程式語言的語法

資科系
林偉川

基本定義

- 字元集：

  一組有限符號的集合稱之為字元集。一般的字元集有二類，一類是 ASCIICode Set，一類是 EBCDIC Code Set。

基本定義

– ASCII Code Set

- 是American Standard Code for Information Interchange 的縮寫。
- 標準的 ASCII Code 有7個位元 可表示 $2^7 = 128$ 種不同的字元。
- 一般使用在 IBM PC 及Apple II上。
- 現今使用的 ASCII Code 已經擴充為8個位元，稱之為ASCII-8。

3

基本定義

– EBCDIC Code Set

- 是Extended Binary Code Decimal Interchange Code 的縮寫
- 標準的EBCDIC Code有8個位元可表示2^8=256 種不同的字元
- 一般是使用在IBM 360及FACOM機器上

4

基本定義

- 定義
  - $S=t_1t_2...t_n$，$ti \in T$ 其中 T 為字元集
  - S 是由 T 中的字元所組成的一串列
    - n=4 則 S 可能為 abcd，ABCD，AEFG 等等
- 字串的長度
  - 假設 $S=t_1t_2...t_n$則 S 的長度可表為 $|S|=n$
    - S 的長度為 n

5

基本定義

- 二字串的連接

  設 p 與 q 為二字串且 $p=m_1m_2\ldots m_u$，$q=n_1n_2\ldots n_v$則

  $p \cdot q=m_1m_2...m_un_1n_2...n_v$

  上式即表示二字串的連接且 $|p \cdot q|=u+v$

  (表 p・q 字串的長度為 u+v)

6

基本定義

– 空字串：

通常以“ε”表示空字串，且｜ε｜=0，

有時空字串也可以“λ”表示。

– T*：

T* 表示為由 T 中的字元所組成任意長度的字串的集合。

例：設 T={p，q}

則 T* ={ ε ，p，q，pp，qq，pq，qp，ppp...}

7

基本定義

– 語言 (Language)：

設 L 為一語言，則 L 是 T* 的一組子集合 (subset)。

例：設 T={p，q} 則 L 可為 {p，pq，qp，...}或 {ppp，qqq，pqp，qpq，...} 等等，只要是 T* 的子集合即可。

– 語言 L 的次方 (Power) 定義如下：

L^o={ ε }　L^n=L • L^{n-1}

例：設 L={p，pq，q}

則 L^0={ ε }　L^1=L

L^2=L。L={pp，ppq，pq，pqp，pqpq，pqq，qp，qq}

8

基本定義

－L^*的定義：

L^* 又稱“Kleene Closure of L”是L 做任意次乘積 (product) 的集合。

$L^*=L^0 \cup L^1 \cup L^2 \cup ... \cup L^n \cup ...$

－L^+ 的定義：

L^+ 又稱”Transitive Closure of L”其定義如下：

$L^+=L^1 \cup L^2 \cup L^3 \cup ... \cup L^n \cup ...$

－二種語言 L1 與 L2 的乘積 定義如下：

$L1 \cdot L2=\{a \cdot b \mid a \in L1, b \in L2\}$

例：L1={p，q}　L2={m，n，mn，nm}則

L1・L2={pm，pn，pmn，pnm，qm，qn，qmn，qnm}

9

文法四要素

- 文法是由四個要素所組成
- T：終端符號表示不能再以其他符號來替代
- N(V)：非終端符號表示可以再以其他符號來替代，而N與T須具以下的關係：$N \cap T=\phi$
- S：起始符號:從事文法推演之步驟由S開始，$S \in N$
- P：文法產生規則，符合條件為$\forall u \rightarrow v \in P$，其中$u \in (N \cup T)^+$，$v \in (N \cup T)^*$

10

文法的分類

- Type 0：無任何限制。
- Type 1：與上下文相關的文法其文法產生規則必須滿足： $\forall u \rightarrow v \in P$，其中$|u| \leq |v|$
- Type 2(BNF)：與上下文無關的文法其文法產生規則必須滿足： $\forall u \rightarrow v \in P$，其中$u \in N$，$v \in (N \cup T)^* - \lambda|$
- Type 3：正規文法 (regular grammar)

