GRAMMAR

- 一般之語言，以數學形式定義如下：
- 定義一：L(T, N, P, S)為一語言。

  T：所有終端符號(Terminal symbol)之集合。

  N：所有非終端符號(Non-terminal symbol)之集合。

  P：所有造句程序(Production)之集合。

  S：所有起始符號(Start symbol)之集合。

- 定義二：

  T*：所有屬於T之終端符號所組成字串之集合。

  ξ：由起始符號S一造句程序P所產生的字串，屬於T*。

- 定義三：

  若且唯若存在一組λ_1，λ_2，...，λ_{n-1}對任一λ_i　　可以由λ_{i-1}直接產生，則λ_n可由λ_0產生。

- 定義四：

  (a) $\xi = \alpha\xi'\beta$

  (b) $\eta = \alpha\eta'\beta$

  (c) 存在一組造句程序p，$\xi' ::= \eta'$

  若且唯若存在符號串合乎上述三條件時，則可由直接產生，即$\xi \rightarrow \eta$。

- In general, a grammar involves four quantities: *terminals, nonterminals, a start symbol, and productions.*

- The basic symbols of which strings in the language are composed we shall call *terminals*. The word “token” is a synonym for “terminal” when we are talking about programming languages. In the example above, certain keywords, such as **begin** and **else** are terminals, as are punctuation symbols such as ‘;’ and operators such as ‘**+**’.

- Nonterminals are special symbols that denote sets of strings. The terms “syntactic variable” and “syntactic category” are synonyms for “nonterminal.” In the examples above, the syntactic categories statement, expression, and statement-list are nonterminals; each denotes a set of strings. One nonterminal is selected as the start symbol, and it denotes the language in which we are truly interested. The other nonterminals are used to define other sets of strings, and these help define the language. They also help provide a hierarchical structure for the language at hand.

- The productions (rewriting rules) define the ways in which the syntactic categories may be built up from one another and from the terminals. Each production consists of a nonterminal, followed by a string of nonterminals and terminals.

statement → **begin** statement-list **end**

statement-list → statement

statement-list → statement ; statement-list

- (例一)英語簡單語句由下列三條造句程序組成。

  P1<sentence> ::= <subject><predicate>.

  P2<subject> ::= lions | cats

  P3<predicate> ::= cry | fly

- P1描述一個語句<sentence>定義為非終端符號主詞<subject>之後，緊跟著另一非終端符號述詞<predicate>，終端符號．句點. P2說明非終端符號主詞<subject>定義為終端符號(terminal symbol) lions或終端符號cats。P3說明非終端符號<predicate>定義為終端符號cry或fly。

- 因此合乎簡單語句<sentence>語法的與具有下列四句是正確，其他為語法錯誤。

語句一	lions cry.
語句二	lions fly.
語句三	cats cry.
語句四	cats fly.

- 以例一說明。T = {lions, cats, cry, fly, . }。N = {<sentence>, <subject>, <predicate>}。P = { P_1 , P_2 , P_3 }。S = {<sentence> }。L = 英語簡單語句。由定義四及P_1可知 為<sentence>， 為 η' <subject><predicate>，即<subject><predicate>可由<sentence>直接產生，寫成

  <sentence> → <subject><predicate>.

- 同理

  <subject> → lions

  <predicate> → cry

  故

  <sentence> $\overset{*}{\rightarrow}$ lions cry.

- 因<sentence> $\overset{*}{\rightarrow}$ lions cry . 依據定義二知其為英語簡單語句中之一個語句，也就是說lions cry . 語法正確。由例一可以看出非終端符號<subject>和<predicate>僅能存在於轉換過程，當轉換完成時所有的符號均為終端符號。依語法規則造出語句，所以稱這些語法規則為造句程序(production)，習慣上以p表示。

語言表示法

- 每一種語言(language)均有一套語法(syntax)。合乎語法為正確。描述語法最常用的為EBNF法(Extended Backus Naur Form)。BNF因描述程式語言ALGOL 60而著名，後來擴展到其他語言的描述，例如英文語句(sentence)可以表示為：

