第二章　語言表示法

- 描述語法最常用的方法為 EBNF（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。

- 本章介紹 EBNF 表示法。

2.1 語言表示法

- 像 BNF 用來描述另一種語言，稱為中繼語言（meta-language），所使用的符號稱為中繼符號（meta-symbol）。BNF 的中繼符號有下列三種：
- 1. **::=** 定義為。
- 2. **|** 或，多項中選擇一項。
- 3. **{ }** 重複發生 0 次、1 次、2 次、...、無窮多次。
- 後來為了描述方便起見，增加下列三種，稱為 EBNF 表示法。
- 4. **< >**
- 非終端符號（non-terminal symbol）。
- 5. **[ ]** 選擇（optional）。
- 方括號內之語法項目可選可不選。
- 6. **\ **
- 必須選一項，且只能選一項。

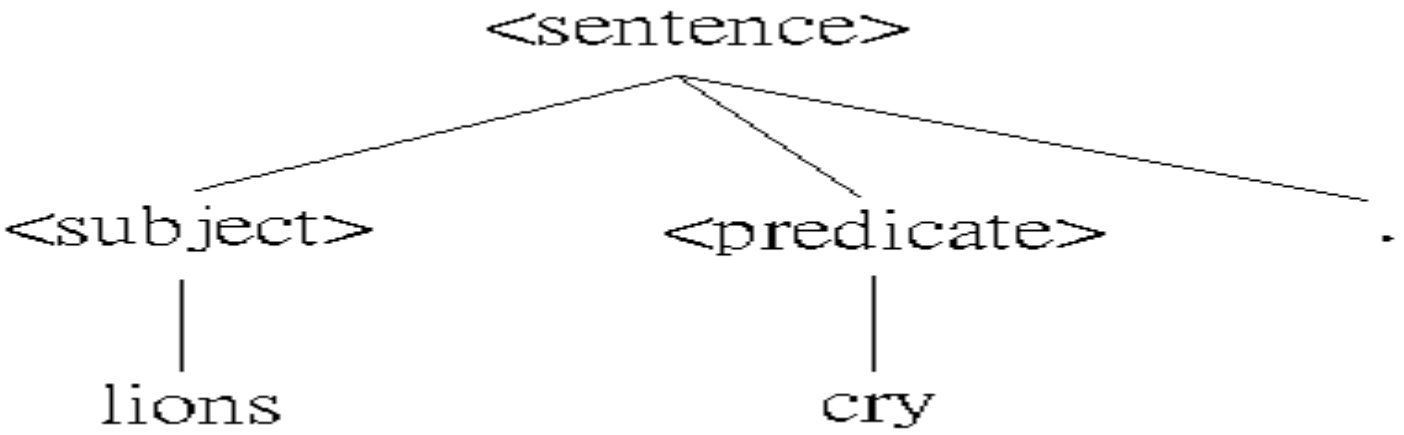
- 【例2-1】
-
- 英語簡單語句由下列三條語法規則所組成。
-
- P1 <sentence> ::= <subject> <predicate> .
- P2 <subject> ::= lions | cats
- P3 <predicate> ::= cry | fly
-
- 語法規則（syntax rule）也稱為產生規則（production）。

- ***語法規則 P1*** 描述一個英語簡單語句，定義為主詞 <subject> 非終端符號，其後跟著表詞 <predicate> 非終端符號，其後再跟著句點終端符號。合乎 P1 語法規則為正確，否則為錯誤。
- ***語法規則 P2*** 說明主詞為終端符號 lions 或 cats 二者選一。
- ***語法規則 P3*** 說明表詞為終端符號 cry 或 fly 二者選一。
- 英語簡單語句 <sentence> 只有下列四種是合法的，其餘為錯誤。
- lions cry.
- lions fly.
- cats cry.
- cats fly.