11

語法與語意

- 語法(Syntax)之描述方法
  - Context-free grammar (推移圖)
  - Backus-Naur Form(BNF)
  - 語法圖
  - Attribute grammar 描述語法與靜態的語意
- 語意(Semantics)之描述方法
  - 靜態語意(static semantics)
  - 動態語意(dynamic semantics)
    - 解釋型語意(interpretive semantics)
    - 公理型語意(axiomatic semantics)
    - 符號型語意(denotational semantics)

12

正規文法分類

- 右線性正規文法(right linear regular grammar)
  - 文法產生規需滿足
    - Au→B or A → u，其中 A,B∈N，u ∈ T
- 左線性正規文法(left linear regular grammar)
  - 文法產生規則需滿足
    - A→Bu or A → u，其中 A,B∈N，u ∈ T

13

題目

有一語法(Grammar)G={N(V)，T，S，P}，其中N表非終端符號之集合, T表終端符號之集合, S為起始符號, P表產生規則之集合。該語法若為與前後文無關, 那麼每一產生規則 $\alpha \rightarrow \beta$, α 與 β 需滿足那個條件 ？ (a) $\alpha \in S, \beta \in T$. (b) $\alpha \in N, \beta \in (NUT)$ (c) $\alpha \in N, \beta \in (NUT)^*$ (d) $\alpha, \beta \in (NUT)^*$

答:C

14

題目

是非題

一個type 0文法需要右邊所有文法規則要與左邊的非終端符號一樣多.

答:錯，無任何限制。

15

B.N.F. 文法

- B.N.F.(Backus Naur Form) grammar
  - type 2 grammar
  - context-free grammar
- B.N.F.文法符號說明:
  - "::=":表示"定義為" ，相當於➔。
  - "{}":表示出現0次, 1次, . . . 。(EBNF才有)
  - "[]":表示出現0次或1次。 (EBNF才有)
  - "|":表示"OR"。
  - "< >":表示非終端符號。

16

範例

B.N.F.文法規則定義如下：

G={ {expr}, {+,-,*,/,(,),a,b,c,d,e,f}, expr, P}

P1.<expr>::=<expr>+<expr>

P2. <expr>::= <expr>-<expr>

P3. <expr> ::= <expr>*<expr>

P4. <expr>::= <expr>/<expr>

P5. <expr>::= (<expr>)

P6. <expr>::= a | b| c | d| e | f

請推導出a+b*c-(d+e)/f

17

範例

請利用下列B.N.F.文法來推導出最下面指令

G={ {assign, id, exp, term, factor},
{:=, +, -, *, DIV, (,), a, b, c, d, 10}, assign, P}

P1 <assign>→<id>:=<exp>

P2 <exp>→<term> | <exp>+<term> | <exp>-<term>

P3 <term>→<factor>|<term>*<factor>|<term>DIV<factor>

P4 <factor>→<id> | (<exp>)

P5 <id> → a | b | c | d | 10

指令:

a:=b DIV 10 + c*d

18

推導過程

- <assign> ::= id := <exp>
  := <exp> + <term>
  := <term> + <term>
  := <term> DIV <factor> + <term>
  := <factor> DIV <factor> + <term>
  := id DIV <factor> + <term>
  := id DIV int + <term>
  :=id DIV int + <term> * <factor>
  :=id DIV int + <factor> * <factor>
  := id DIV int + id * <factor>
  := id DIV int + id * id

19

剖析樹

剖析樹(parsing tree)的定義：

根據語言的B.N.F.描述，將運算式轉換成相對應的樹狀結構，則稱此樹狀結構為剖析樹。

20

剖析樹

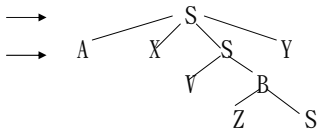
茲有下列文法：G={{S,A,B},{x,y,v,u,z, ε }, S, P}

