grammars
  - X ::=a, X ::= aY, S ::= λ allowed
  - 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
    - $P = \{(A ::= aB) \cup (A ::= a) \mid A, B \in \Sigma_N \wedge a \in \Sigma_T\}$



Formal Grammars. Chomsky Hierarchy

Class	Grammars	Languages	Automaton
Type-0	Unrestricted	Recursively enumerable (Turing-recognizable)	Turing machine
Type-1	Context-sensitive	Context-sensitive	Linear-bounded
Type-2	Context-free	Context-free	Pushdown
Type-3	Regular	Regular	Finite

Formal Grammars. Chomsky Hierarchy (G0)

- Languages defined by Type-0 grammars are accepted by Turing Machines: Recursively enumerable languages.
- More generic grammars (all possible grammars). Types:
  - (a) unrestricted grammars and
  - (b) phrase-structured.
- Rules are of the form:
  - $u ::= v$,
  - *where u and v are* arbitrary strings over an alphabet Σ *and* $u \neq \lambda$

Formal Grammars. Chomsky Hierarchy (G0)

- Restriction: $\lambda ::= v \notin P$

- Sentential form:

$$u = xAy,\ x,y \in \Sigma^*,\ A \in \Sigma_N$$

- Example:

$$G = \{(0,1), (S), S, P\}, \text{ where:}$$

$$P = \{(S ::= 000S111), (0S1 ::= 01)\}$$

Formal Grammars. Chomsky Hierarchy (G0)

- **Phrase-structured grammars:**
  - $(xAy ::= xvy) \in P$, where:

    $x,y \in \Sigma^*$, $A \in \Sigma_N$, $v \in \Sigma^*$

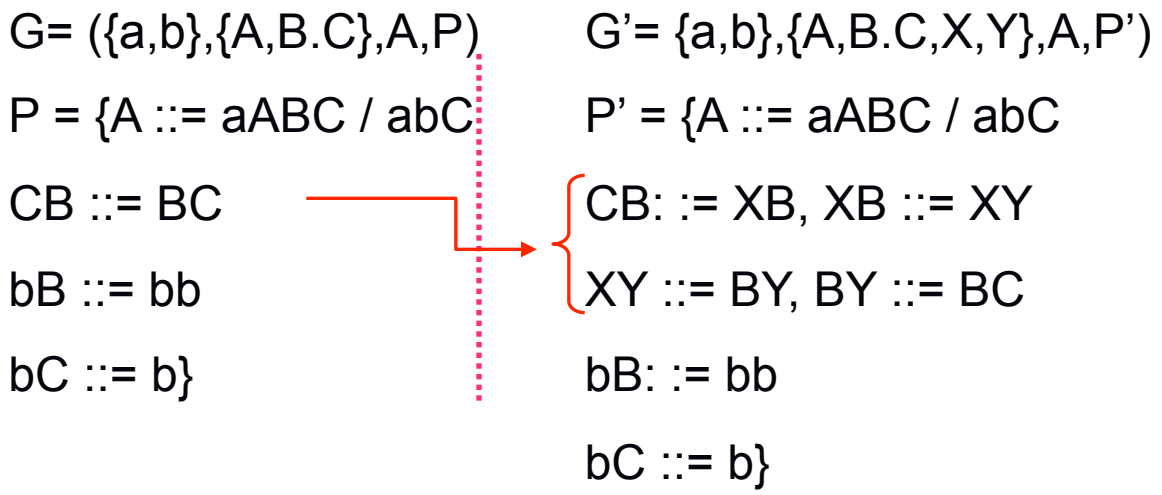
- If $v=\lambda$, then:

  $xAy ::= xy$ compressor rules

- Compressor Grammar: Contains at least a compressor rule.

Formal Grammars. Chomsky Hierarchy (G0)

- COROLLARY: : Every L(G0) can be described by means of a **Phrase-structure** G0 grammar.



Formal Grammars. Chomsky Hierarchy (G1)

- Languages defined by Type-1 grammars are accepted by linear-bounded automata: Context-sensitive languages.

- Syntax of some natural languages (Germanic).

- Rules are of the form:
  - $xAy ::= xvy$

  where:
  - $x,y \in \Sigma^*$, $A \in \Sigma_N$
  - $v \in \Sigma^+$ Compressing rules are not allowed.
  - *Exception:* $S \rightarrow \lambda$ can be included in P.

19.III

Formal Grammars. Chomsky Hierarchy (G1)

- Examples :

- Grammar that is not G1:

$$G= (\{a,b\}, \{S\}, S, P)$$

$$P= \{S ::= aaaaSbbbb, aSb ::= ab\}$$

- Grammar that is G1:

$$G= (\{a,b\}, \{S\}, S, P)$$

$$P= \{S ::= aaaaSbbbb, aSb ::= abb\}$$

Formal Grammars. Chomsky Hierarchy (G1)

- **Non-decreasing property in G1:**

  Strings obtained using the derivations of a G1 are not decreasing length, that it is to say:

  $$\mathbf{u ::= v} \Rightarrow |\mathbf{v}| \geq |\mathbf{u}|$$

  the length of the right side in the production is greater than or equal to the length of the left side

  - Demonstration:

    $\alpha A \beta ::= \alpha \gamma \beta$

    where $\gamma \in (\Sigma_T \cup \Sigma_{NT})^+$ due to the definition of G1 (no compressing rules)

    that it is to say, $\gamma \neq \lambda$ always, and then $|\gamma| \geq 1$

    But $|A| = 1$, so $\mathbf{u ::= v} \Rightarrow |\mathbf{v}| \geq |\mathbf{u}|$

Formal Grammars. Chomsky Hierarchy (G2)

- Languages defined by Type-2 grammars are accepted by push-down automata: context-free languages.
- Natural languages are almost entirely definable by type-2 tree structures.
- Rules are of the form:
  - $A ::= v$ *(only one NT symbol on the left)*

  where
  - $A \in \Sigma_N$ ($S ::= \lambda$ can appear)
  - $v \in \Sigma^*$
- $\forall$ L(G2) $\exists$ L(G2') without rules $A ::= \lambda$ ($A \neq S$)

Formal Grammars. Chomsky Hierarchy (G2)

- r ∈ P have **only one nonterminal in the left side.**
- **(S ::= λ) ∈P** and **(A ::= λ) ∉P**

  (algorithm for cleaning rules that are not generative)
- Example:

$$\mathbf{G= \{(a,b), (S,A),S, P\}}$$

$$P=\{(S ::= aS,\ S ::= aA,\ A ::= bA,\ A ::= b)\}$$

- **Context-Free:** A can be changed for v in any context.

Formal Grammars. Chomsky Hierarchy (G2)

- The syntax of most programming languages is context-free (or very close to it).
- EBNF / ALGOL 60…
- Due to memory constraints, long-range relations are limited.
- Common strategy: a relaxed CF parser that accepts a superset of the language, invalid constructs are filtered.
- Alternate grammars proposed: indexed, recording, affix, attribute, van Wijngaarden (VW).

Formal Grammars. Chomsky Hierarchy (G3)

- Languages defined by Type-3 grammars are accepted by finite state automata: regular languages.
- Most syntax of some informal spoken dialog.
- *Two types:*
  - **left-linear** or left regular grammars.
  - **right-linear** or right regular grammars.

Formal Grammars. Chomsky Hierarchy (G3)

- **left-linear** or left regular grammars:
  - **A ::= a**
  - **A ::= Va**
  - **S ::= λ**
  - **right-linear** or right regular grammars
  - **A ::= a**
  - **A ::= aV**
  - **S ::= λ**
- Where
  - $a \in \Sigma_T$
  - $A, V \in \Sigma_N$, S is the axiom

Formal Grammars. Chomsky Hierarchy (G3)

- r ∈ P have only one nonterminal symbol in the left side and the right side begins with a terminal symbol followed or not by nonterminal symbols (**right-linear)**

- Example:

$$G= (\{a,b\}, \{S,A\},S, P),$$

$$P=\{S ::= aS, S ::= aA, A ::= bA, A ::= b\}$$

Formal Grammars. Chomsky Hierarchy (Examples)

- **Examples of Grammars** (Isasi, Martínez and Borrajo, pages16 -19)

  G_1= ({0,1}, {A,B,S},S, P), P = {S::= A0, A0::= 1B1, 1A::= 0B0, B ::= λ | 1 | 0}

  G_2= ({0,1}, {A,B},A, P), P = {A ::= 1B1 | 11, 1B1 ::= 101 | 111}

  G_3= ({0,1}, {A,B},A, P), P = {A ::= 1B1 | 11, B ::= 0 | 1}

  G_4= ({0,1}, {A,B, C},A, P), P = {A ::= 1B, B ::=1 | 0C | 1C, C ::= 1)}

Formal Grammars. Chomsky Hierarchy (Examples)

- **Examples of Grammars** (Isasi, Martínez and Borrajo, pages16 -19)

  G_1= ({0,1}, {A,B,S},S, P), P={S ::= A0, A0::= 1B1, 1A ::= 0B0, B ::=λ | 1 | 0}

  B::=λ is a compressing rule.