  <sentence> ::= <subject><predicate>

- 像BNF用來描述另一種語言，稱為類語言(metalanguage)，其使用的符號為類符號(metasymbol)。BNF的類符號有三種：

  (1) ::= 定義為。
  (2) | 或。
  (3) <> 非終端符號(non-terminal symbol)。

- 後來為了描述方便起見，增加數種：

  (4) {} 重複發生0次，1次，2次，...，多次
  (5) [] 選擇。方括號內之結構可以選用，也可以不選用。
  (6) \\ 在背斜線(back slash)中之結構必須選一種。

以上六種類符號構成了EBNF類語言。

語句分析

- 辨認語句是依輸入的語句，導出產生該語句的造句程序。一般稱此工作為剖析(parsing)，其工作通常複雜而困難，有時甚至不可能做到。而用那一類型的造句程序來定義語言是影響辨認工作是否困難的主要原因。常用的為由上而下(top-down)的剖析方法，其法由起始符號找出產生整個語句的一系列造句程序。例如要剖析lions cry . 是否合乎英語簡單語句，從起始符號<subject>開始，由P_2得知<subject>可以直接產生lions，由P_3的起始符號<predicate>直接產生cry，故知lions cry . 為英語簡單語句。

以剖析樹(parsing tree)表示如下：

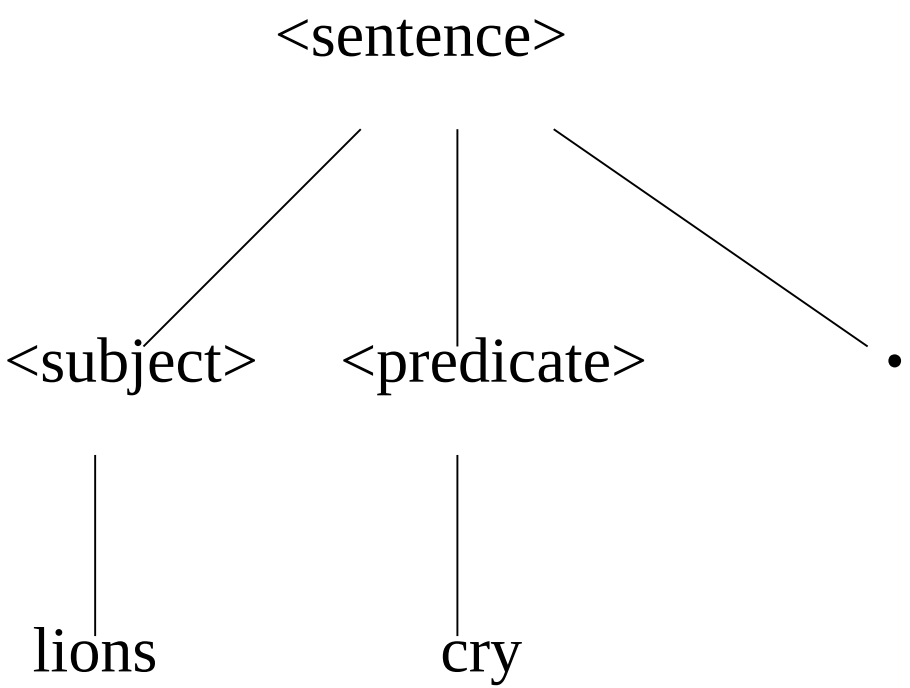