S→AxSy｜vB

A→u｜ ε

B→zS｜ ε

其中 S, A, 及B為非終端符號。請建構字串 **AxvzSy** 相對應的剖析樹



21

範例

B.N.F.文法規則定義如下：

G={ {expr}, {+,-,*,/,(,),a,b,c,d,e,f}, expr, P}

1.<expr>::=<expr>+<expr>

2. <expr>::= <expr>-<expr>

3. <expr> ::= <expr>*<expr>

4. <expr>::= <expr>/<expr>

5. <expr>::= (<expr>)

6. <expr>::= a | b | c | d| e | f

請到推導出a+b*c-(d+e)/f，並建構相對應的剖析樹(a+b*c)

22

範例

請利用下列B.N.F.文法來建構最下面指令及相對應的剖析樹

<assign>::= <id>:=<exp>

<exp>::=<term> | <exp>+<term> | <exp>-<term>

<term>::=<factor> | <term>*<factor> | <term> DIV <factor>

<factor>::=<id> | (<exp>)

<id> ::= a | b | c | d

指令:

a:=b DIV 10 + c*d

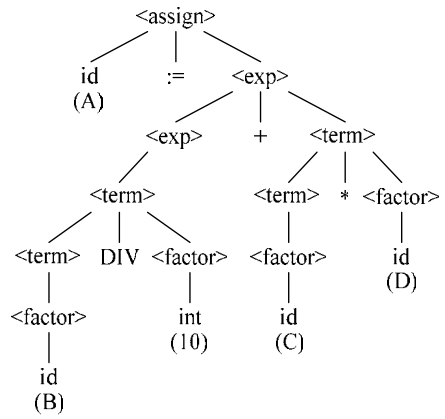
23

推導過程

- <assign> ::= id := <exp>
  := <exp> + <term>
  := <term> + <term>
  := <term> DIV <factor> + <term>
  := <factor> DIV <factor> + <term>
  := id DIV <factor> + <term>
  := id DIV int + <term>
  :=id DIV int + <term> * <factor>
  :=id DIV int + <factor> * <factor>
  := id DIV int + id * <factor>
  := id DIV int + id * id

24

剖析樹



25

模擬兩可的文法(ambiguous grammar)

- 模擬兩可的文法的意義:

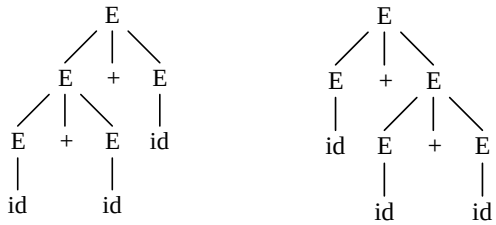
  根據語言的 B.N.F. 描述，**對同一句子(sentence)可繪出二個或二個以上不同的剖析樹**，則稱此語言的文法是模擬兩可的。

26

模擬兩可的文法

範例：

請利用文法E→E+E｜id，畫出指令id+id+id之兩個不同的剖析樹，根據題意所給的條件會產生模擬兩可



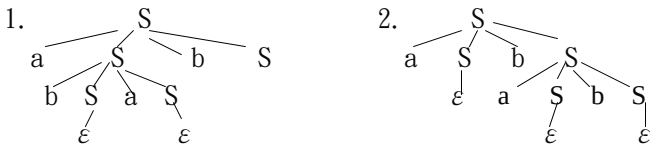
27

模擬兩可的文法

範例：

試證明下列文法為模擬兩可的?