  A0::= 1B1 and 1A::= 0B0 do not keep the context.

  It is a **G0** without restrictions.

  L = {11, 101, 111}

  Does this simple language need a grammar of the highest level in the Chomsky Hierarchy?

Formal Grammars. Chomsky Hierarchy (Examples)

- **Examples of Grammars** (Isasi, Martínez and Borrajo, pages16 -19)

  G_2= ({0,1}, {A,B},A, P), P={A ::= 1B1 | 11, 1B1 ::= 101 | 111}

  No compressing rules.

  1B1::= 101 / 111 keep the context.

  It is a **G1**.

  L = {11, 101, 111}

  Does this simple language need a G1grammar in the Chomsky Hierarchy?

Formal Grammars. Chomsky Hierarchy (Examples)

- **Examples of Grammars** (Isasi, Martínez and Borrajo, pages16 -19)

  G_3= ({0,1}, {A,B},A, P), P={A ::= 1B1 | 11, B ::= 0 | 1}

No compressing rules.

Only one nonterminal symbol in the left side.

Freedom in the right side → G that is context-free **G2**.

L = {11, 101, 111}

Does this simple language need a G2 grammar in the Chomsky Hierarchy?

Formal Grammars. Chomsky Hierarchy (Examples)

- **Examples of Grammars** (Isasi, Martínez and Borrajo, pages16 -19)

  G_4= ({0,1}, {A,B, C},A, P), P={A ::= 1B, B ::= 1 | 0C | 1C, C ::= 1)}

No compressor rules.

Only one symbol in the left side.

No freedom in the right side → G right-linear **G3 RL**.

L = {11, 101, 111}

Does this simple language need a G3 grammar in the Chomsky Hierarchy? YES

Formal Grammars. Chomsky Hierarchy

- A language is called type i (i= 0, 1, 2, 3) if there is a grammar G type i that fulfills L= L(Gi)

- Hierarchy: inclusion relationship.

$$G3 \subset G2 \subset G1 \subset G0$$

Formal Grammars. Chomsky Hierarchy (Examples)

- Obtain a G1 grammar for the language $L=\{a^n b^n c^n\}$
- Obtain a grammar given the alphabet $\Sigma=\{a,b,c,d\}$ and the language L={strings that do not contain the sequence bc}
- Obtain a grammar for the language $L=\{a^n b^m c^p a^q b^n \mid q=p+m;\ n,m\geq 1;\ p\geq 0\}$
- Given the following productions of different grammars, explain which is the associated language:

P_1: **S ::=λ | A**
A::=AA | c

P_2: **S::=λ | A**
A::=cAd | cd

P_3: **S::=λ | A**
A::=AcA | c

P_4: **S::=cA**
A::=d | cA | Td
T::=Td | d

P_5: **S::=λ | A**
A::=Ad | cA | c | d

- Obtain the language generated by the grammar ({0,1}, {S,A,B,C},S, {S ::= BAB, BA ::= BC, CA ::= AAC, CB ::= AAB, A ::= 0, B ::= 1})

Formal Grammars. Chomsky Hierarchy (Examples)

- Obtain a grammar for the language $L=\{x^n y^m z^k \mid m, n, k \geq 0, k=m+n\}$
- Obtain a grammar for a language with $\Sigma=\{0,1\}$ $N(0)=N(1)+1$
- Given the grammar $G = (\Sigma_T, \Sigma_N, S, P)$, with:

$\Sigma_T = \{0,1,2,3,4,5,6,7,8,9\}$
$\Sigma_N = \{S, D, U\}$
$P = \{$
S ::= 0 | 3 | 6 | 9 | 0S | 3S | 6S | 9S | 1D | 4D | 7D | 2U | 5U | 8U
D ::= 2 | 5 | 8 | 2S | 5S | 8S | 1U | 4U | 7U | 0D | 3D | 6D | 9D
U ::= 1 | 4 | 7 | 1S | 4S | 7S | 2D | 5D | 8D | 0U | 3U | 6U | 9U
}

Describe the language that is generated.

Formal Grammars. Chomsky Hierarchy (Examples)

(Hopcroft-Ulman,1979)

- **L = {a^i / i is multiple of 2} described by a G0 with phrase-structure:**
  G = ({a}, {S,A,B,C,D,E}, S, P)
  P = {
  S ::= ACaB
  Ca ::= aaC
  CB ::= DB
  CB ::= E
  aD ::= Da
  AD ::= AC
  aE ::= Ea
  AE ::= λ
  }

Is it well classified? Why?

Formal Grammars. Chomsky Hierarchy (Examples)

(Hopcroft-Ulman,1979)

- **G = ({a}, {S,A,B,C,D,E}, S, P)**
  P = {
  S ::= ACaB
  Ca ::= aaC
  CB ::= DB
  CB ::= E
  aD ::= Da
  AD ::= AC
  aE ::= Ea
  AE ::= λ
  }

Due to these rules, it is not really a phrase-structure grammar, but it can be transformed to one of these grammars (G0 without restrictions)

Formal Grammars. Chomsky Hierarchy (Examples)

(Hopcroft-Ulman,1979)

- **L = {0^n 1^n / n>=1} described by the grammar G: G = ({0,1}, {S,A,B}, S, P)**
  P = {
  S ::= 0A
  A ::= 0AB
  A ::= 1
  B ::= 1}

- **L = {0^i 1^j 2^k/ i=j or j=k} described by the grammar G: G = ({0,1,2}, {S,A,B,C}, S, P)**
  P = {
  S ::= AB
  S ::= CD
  A ::= 0A1
  A ::= λ
  B ::= 2B
  B ::= λ
  C ::= 0C
  C ::= λ
  D ::= 1D2
  D ::= λ}

G2 grammars: a single symbol to the left and freedom in the right symbols.

Formal Grammars. Regular grammars (Equivalences)

EQUIVALENT GRAMMARS

1. ∀ G3 right-linear grammar with rules A::= aS

   ∃ A right-linear grammar G3’ equivalent without rules A::= aS

2. ∀ G3 right-linear

   ∃ a G3’ left-linear equivalent grammar.

Formal Grammars. Regular grammars (Equivalences)

1. **∀ G3 right-linear grammar with rules A::= aS**

   ∃ An equivalent right-linear G3' grammar without rules A::= aS

   Transformation Procedure:

   1. Eliminate rules with S in the right side of the productions
   2. Incorporation of a new symbol in the alphabet Σ_N
   3. ∀ S::= x, where $x \in \Sigma^+$, a rule B::= x is added.
   4. The rules A::= aS are transformed into A::= aB rules.
   5. Rule S::= λ is not affected by this algorithm.

Formal Grammars. Regular grammars (Equivalences)

Transformation Procedure:

1. Eliminate rules with S on the right side of the productions
   - Create a new NT symbol B (for example)
   - Create a new rule B::=x ∀S::= x
   - Change S in the body of all productions by the new symbol.

Example:

P={S::=1 | 1A, A::=0S}

New B

B::=1, B::=1A, ~~A::=0S~~ A::=0B

P'={S::=1 | 1A, B::=1 | 1A, A::=0B}

Formal Grammars. Regular grammars (Equivalences)

2. **∀ G3 right linear ∃ an equivalent G3' left linear**

Transformation Procedure. *First eliminate S on the right side of productions.*

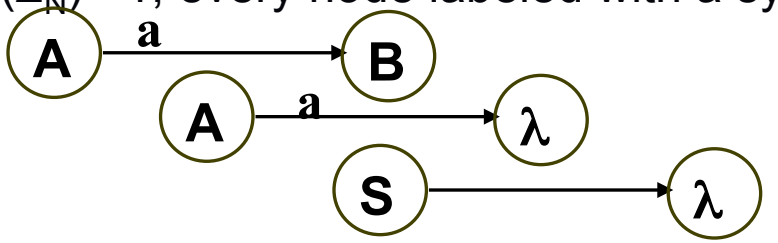
1. **Creation of a graph:**

a) Number of nodes= C(Σ_N) +1, every node labeled with a symbol in Σ_N and another with λ

b) every A::=aB ∈P

c) every A::=a ∈P

d) if S::= λ ∈P



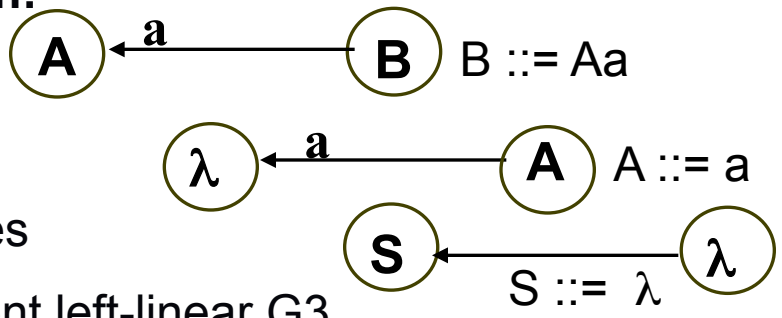
2. **Transformation of the previous graph:**

a) Exchange labels of S and λ

b) invert the direction of the arcs

c) Undo the path and generate the new rules

⇒ interpret the graph to obtain the equivalent left-linear G3.