2.2 語法分析

- 辨認語句是依輸入的本文，導出（derive）產生該語句的剖析程序。一般稱此工作為語法分析（syntax analysis）。其工作通常很繁雜，有時甚至不可能做到。而用那一類型的剖析方法來定義語言是影響辨認工作是否困難的最主要原因。常用的為由上而下（top down）的剖析方法，或由下而上（bottom up）的剖析方法。
-
- 由上而下的剖析方法，其法由起始符號找出產生整個語句的一系列剖析程序。

- 例如下列句子：
- lions cry.
- 要剖析該句是否合乎英語簡單語句。
-
- 從非終端符號 <subject> 起始符號集合 {lions, cats} 開始，請注意這裡的大括號表示集合（set），不是 EBNF 記號。集合中共有二個元素，lions 及 cats。由 P2 產生規則知道非終端符號 <subject> 可以直接產生終端符號 lions。由 P3 產生規則知非終端符號 <predicate> 可以直接產生終端符號 cry。故知「lions cry.」為合法之英語簡單語句。
- 以剖析樹（parsing tree）表示如下圖所示。



- 【例2-2】
- S 語言以 EBNF 定義如下。請剖析字串 xyz 是否合乎 S 語法。
- P1 <S> ::= x <A>
- P2 <A> ::= z | y <A>

- 【例2-3】
-
- <Condition> 語法以 EBNF 定義如下。請剖析本文 ab < cd 是否合乎 <Condition> 語法。
-
- P1 <Condition> ::= <Id> \<|=|>\ <Id>
- P2 <Id> ::= <Alpha> { <Alpha> }
- P3 <Alpha> ::= a | b | c | d

2.2.1 起始符號規定

- 若有一語法規則如下：
- <A> ::= <A1> | <A2> | <A3> | ... | <An>
- 則所有能從 <Ai> 產生語句的起始符號集合之交集必須是空集合。要取得 <Ai> 的起始符號集合，可用下列兩條規則。
- ***規則一***
- **若起始符號就是終端符號時，**
- **則為該終端符號之集合。**
- ***規則二***
- **若起始符號為非終端符號時，則為所有**
- **非終端符號之起始符號集合之聯集。**

自然排除左遞迴

- 這樣的起始符號規定自然排除左遞迴（left recursive）的語法。例如下列的語法規則：
- <A> ::= <A> a | b
- 則
- start(<A>a) ∪ start(b) = {b} ∪ {b} = {b}
- <A>a 與 b 的起始符號均為 {b}，違反起始符號交集必須是空集合的規定。
-
- 不過還好，這種左遞迴的語法都能改成重複的語法，可改成非左遞迴的語法如下：
- <A> ::= b {a}

- 【例2-5】
- T 語言定義如下。請求出 <T>，<A>，<B> 之起始符號集合。
- P1　<T> ::= <A> | <B>
- P2　<A> ::= x <A> | y
- P3　<B> ::= x <B> | z

修改左遞迴符合起始符號規定

- 非終端符號 <A> 與 <B> 的起始符號集合之交集為 {x}，不是空集合，違反起始符號規定。
- 語法若改為如下的規則：
- P1 <T> ::= <C> | x <T>
- P2 <C> ::= y | z
- 則符合起始符號規定。

2.2.2 跟隨符號規定

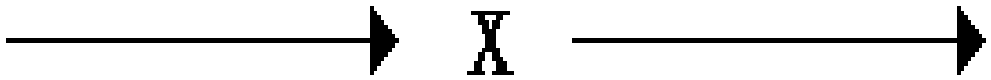
- 對於任一非終端符號 <A> 具有產生空符號能力時，則所有 <A> 之起始符號集合，與所有 <A> 之跟隨符號集合之交集，必為空集合。
- 關於 <A> 之跟隨符號集合 follow(<A>) 可依下列方法求得。
- 若
- <B> ::= α <A> β
- 則
- follow(<A>)=start(β)

2.3 語法圖

- 採用*由上而下*的剖析方法來做辨認的工作，是常用而且十分合適的。若配合語法圖（syntax graph）更可收事半功倍之效。由上而下之剖析，其特徵是由剖析非終端符號的起始符號開始，直到全部轉換成終端符號為止。非終端符號只存在於導出過程，因此若能對所有非終端符號均建立其相對應之語法圖，即可完成剖析。