S→aSbS｜bSaS｜ ε ，例如：abab



28

文法練習

- 試證明下列文法為模擬兩可的? id+id+id

G={{S,A,id}, {a,b,c,+}, S, P}

1.<S>➔ <A>

2. <A>➔<A>+<A> | <id>

3. <id> ➔ a | b | c

請畫出其剖析樹？

29

文法練習

- B.N.F.文法規則定義如下：

G={ {assign,expr,id,term,factor}, {:=,+,-,*,/,(,),a,b,c,d}, assign, P}

1.<assign>::= <id> = <expr>;

2. <id>::= a | b | c | d

3. <expr> ::= <expr>+<term> | <expr>-<term> | <term>

4. <term> ::= <term>*<factor> | <term>/<factor> | <factor>

5. <factor> ::= (<expr>) | <id>

請畫出下列語言的剖析樹？

(1) a=b+(c*a); (2) a=(a+b)*c; (3) a=b+c+a; (4) a=a*(b+c); (5) a=b*(c*(a+b));

30

懸置else問題

- 懸置 else(dangling else)的意義

  若有一敘述如下:

  “if 條件A then if 條件B then E1 else E2” 則 else 將無法確定要與第一個或第二個 if 結合，這種現象就是所謂的dangling else問題。
- 即：*if* 條件A then if 條件B then E1 *else* E2

  或 if 條件A then *if* 條件B then E1 *else* E2

31

模擬兩可的文法if

<stmt> → <if_stmt>| <assign>

<if_stmt> → if <logic_expr> then <stmt> |

if <logic_expr> then <stmt> else <stmt>

<stmt> → <if_stmt> | <assign>

```
b=12* 23-  89  +  64;
if a>=b then
  if d>=c then
    c=b+c;
  else
    a=d+b;
```

32

利用最接近未結合原則來解決

<stmt> → <if_stmt> | <assign>

<if_stmt> → <matched> | <unmatched>

<matched> →if <logic_expr> then <matched> else <matched> | <assign>

<unmatched> → if <logic_expr> then <stmt> | if <logic_expr> then <matched> else <unmatched>

<logic_expr> → <id> > <id> | <id> < <id> | <id> >= <id> | <id> <= <id> | <id> == <id> | <id> != <id>

33

懸置else問題

- 各種語言解決 dangling else 的方法
  - PASCAL
    - 利用begin-end作為分界，來解決懸置else問題
  - ALGOL 60
    - 利用begin-end作為分界，來解決懸置else問題
  - ALGOL 68
    - 利用if...fi作為分界，來解決懸置else的問題
- 近代高階程式語言
  - 多利用最接近未結合原則來解決此問題

34

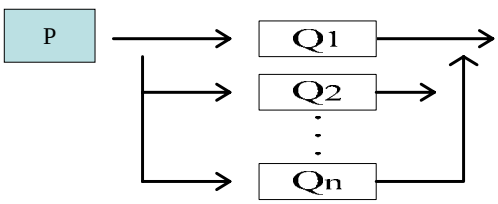
描述程式語言語法的方式

- B.N.F.法
- 語法圖(Syntax Diagram)
- 推移圖(Transition Diagram)

35

語法圖

- □：表示非終端符號
- ○：表示終端符號
- P::=Q1｜Q2｜・・・｜Qn

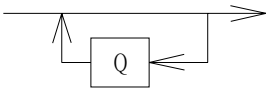


36

描述程式語言語法的方式

b. P::= Q1 Q2 ··· Qn

c. P::= {Q}



d.  ： 表示非終端符號

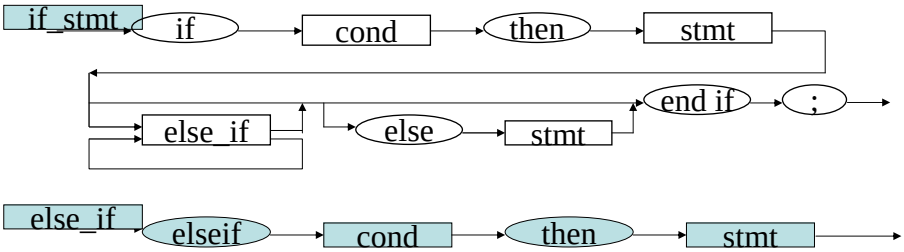
e.  ： 表示終端符號

語法圖

EBNF

<if_stmt>→if <cond> then <stmt> {<else_if>}
[else <stmt>] endif;