Formal Grammars. Regular grammars (Equivalences)

- **Exercise (Alfonseca pg 58)** Given the right-linear G3 grammar,

  G = ({0,1}, {S,B}, S, P), where

  P = {S ::= 1B |1, B ::= 0S}, calculate the equivalent left-linear G3 grammar.

  Demonstrate that they generate the same language.

20 (bis) III & IV

OUTLINE

- Definition of a grammar (notation)
  - Sentential form. Sentence
  - Equivalent grammars
  - Sentences and handles
  - Recursion
- Chomsky Hierarchy
  - Regular grammars (Type 3, G3) and equivalences
- **Parse trees**
  - **Ambiguity**
- Context-free grammars languages(Type 2, G2)
  - Language generated by a Type-2 Grammar
  - Well-formed grammars
  - Chomsky. Normal Form. Greibach Normal Form

Formal Grammars. Representations

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

Formal Grammars. Representations (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 derivation:
  - the rightmost nonterminal symbol is always replaced when the rules are applied.

Formal Grammars. Representations (Derivation)

S ::= AB

A ::= Ax | y

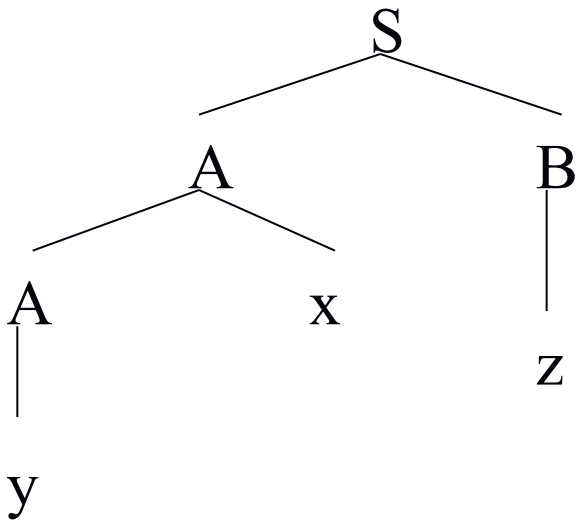
B ::= z

Leftmost derivation:

S → AB → AxB → yxB → yxz

Formal Grammars. Representations (Parse tree)

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



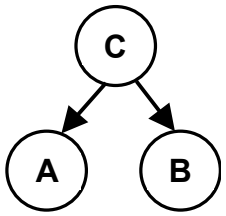
Formal Grammars. Parse trees

- The derivations of the grammars G type 1, 2 and 3 can be represented by a hierarchical structure called **parse tree.**

- It represents productions used for the generation of a word, that it is say, the structure of the words according to G.

Formal Grammars. Parse trees

- It is an ordered and labeled tree that is build:
  - The root is denoted by the axiom of G.
  - A direct derivation is represented by a set of branches coming out of a given node (left side of the P).
  - By the application of a rule → a symbol on the left side is replaced by a word *u* on the right side. For each symbol in u, a branch is drawn from a nonterminal to this symbol: e.g. Given a rule u=AB and the production P: C→u



The leftmost symbol in a P, it is also leftmost situated in the tree.

Formal Grammars. Parse trees

- **In a G1, the tree must also retain the context information:**
- For each branch:
  - the starting node is called **parent** of the end node.
  - an end node is called **child** of the parent node.
  - two child nodes of the same parent are called **brothers**.
  - a node is an **ascendant** of another if it is its parent node or an ascendant of its parent node.
  - a node is a **descendant** of another if it is its child or a descendant of its child nodes.

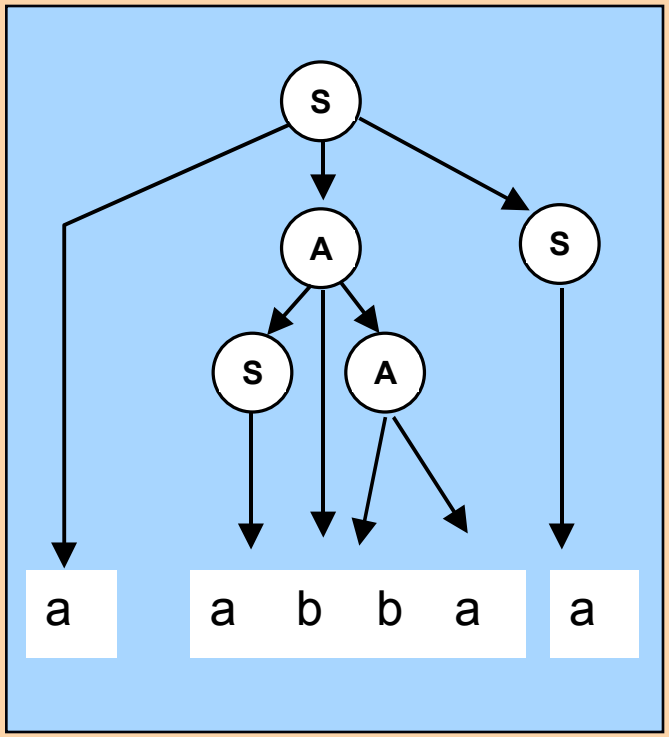
Formal Grammars. Parse trees

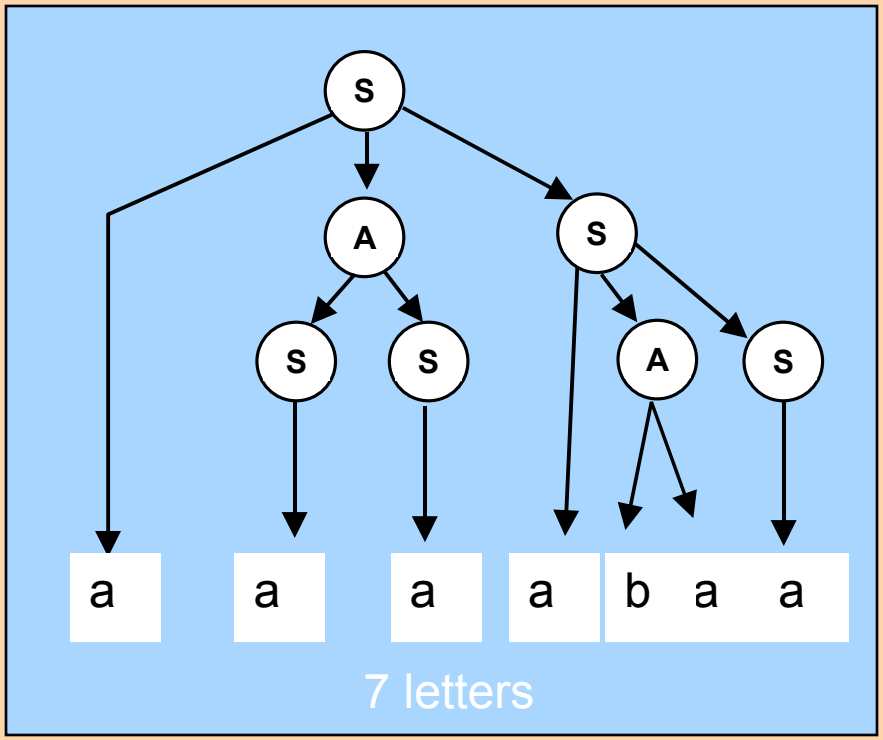
- In the process of construction of the tree, the end nodes of each successive step, read from left to right, give us the **sentential form** obtained by the derivation represented by the tree.

- The set of the leaves of the tree (nodes denoted by terminal symbols or λ) read from left to right give us the **sentence** generated by the derivation.