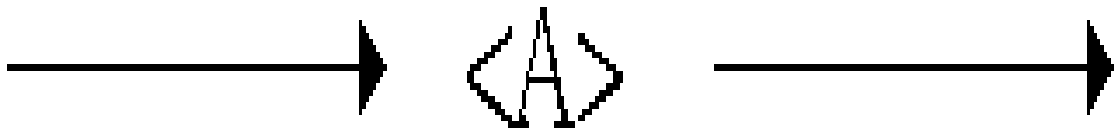
2.3.1 語法圖 G1

- 若 <A> ::= x 為終端符號 x，則其語法圖如下圖所示。



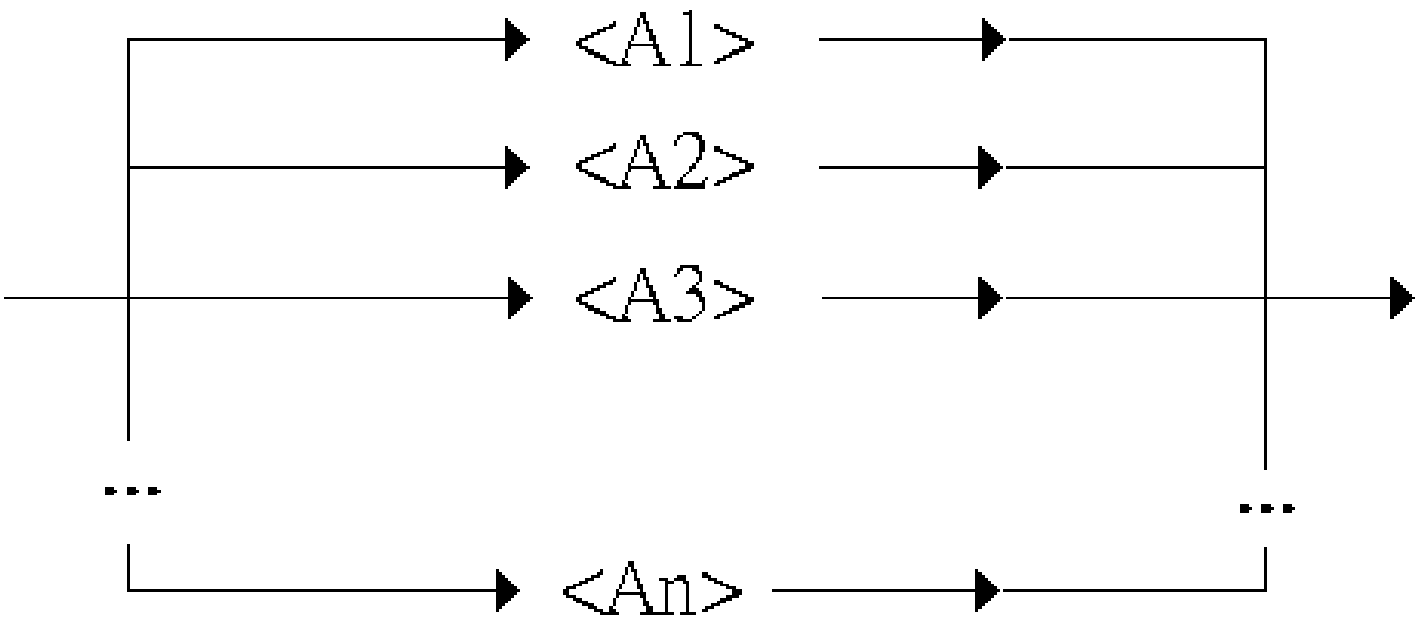
2.3.2 語法圖 G2

- 若 <A> 為非終端符號，則其語法圖如下圖所示。



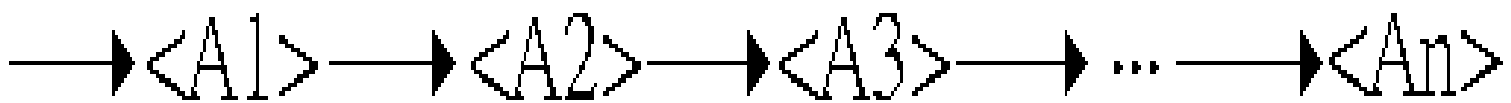
2.3.3 語法圖 G3

- 若 <A> ::= <A1> | <A2> | <A3> | ... | <An> 則其語法圖如下圖所示。



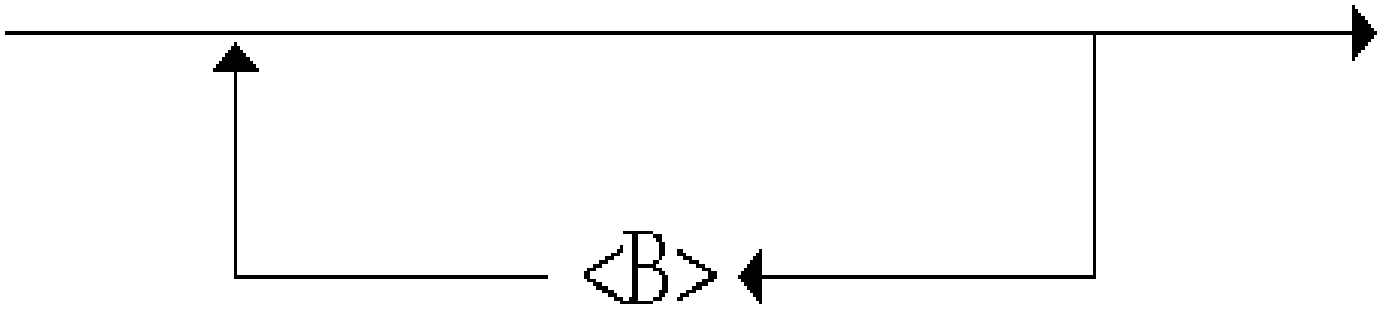
2.3.4 語法圖 G4

- 若 <A> ::= <A1> <A2> <A3> ... <An> 則其語法圖如下圖所示。



2.3.5 語法圖 G5

- 若 <A> ::= {<B>} 則其相對應之語法圖如下圖所示。



2.4 語法圖轉換為程式

- 語法圖也可以說是程式流程圖。雖然如此，乃需要一套嚴僅的轉換規則將語法圖轉換為程式。為了簡潔起見，假設欲剖析之本文存放在字元陣列 line[] 之中，nextChar 為 line[] 陣列中位置 cp 所指的字元，nextChar 字元由方法 advance() 所提供。語法錯誤之報告訊息由方法 error() 所提供。
- **line[ ]　　欲剖析之本文存放在字元陣列。**
- **cp　　字元陣列 line[] 之字元位置。**
- **nextChar　為 line[ ] 陣列中位置 cp 所指字元。**
- **advance()　傳回下一個字元 nextChar。**
- **error()　　傳回語法錯誤之訊息。**

2.4.1 語法規則 T1