圖2.1 剖析樹。說明lions cry . 之剖析情形

- （例二）S語言以EBNF定義如下，以小寫字母表示者為終端符號。

  P_1 <S> ::= x <A>

  P_2 <A> ::= z | y <A>

  請以由上而下的剖析方法剖析xyz是否合乎S語言。

（解）

<S>	xyz	
x <A>	xyz	依造句程序P_1
<A>	yz	
y <A>	yz	依造句程序P_2
<A>	z	
z		依造句程序P_2

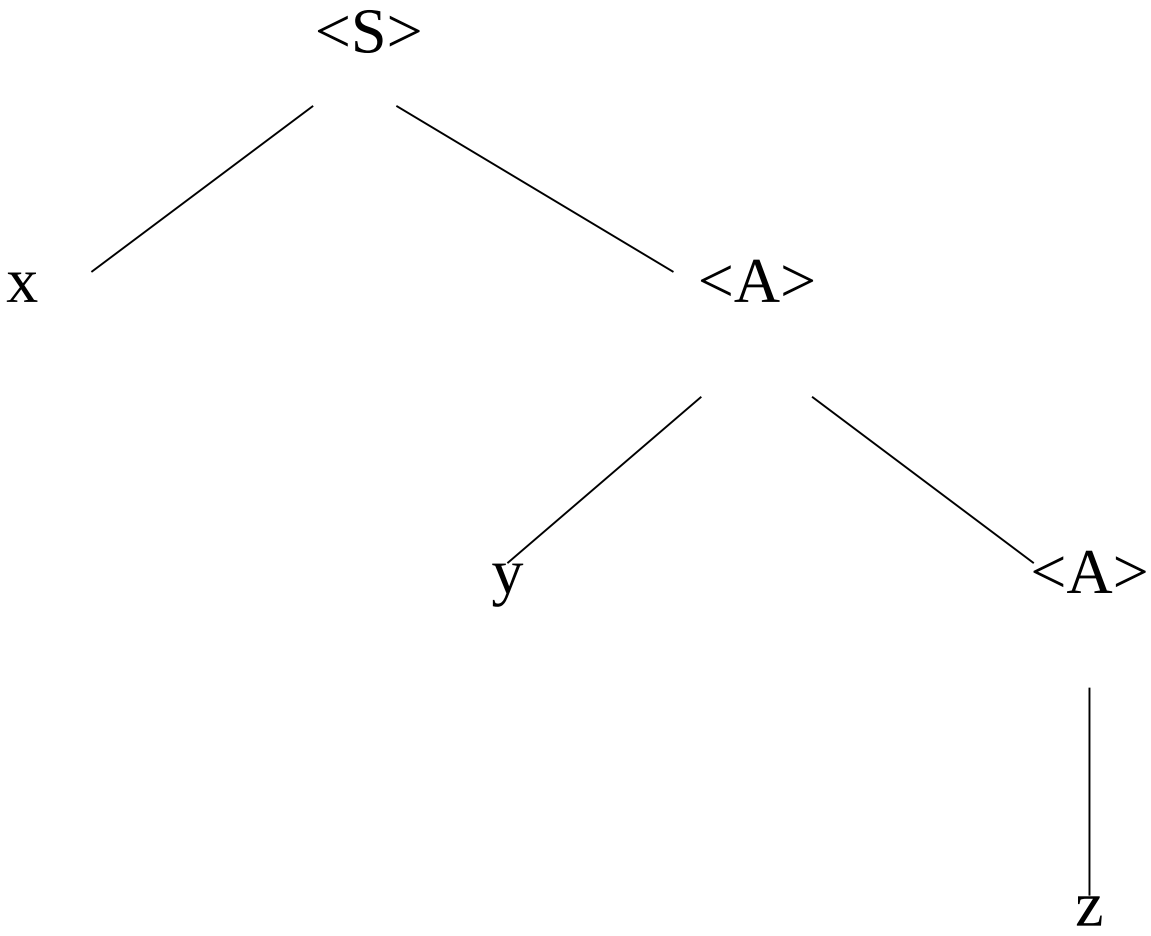


圖2-2 由上而下剖析xyz是否合乎S語言

- (例三) 有一T語言，以EBNF表示如下。

  P_1 <T> ::= <A> | <B>

  P_2 <A> ::= x <A> | y

  P_3 <B> ::= x <B> | z

  請以由上而下方法剖析xxz使否合乎T語言。

（解）

<T>	xxz	<T>	xxz
<A>	xxz	<B>	xxz
x<A>	xxz	x<B>	xxz
<A>	xz	<B>	xz
x<A>	xz	x<B>	xz
<A>	z	<B>	z
?		z	

- 在剖析過程中，<T>之轉換，若選<A>，發現xxz不合乎T語言，若選<B>，發現xxz合乎T語言。問題在造句程序P_2及P_3之起始符號均為x，以致無法判斷鑰匙用那一造句程序來剖析，這種情形應該避免。