Formal Grammars. Parse trees

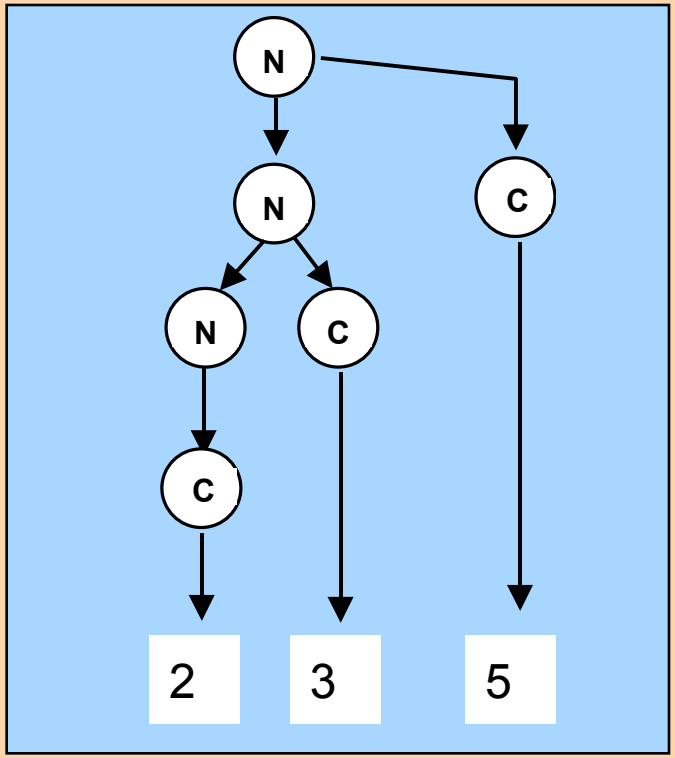
- Given G = ({a,b}, {A,S}, S, {S ::= aAS | a, A ::= SbA | SS | ba}). Find a parse tree for a sentence with 6 letters.





Formal Grammars. Parse trees

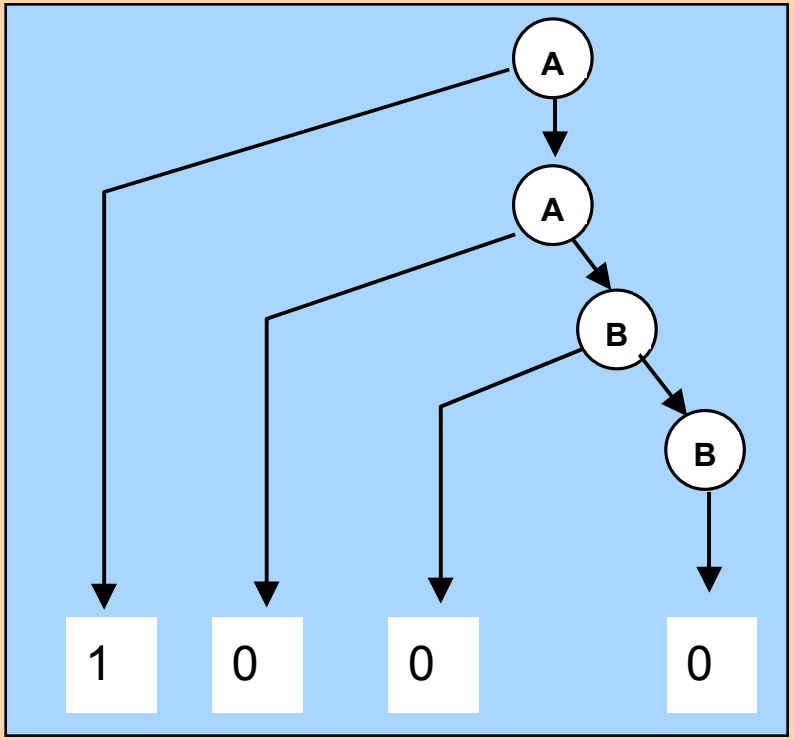
- Given the grammar G = ({0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, {N,C}, N,
- {N ::= NC | C, C ::= 0 | ... | 9}) Calculate a parse tree for N → 235



Formal Grammars. Parse trees

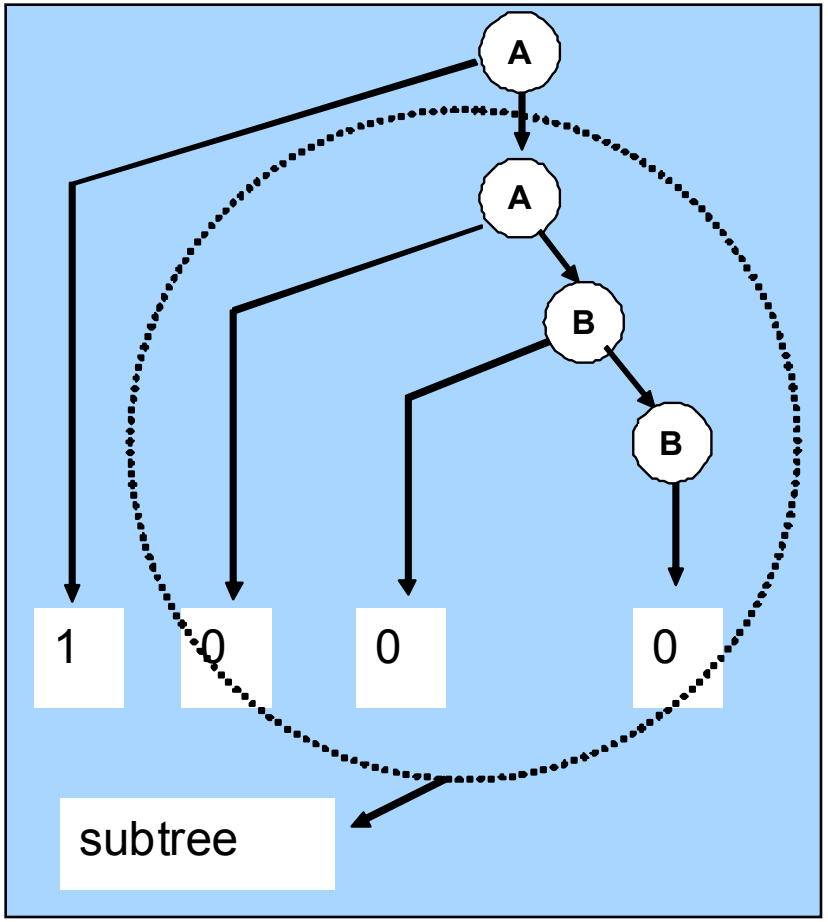
- Given G = ({0,1}, {A,B}, A, {A ::= 1A | OB, B ::= 0B | 0}) One of its valid derivations is: A → 1A → 1OB → 100B → 1000

  Calculate a parse tree for the derivation A → 1000



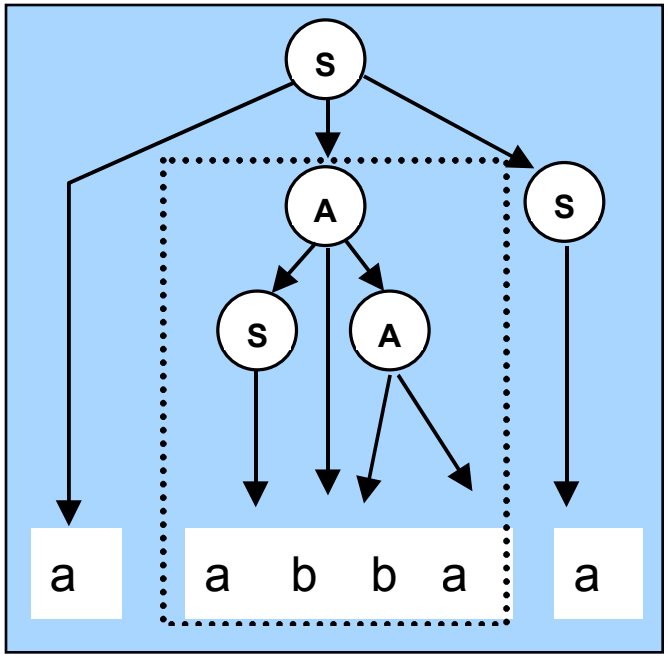
Formal Grammars. Parse subtrees

- Given a tree A corresponding to a derivation, a **subtree of A** is the tree whose root is an any node of A, whose nodes are all descendents of the root of the subtree of A , and whose branches link all these nodes.



Formal Grammars. Parse trees

- **Theorem:** the leaves nodes of a subtree, read from left to right, form a sentence with regard the nonterminal symbol root of the subtree.



abba is a sentence of the sentential form *aabbaa* with regard the symbol A

Formal Grammars. Ambiguity

Concepts related to the derivation tree:

- If a sentence can be obtained from a G by means of two or more parse trees, the sentence is ambiguous.
- A G is ambiguous if it contains at least one ambiguous sentence.
- Even if a G is ambiguous, the language that it describes might be not ambiguous [Floyd 1962] $\Rightarrow$ it is possible to find an equivalent grammar G that it is not ambiguous.