- 若 <A> ::= x，但 x 為終端符號。其轉換程式如下：
-
- if (nextChar=='x')
- advance();
- else
- error();

2.4.2 語法規則 T2

- 若 <A> 為非終端符號，則其語法轉換程式如下：
-
- A();

2.4.3 語法規則 T3

- 若 <A> ::= <A1> | <A2> | <A3> | ... | <An> 則其語法轉換程式如下：
-
- switch (nextChar)
- {
- case start(<A1>) : A1(); break;
- case start(<A2>) : A2(); break;
- case start(<A3>) : A3(); break;
- ...
- case start(<An>) : An(); break;
- default : error();
- }

2.4.4 語法規則 T4

- 若 <A> ::= <A1> <A2> <A3> ... <An>, 則其語法轉換程式如下：
-
- A1();
- A2();
- A3();
- ...
- An();

2.4.5 語法規則 T5

- 若 <A> ::= {<B>}，則其相對應之語法轉換程式如下：
-
- while (nextChar==start(<B>))
- {
- B();
- }

- 【例2-8】
-
- 語言 E 以 EBNF 法表示如下。
-
- P1　<E> ::= <T> { \ + | - \ <T> }
- P2　<T> ::= <F> { \ * | / \ <F> }
- P3　<F> ::= x | y | z
-
- 1. 請列出所有終端及非終端符號。
- 2. 列出每一非終端符號之起始符號及跟隨符號之集合。
- 3. 製作相對應之語法圖。
- 4. 設計剖析程式 eparser.c。
-