- 規定甲：

  若有一造句程序

  <A> ::= $\xi_1 | \xi_2 | \ldots | \xi_n$

  則所有能從產生語句的起始符號之交集必須是空集合。

  即 $start(\xi_i) \cap start(\xi_j) = \phi$

  但i, j = 1,2,…,n

  且i ≠ j

- 集合$start(\xi)$ 是由所有能從產生語句的起始符號所組成，可由下列兩個規則得到。

- 規則一：若起始符號就是終端符號，則為該終端符號之集合。即

$$start(x\xi) = \{x\}$$

- 規則二：若起始符號為非終端符號，其造句程序為

<A> ::= $\alpha_1 \mid \alpha_2 \mid ... \mid \alpha_n$

則

$$start(< A >) = start(\alpha_1) \cup start(\alpha_2) \cup ... \cup start(\alpha_n)$$

- 在例三中，終端符號x同時屬於start(<A>)和start(<B>)，因此其交集不是空集合，違反規定甲。例三的語法若改為

  <S> ::= <C> | x <S>

  <C> ::= y | z

  則符合規定甲。但還有一些問題必須克服，請看例四。

- （例四）有一語言U以EBNF表示

<U> ::= <A> x

<A> ::= x |ε

請剖析x是否為U語言之語句。

（解）

<U>	x	<U>	x
<A> x	x	<A> x	x
x x	x	ε x	x
x	?	x	x

- 剖析得兩種不同的結果，問題在於<A>選x或ε。稱為空符號串問題，這種現象僅有在非終端符號能產生空符號串時才發生，為了避免這種問題的發生，必須另有規定。

- 規定乙：

對於任一非終端符號，且具有產生空符號串之能力時，則所有<A>之起始符號集合，與所有<A>之跟隨符號集合之交集必須為空集合。即

$$start(< A >) \cap follow(< A >) = \phi$$

關於集合follow(<A>)可依下述方法求得。對於任一造句程序

$$< B >::= \xi < A > \eta$$

若集合S_i = start(η_i)，則集合follow(<A>)為所有S_i之聯集，即

$$follow(< A >) = start(\eta_1) \cup start(\eta_2) \cup ... \cup start(\eta_n)$$

- 若存在一η_i會產生空符號串時，則集合follow(<B>)必包含於集合follow(<A>)之內。即

$$follow(<B>) \subset follow(<A>)$$

- 在例四之中，<A>違反規定乙，因為

$$start(<A>) = follow(<A>) = \{x\}$$

- 若一語言有表示重複符號串之需求時，通常使用遞迴(recursive)結構表示。例如

  <A> ::= <B> | <A><B>

  此造句程序將產生如下之符號串

  <B>, <B><B>, <B><B><B>, …

  但此造句程序違反規定甲，因為

$$start(<B>) \cap start(<A><B>)$$
$$= start(<B>) \neq \phi$$

若修改為

$$<A>::=\varepsilon|<A><B>$$

將產生如下之符號串

ε, <B>, <B><B>, <B><B><B>, …

卻又違反規定乙，因為

$$start(<A>)\cap follow(<A>)$$
$$=\{\varepsilon, start(<B>)\}\cap start(<B>)$$
$$=start(<B>)\neq\phi$$

- 因此可知由於規定甲和規定乙的限制，無法使用左遞迴結構來描述語言。有二種方法可以克服困難，其一為改用右遞迴結構，如

$$<A>::=\varepsilon|<B><A>$$

- 其二為使用EBNF中的{<B>}表示重複結構。

- 除了語法外，語意semantic也是很重要的，如例一中的lions fly .語法正確，但語意卻有問題。

【例二】多重選擇題

The following grammar N = { B ,Σ} T = { a , b }

P = {Σ→a B

B →a B B

B →b }

Generates strings (a)abab (b)aabb (c)aaabbb (d)aababb.