Formal Grammars. Ambiguity

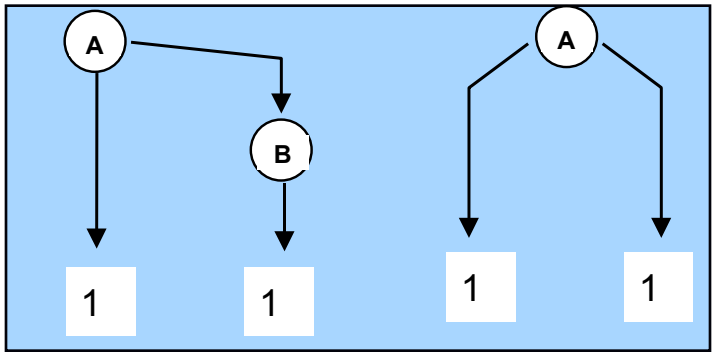
Concepts related to the derivation tree:

- There are languages for which it is not possible to find no ambiguous grammars ⇒ Inherent ambiguous languages [Gross 1964]
- The property of ambiguity is undecidible. It is only possible to find sufficient conditions to ensure that a G is not ambiguous.
- Undecidible: there is not an algorithm that accepts a G and ensures in a finite time whether a G is ambiguous or not.

Formal Grammars. Ambiguity

- From the previous definitions, there are 3 levels of ambiguity:
  - **Sentence:** a sentence is ambiguous if it can be obtained by means of two or more different derivation trees.

e.g.: G = ({1}, {A,B}, A, {A ::= 1B | 11, B ::= 1})



  - **Grammar**: it is ambiguous if it contains at least an ambiguous sentence, e.g.: the previous G.

  - **Language:** a language L is ambiguous if there is an ambiguous grammar that generates it, e.g.: L = {11} is ambiguous.

Formal Grammars. Ambiguity

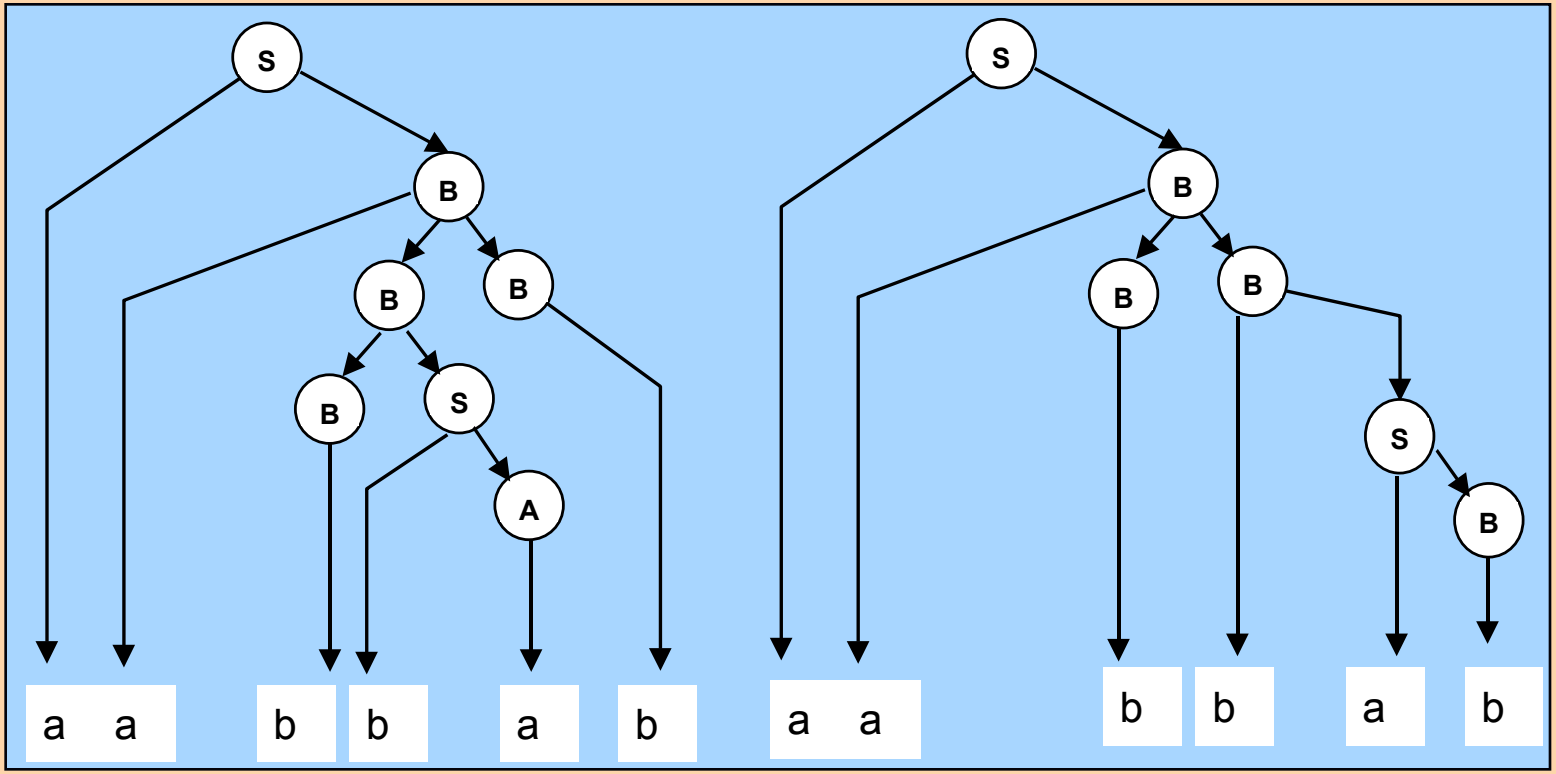
- **Inherent Ambiguous Languages:** languages for which it is not possible to find a not ambiguous G $\Rightarrow$ example L = $\{\{a^n b^m c^m d^n\} \cup \{a^n b^n c^m d^m\} / m,n \geq 1\}$

  example: L = {11} is not inherent ambiguous.

  G' = ({1}, {A}, A, {A ::= 11})

Formal Grammars. Ambiguity

- Given the grammar G = ({a,b}, {A,B,S}, S, P)
  P = { S::= bA | aB, A::= bAA | a | aS, B ::= b | BS | aBB} Show that it is an ambiguous grammar given that the sentence “aabbab” is ambiguous.



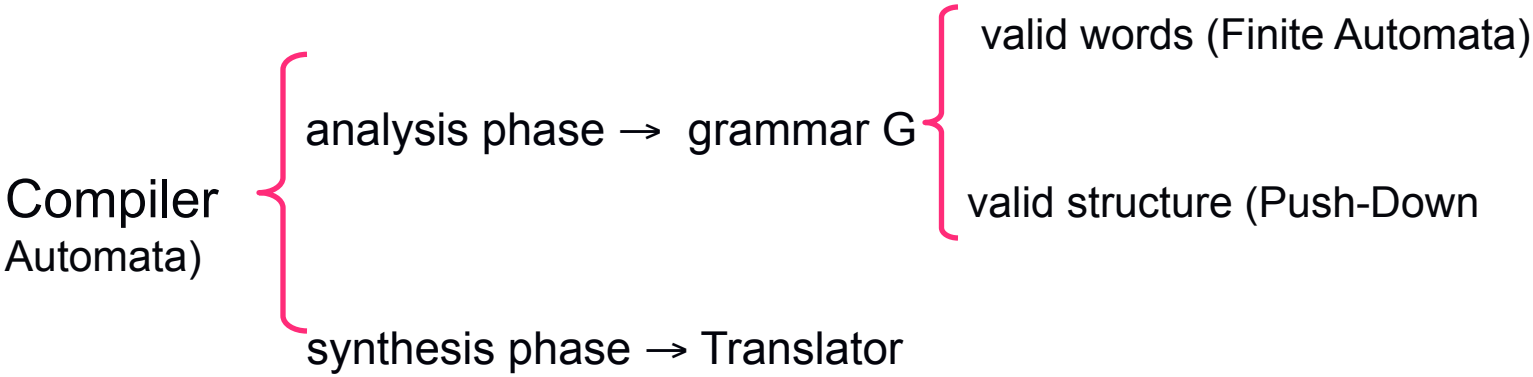
OUTLINE

- Definition of a grammar (notation)
  - Sentential form. Sentence
  - Equivalent grammars
  - Sentences and handles
  - Recursion
- Chomsky Hierarchy
  - Regular grammars (Type 3, G3) and equivalences
- Parse trees
  - Ambiguity
- **Context-free grammars languages(Type 2, G2)**
  - **Language generated by a Type-2 Grammar**
  - **Well-formed grammars**
  - **Chomsky. Normal Form. Greibach Normal Form**

Formal Grammars. Context-Independent Grammars

- Type 2 grammars in Chomsky Hierarchy.
- Importance:

  They are used in the definition of programming languages and the compilation processes.



Formal Grammars. Context-Independent Grammars

- Languages generated by Chomsky Hierarchy type 2 grammars are called:
  - Context-Independent or Context-Free languages.
  - Represented by L(G2).
  - There are algorithms that recognize if a L (G2) is empty, finite or infinite.

Formal Grammars. Context-Independent Grammars

- Language, empty or not?:

  Let G2, m = $C(\Sigma_{NT})$, **L(G2)** $\neq \phi$ if $\exists$ x $\in$ L(G2) that can be generated with a derivation tree in which all the paths have a length $\leq$ m

  All the derivation trees with paths $\leq$ m = $C(\Sigma_{NT})$ by means of the algorithm:

  a) Set of trees with length 0 (a tree having S as a root and without branches).

  b) From the set of trees of length m, we generate the set with length m +1 by using the initial set and applying a production that does not duplicate a nonterminal in the considered path.

  c) The step b) is recursively applied until any tree with length $\leq$ m can not be obtained. Given that m and the number of productions P is finite $\Rightarrow$ the algorithm ends.

  L(G2) = ϕ if none of the trees generates a sentence

Formal Grammars. Context-Independent Grammars

- Example of L(G2) = ϕ

  G = ({a,b}, {A,B,C,S}, S, P)

  P = {

  S ::= aB | aA

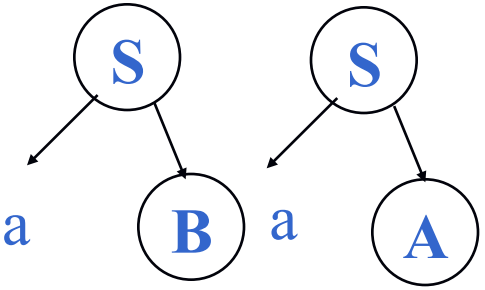
  A ::= B | abB

  B ::= bC}

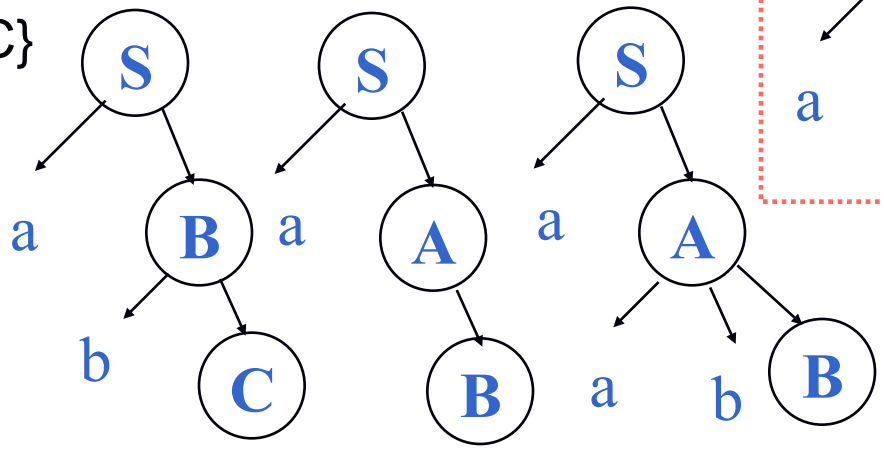
$m=C(\Sigma_{NT}) = 4$

1. m=0

2. m=1



3. m=2



4. m=3

Sentences are not generated and there are nonterminals already obtained

Empty language

Formal Grammars. Context-Independent Grammars

- If L(G2) is nonempty, evaluate if **L(G2) = ∝**

A graph is built whose nodes are labeled with the symbols of (Σ_{NT}) by means of the algorithm:

a) if ∃ a production A::= α B β, an arc from A to B is created, where A,B $\in \Sigma_{NT}$ and $\alpha,\beta \in \Sigma^*$
b) If there are not cycles in the graph, then L(G2) = finite
c) L(G2) = ∝ if there are cycles that are accessible from the axiom and correspond to derivations A $\rightarrow+$ α A β, where /α/ + /β/ >0 (they can not be λ simultaneously).

L(G2) ≠ ∝ if there are not cycles in the graph.

Formal Grammars. Context-Independent Grammars

- Example of **L(G2) =** $\propto$

  G = ({a,b,c}, {A,B,C,S}, S, P)

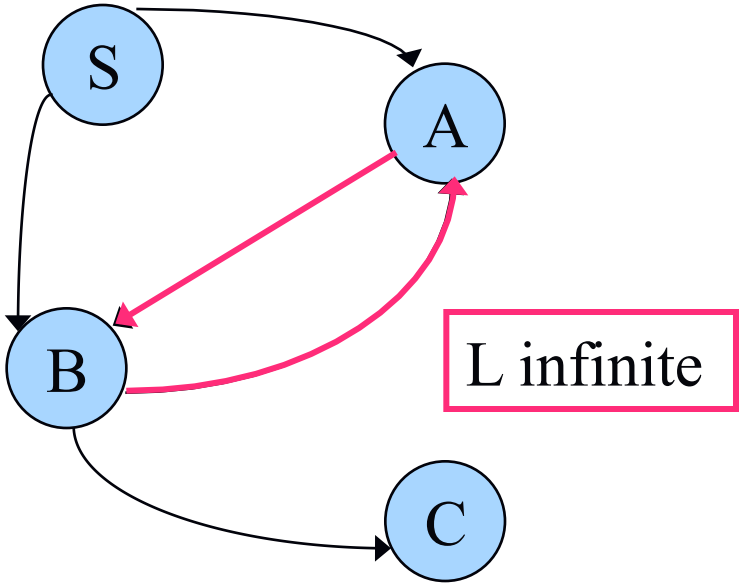
  P = {

  S ::= aB | aA

  A ::= abB

  B ::= bC | aA

  C ::= c }



Context-Independent Grammars. Well-formed Grammars

- Transformation of a given G into another whose production rules are in a well-formed format without imperfections:

1. **Possible imperfections (make the grammar not-clean):**
    1.1. Unnecessary Rules.
    1.2. Unreachable Symbols.
    1.3. Useless Rules.
2. **Not Generating symbols.**
3. **Not Generating rules.**
4. **Redenomination rules (Unit Productions).**

***After eliminating 1, 2, 3, 4 the grammar is* clean *and* well-formed**

Context-Independent Grammars. Well-formed Grammars

1. **Possible imperfections**

1.1 Unnecessary Rules: rules **A::= A** $\in$**P** are unnecessary and make the grammar to become ambiguous $\Rightarrow$ eliminate.

1.2 Unreachable Symbols:

If $U \in \Sigma_N \neq S$ is unreachable then $S \not\rightarrow^* xUy$.

This means that if U does not appear in the right side of any production rule, then U is unreachable (the reciprocal is not true)

Elimination of unreachable symbols:

a) boolean matrix (Alfonseca, Chapter 3).

b) Graph that is identical to L(G2) = $\propto$. Unreachable symbols are those not reached from the axiom.

Eliminate those symbols and the rules that contain them.

Context-Independent Grammars. Well-formed Grammars

1. **Possible imperfections**

1.2 Unreachable symbols: example:

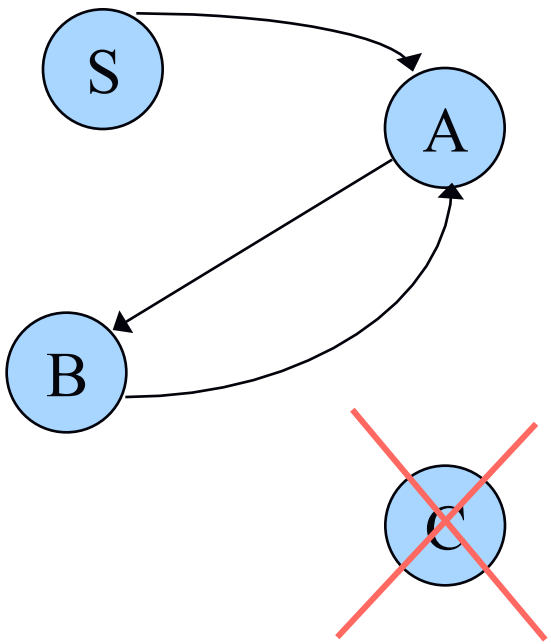
G = ({a,b,c}, {S,A,B,C}, S, P),

where P = { S ::= aA

A ::= Bc

B ::= bA

C ::= c }



Context-Independent Grammars. Well-formed Grammars

1. **Possible imperfections**

1.3 Useless Rules: are those that do not contribute to the words formation.

If U is a useless symbol then $U \rightarrow^* t$, where $t \in \Sigma_T^*$

Algorithm:

a) Annotate nonterminal symbols for which there is a rule $U ::= x$ where $x \in \Sigma^*$. (it is a string of terminals or λ, or in successive derivations it contains annotated nonterminal symbols)

b) If every nonterminal is annotated $\Rightarrow$ there are not useless symbols (END).

c) If a nonterminal symbol was annotated last time in a), then return to a).

d) Every $U \in \Sigma_N$ not annotated is useless.