【解】(b)(c)(d)

【說明】(b)(c)(d)的導出過程如下：

Σ→a B→a a B B→a a b b

Σ→a B→a a B B→a a a B B B→aaab b b

Σ→a B→a a B B→a a b a B B→a a b a b b

- 【例三】Suppose that a grammar has the following productions:

  S→a B c

  B→b X b

  B→b X

  X→a

  X→a b

Is the string “a b a b c” in the language generated by the grammar?

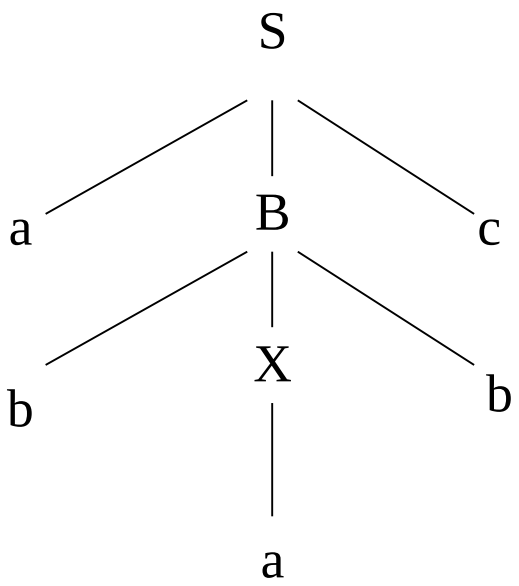
Is the grammar ambiguous? Justify your answer.

Describe the set of all strings in the language which is generated by the grammar.

【解】

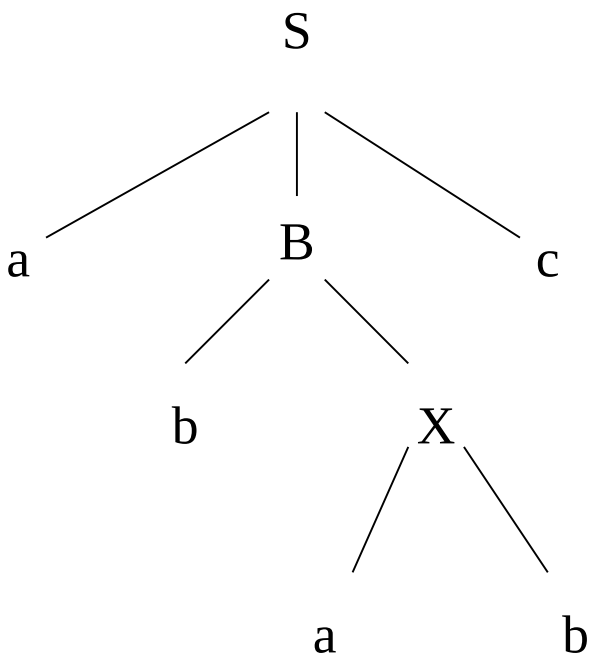
(一)Yes, the string “a b a b c” is in the language generated by the grammar.

【說明】字串 “a b a b c”能從上述文法導出，以剖析樹(parse tree)敘述導出情形如下：



（二）Yes, the grammar is ambiguous.

本例之文法對於某些字串的展開有一種以上的導出方式，所以此文法是語意不明確(ambiguous)之文法；例如：本例（一）中字串 “a b a b c”有另一種導出方式，以剖析樹敘述導出情形如下：



（三）the set of all strings in the language which is generated by the above grammar ={string of the form a b a b^n c for 0≦n≦2}

【說明】依上述文法的各種情形分別展開之，產生的字串如右：a b a b c、a b a b b c及a b a c，這些字串的通式是a b a b^n c (其中0≦n≦2)。

- 【例四】Is the following grammar an ambiguous grammar?

  [Yes or no only]

  E→E + E | E * E | (E) | - E | <id>

  Where E represents an abbreviation for expression, and <id> represents an identifier.

【解】Yes, the above grammar is an ambiguous grammar.

【說明】本例的文法對於某些指述(statement)的展開具有一種以上的導出方式，所以此文法是語意不明確之文法(ambiguous grammar)；例如：<id> * <id> + <id>利用上述文法展開具有一種以上的導出方式，分別以剖析樹(parse tree)表示如下：