Context-Independent Grammars. Well-formed Grammars

1. **Possible imperfections**

1.3 Useless Rules

example: G= ({e,f}, {S,A,B,C,D}, S, P)

where P= {

S::= Be

A::= Ae | e

B::= Ce | Af

C::= Cf

D::= f }

First iteration:

D::= f y A::= e

Second iteration:

B::= Af

Third iteration:

S::= Be

Nonterminals not annotated: C → this symbol and the rules that contain it can be eliminated.

Context-Independent Grammars. Well-formed Grammars

2. **Elimination of Not Generating symbols:**

   Given G2 = (Σ_T, Σ_N, S, P), **$\forall A \in \Sigma_N$** we will build the grammar G(A), where A is the axiom. If **L(G(A))= ϕ $\Rightarrow$ A is a Not Generating symbol** and can be eliminated, as well as all the rules containing it, obtaining another equivalent G2.

Context-Independent Grammars. Well-formed Grammars

3. **Elimination of Not Generating rules:** they are A ::= λ (A≠S)

   - if λ ∈**L(G)**: then we add S::= λ **and** ∀ A∈Σ_N (A ::= λ A≠S) and ∀ rule de G with the structure B ::= xAy, we add a P' rule with the structure B ::= xy, except the case x=y=λ

   Example:

   A::=C0B, B::=BC, C::=0B, B::= λ

   A::=C0B, A::=C0, B::=BC, B::=C, C::=0B, C::=0

Context-Independent Grammars. Well-formed Grammars

4. **Elimination of Redenomination rules:** they are rules type A ::= B

Algorithm:

a) The equivalent G' contains all the rules except:

A ::= B

G' is obtained with a new P'

$$\forall A \in \Sigma_N \mid A ::= B \text{ in } G \text{ and } \forall (B ::= x) \in P$$

$$P' = P + \{A ::= x\}$$

Context-Independent Grammars. Well-formed Grammars

- **Exercise (Alfonseca pg. 210)** Given the grammar G = ({0,1}, {S,A,B,C},S, P), where P =

  {S::= AB | 0S1 | A | C

  A ::= 0AB | λ

  B::= B1 | λ}

  Calculate the equivalent well-formed grammar.
- **Exercise (Isasi, Martínez, Borrajo 4.7)** Given the grammar G = ({0,1}, {S,A,B,C,D,E,F},S, P), where P =

  {S::= AB | 0E | A | CS1

  A ::= 0AS | λ | A0 | C

  B::= B1 | 1

  D::= B1 | λ | 1F

  E ::= E1

  F::=0D }

  Calculate the equivalent well-formed grammar.

Context-Independent Grammars. Well-formed Grammars

- **G = ({0,1}, {S,A,B,C},S, P),**
  where P = { S::= AB | 0S1 | A | C, A ::= 0AB | λ, B::= B1 | λ }
  - There are not unnecessary rules.
  - There are not unreachable symbols.
  - There are not useless rules and not generating symbols.
  - Not generating rules.
  - Redenomination rules.

Context-Independent Grammars. Well-formed Grammars

- **G = ({0,1}, {S,A,B,C},S, P),**
  where P = { S::= AB | 0S1 | A | λ, A ::= 0AB | λ, B::= B1 | λ }
  - There are not unnecessary rules.
  - There are not unreachable symbols.
  - There are not useless rules and not generating symbols.
  - Not generating rules: the result their elimination is:

  P' = {S::= A | B | AB | 0S1 | λ, A::= 0AB | 0A | 0B |0, B::=B1 | 1**}**
  - Redenomination rules.

There are 2:

S::= A | B, are replaced for the corresponding right sides.

S::= 0AB| 0A | 0B | 0 | B1| 1 | 0S1| λ |AB

Context-Independent Grammars. Well-formed Grammars

- G = ({0,1}, {S,A,B,C},S, P),
  where P = { S::= AB | 0S1 | A | C, A ::= 0AB | λ, B::= B1 | λ }

- G = ({0,1}, {S,A,B},S, P),
  where P = {S::= 0AB | 0A | 0B | 0 | B1 | 1 | 0S1| AB | λ,
  A::= 0AB | 0A | 0B | 0, B::=B1| 1}

It is the equivalent well-formed grammar.

Context-Independent Grammars. Normal Forms

- There are possible notations for G2 grammars:
  - They define the valid structures for the production rules.
- We are going to study:
  - Chomsky Normal Form (CNF).
  - Greibach Normal Form (GNF).

Context-Independent Grammars. Normal Forms (CNF)

- Chomsky Normal Form (CNF).
  - Every nonempty context-free grammar without λ has an equivalent grammar *G* in which all productions are in one of the following two simple forms:
    - A ::= BC
    - A ::= a

Context-Independent Grammars. Normal Forms (CNF)

- Chomsky Normal Form (CNF).
  - Algorithm:

    a) Start with a well formed grammar.

    b) Separate rules *A* ::= *a* (they are in CNF)

    c) Break bodies of length greater than 1 into a set of productions, each with a body consisting of nonterminals.

    Example:

    A::= aBCD →A::=EBCD→E::=a

Context-Independent Grammars. Normal Forms (CNF)

- Chomsky Normal Form (CNF).
  - Algorithm:

    Step c):

    - For every terminal *a* that appears in a body of length 2 or more, create a new nonterminal *A* with only one production (*A*::=*a*).
    - Use *A* instead of a everywhere *a* appears in a body of length 2 or more.

Context-Independent Grammars. Normal Forms (CNF)

- Chomsky Normal Form (CNF).
  - Algorithm:

    Step c):

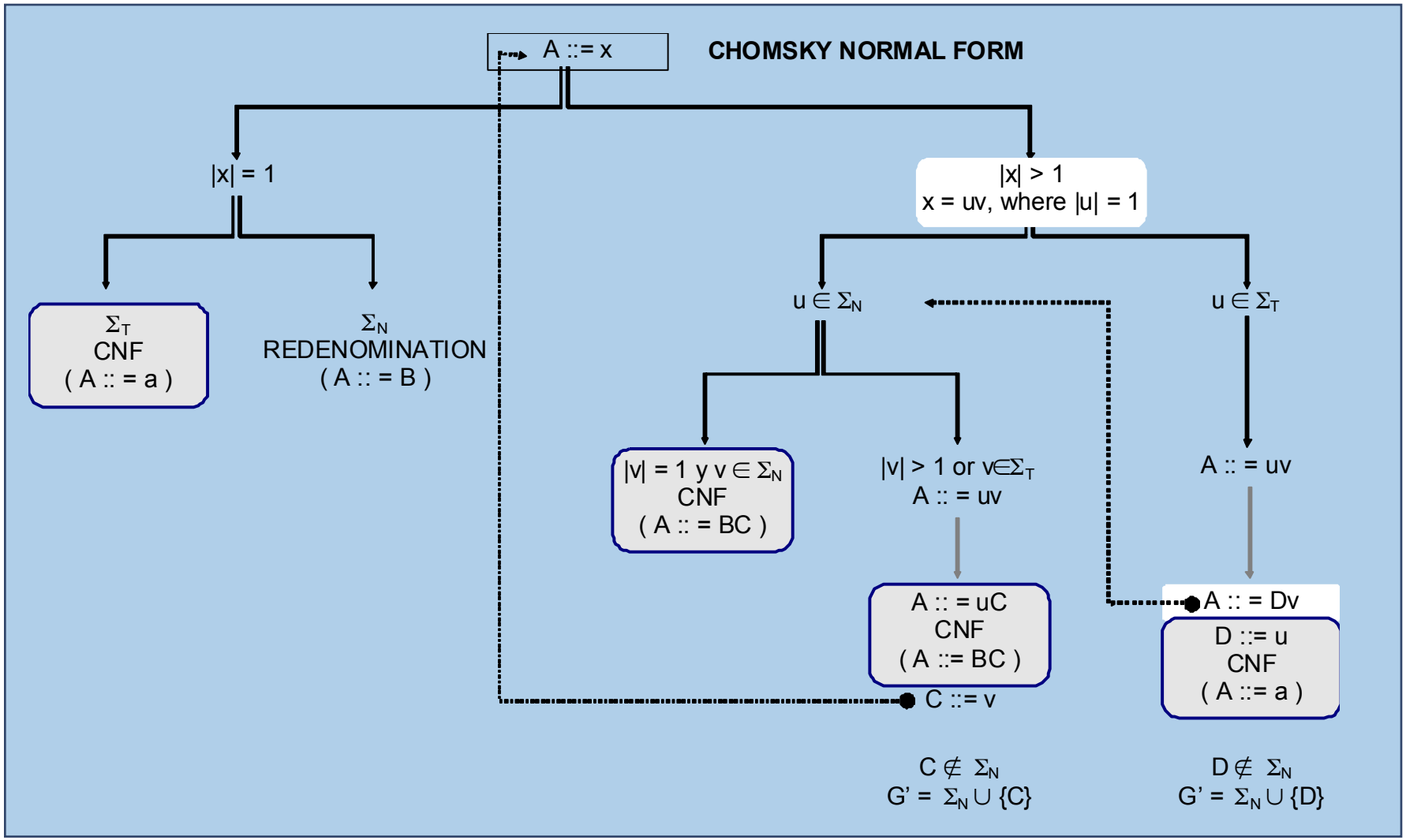
    - Break the productions

      $$A ::= B_1B_2\cdots B_k,\ k> 2$$

    into a group of k-1 productions with two nonterminal in each body by introducing k-2 nonterminal $C_1, C_2, \cdots, C_{k-2}$

    $$A ::= B_1C_1,\ C_1::=B_2C_2,\cdots,C_{k-3} ::= B_{k-2}C_{k-2},\ C_{k-2} ::= B_{k-1}B_k$$

Context-Independent Grammars. Normal Forms (CNF)



Context-Independent Grammars. Normal Forms (CNF)

- **Exercise (Alfonseca pg 212)** Given the grammar G = ({0,1},

  {S,A,B},S, P), where

  P = { S::= AB | 0S1 | 0AB | 0B | 0A | 0 | B1 | 1 | λ

  A ::= 0AB | 0B | 0A | 0

  B::= B1 | 1 }

  Calculate the equivalent CNF grammar.

Context-Independent Grammars. Normal Forms (GNF)

- **GNF** is a very interesting notation for several techniques for syntax recognition. Every rule has in the right side an initial terminal symbol optionally followed by one or more nonterminal symbols.
- THEOREM: Every **context-free L without λ** can be generated by a G2 in which all the rules have the structure:
  - **A ::= aα** where $A \in \Sigma_{NT}$ $a \in \Sigma_T$ $\alpha \in \Sigma_{NT}^*$
  - If $\lambda \in L$, then it is necessary to define **S::= λ**
- THEOREM: Every G2 can be reduced to another equivalent G2 without left-recursive rules.

Context-Independent Grammars. Normal Forms (GNF)

- **GNF:** to tranform a G2 grammar into its equivalent Greibach normal form:

  1. Eliminate left recursion.
  2. Use the algorithm to transformate to GNF, verifying in every step that left recursive rules are not created, using step 1 to eliminate them if they are created.

Context-Independent Grammars. Normal Forms (GNF)

1. Eliminate left recursion: only one step (several steps are described in Isasi, Martínez, Borrajo, pg 24):

G = ({α,β}, {A}, A, P), where P = {A::= A α | β}

$A \rightarrow \beta$

$A \rightarrow A\,\alpha \rightarrow \beta\,\alpha$

$A \rightarrow A\,\alpha \rightarrow A\,\alpha\,\alpha \rightarrow \beta\,\alpha\,\alpha$

$A \rightarrow A\,\alpha \rightarrow A\,\alpha\,\alpha \rightarrow \ldots \rightarrow \beta\,\ldots\ldots\,\alpha\,\alpha$

A ::= β

A::= βX

X::= α X

X::= α

Context-Independent Grammars. Normal Forms (GNF)

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

Context-Independent Grammars. Normal Forms (GNF)

- 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. $P' = P \cup \{ A ::= \beta_1 \cdot A' | \beta_2 \cdot A' | \ldots | \beta_m \cdot A',$

$A' ::= \alpha_1 \cdot A' | \alpha_2 \cdot A' | \alpha_n \cdot A' | \lambda \}$

Context-Independent Grammars. Normal Forms (GNF)

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

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

Context-Independent Grammars. Normal Forms (GNF)

- 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

Context-Independent Grammars. Normal Forms (GNF)

1. Eliminate left recursion: Summary

Given a grammar G = ({α_1, α_2, β_1, β_2}, {A}, A, P),

Where P = {A::= A α_1/ Aα_2 / β_1/ β_2}

The final result is:

A::= β_1 | β_2 | β_1X | β_2X

X::= α_1 | α_2 | α_1 X | α_2X

Context-Independent Grammars. Normal Forms (GNF)

2. Transformation of a well-formed G2 without left-recursion to GNF:

2.1 Define a partial relation in Σ_{NT}

$\Sigma_{NT} = \{A_1, A_2, ..., A_n\}$ based in: if $A_i \rightarrow A_j \alpha$, then A_i precedes A_j.

If there are "contradictory" rules, use on of them for the order and look at the rest to select the most convenient rule.

2.2 Classify the rules in three possible groups:

Group 1: $A_i \rightarrow a \alpha$,where a $\in \Sigma_T$ and $\alpha \in \Sigma^*$

Group 2: $A_i \rightarrow A_j \alpha$ where Ai precedes A_j in the set of orderly Σ_{NT}

Group 3: $A_k \rightarrow A_i \alpha$ where Ai precedes A_k in the set of orderly Σ_{NT}

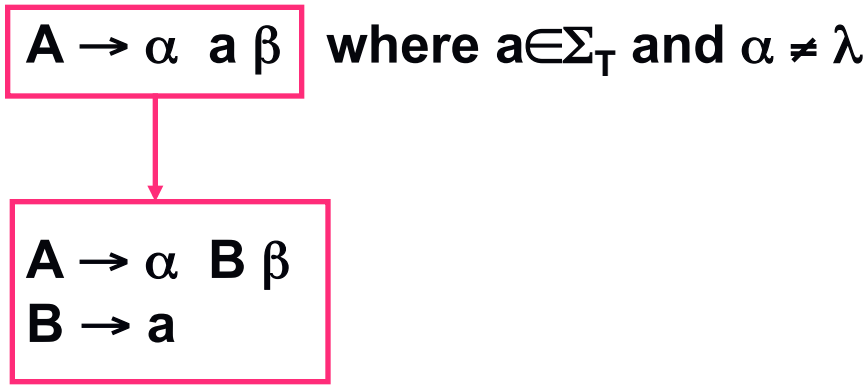
2.3 The rules are transformed group 3 → group 2 → group 1: FNG

$A_k \rightarrow A_i \alpha$ substitute **A_i** for the right side of every rule in every group that contains **Ai** as left side.

Repeat the same process with group 2 rules.

Context-Independent Grammars. Normal Forms (GNF)

2. Transformation of a well-formed G2 without left-recursion to GNF:
   2. 4. If all the rules are Group 1, the G is in GNF **and we have only to delete terminal symbols that do not appear on the right-hand header.**



Context-Independent Grammars. Normal Forms (GNF)

- **Exercise (Alfonseca pg 217)** Given the grammar G= ({a,b}, {A,B,C}, A, P), where P = {A ::= BC, B ::= CA | a, C ::= AB | b}

  Calculate the equivalent GNF.

- **Exercise (exam September 1994)** Given the grammar G= ({a,b}, {S}, S, P), where P = {S::= aSb | SS | λ}

  Calculate the equivalent GNF.

- **Exercises (Isasi, Martínez, Borrajo; 4.21- 4.25)**

Context-Independent Grammars. Left factoring

- It resolves a problem during the compilation process: Eliminate productions that have common first symbol(s) on the right side of the productions.

- Example: G = ({If, Then, Else, Repeat, Until, Forever}, {S,C}, S, P), some rules in P are:

  S ::= If *C* Then *S* Else *S* | If *C* Then *S* | Repeat *S* Until *C* |

  Repeat *S* Forever

Context-Independent Grammars. Left factoring

- For productions of the form:
  - *P*: $A ::= \beta \cdot \alpha_1 \mid \beta \cdot \alpha_2$ (for every $A \in \Sigma_{NT}$)
- Algorithm:
  1. $\Sigma_N = \Sigma_N \cup \{A'\}$
  2. $P' = P \cup \{A ::= \beta \cdot A',$
     $A' ::= \alpha_1 \mid \alpha_2\}$

Note: α and β can be strings of terminals and nonterminals

Context-Independent Grammars. Left factoring

- Example:
  - P : $S \rightarrow$ ***if*** C ***then*** S ***else*** S

    $S \rightarrow$ ***if*** C ***then*** S

    $S \rightarrow$ ***repeat*** S ***until*** C

    $S \rightarrow$ ***repeat*** S ***forever***
- Solution:
  - P': $S \rightarrow$ ***if*** C ***then*** $S\ A'$

    $S \rightarrow$ ***repeat*** $S\ A''$

    $A' \rightarrow$ ***else*** $S \mid \lambda$

    $A'' \rightarrow$ ***until*** $C \mid$ ***forever***

Context-Independent Grammars. Left factoring

- 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

Context-Independent Grammars. Left factoring

- A grammar may not appear to have left recursion or left factors:

  - S ::= Tu | wx
    T ::= Sq | vvS

  - S ::= Tu | wx
    T ::= Tuq | wxq | vvS

  - S ::= Tu | wx
    T ::= wxqT' | vvST'
    T' ::= uqT' | λ