<else_if> →elseif <cond> then ＜stmt＞

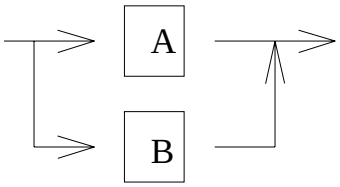


範例

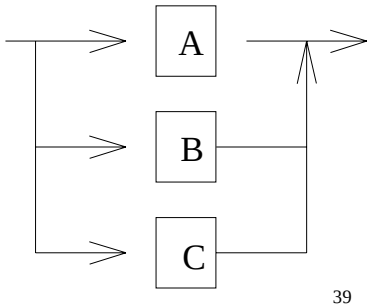
請寫出下列語法圖所對應的BNF:

(a) P::= A | B　　(b) P::= A | B | C

(a)



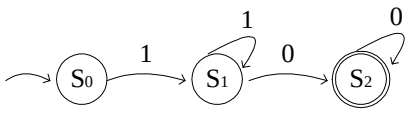
(b)



39

推移圖

- $1^{+}0^{+}$對應的推移圖

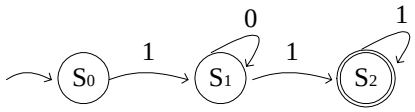


其中S_0為起始狀態，而S_2為終止狀態

40

推移圖

- 10^*1^+ (Regular expression)對應的推移圖

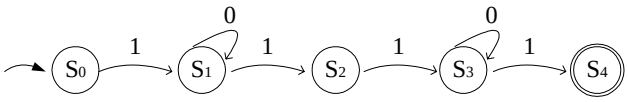


其中S_0為起始狀態，而S_2為終止狀態

41

推移圖

- 10^*110^*1 對應的推移圖



其中S_0為起始狀態，而S_4為終止狀態

42

語意的描述

- 靜態語意(static semantics)
- 動態語意(dynamic semantics)
  - 解釋型語意(interpretive semantics)
  - 公理型語意(axiomatic semantics)
  - 符號型語意(denotational semantics)

43

靜態語意

- 意義
  - 由於有許多語言規則沒有辦法單純的用BNF文法來做一個描述，因此必需要用其他的規則來加以處理
- 作法
  - 在程式執行之前或語言處理器翻譯時處理
- 範例
  - 變數必須先宣告再引用
- 可用屬性文法(attribute grammar)表達
  - 屬性文法是一種可以描述靜態語意之方法

44

屬性文法

- **定義: 一個屬性文法是一個 CFG G = (V, T, S, P) 有下列新增項目:**
  - **對每一文法符號x，有一個屬性值集合 A(x) (Synthesized or Inherited attributes)**
  - **對每一文法規則的非終端符號，有一個意義函式 (semantic functions) 集合**

    對文法規則 X0 → X1…Xn 其X0之 synthesize attribute 是以一意義函式S(X0)=f(A(X1),…,A(Xn)) 來運算。其屬性值由其子節點來決定 (藉由剖析樹來傳達意義資訊)

45

屬性文法範例

- **Ada程式語言之副程式名稱要與副程式之*end*後的名稱相同**

  String attribute of <proc_name> : <proc_name>.string

  文法規則:

  <proc_def> → procedure <proc_name>[1]
  <proc_body>
  ***end* <proc_name>[2]**

  語意規則:

  <proc_name>[1].string = <proc_name>[2].string

46

屬性文法範例

- 變數的設定值，變數設定的LHS及RHS並不一定是同一個資料型態
- 運算式id1=id2 + id3
  - id2 & id3 可以是 int_type or real_type
  - 兩個 id's必需同一型態
  - 運算後的型態必需符合所期盼的型態 (expected_type)

47

屬性文法範例

*BNF (*Syntax rule)：

1.<*assign*> → <*var*> = <*expr*>

2.<expr> → <var> + <var>

3. <expr> → <var>

4. <var> → A | B | C

Attributes:

1. Syntax rule: <assign> → <var> = <expr>
   Semantic rules:
   <expr>.expected_type ← <var>.actual_type
   由LHS type決定

48

屬性文法範例

2. Syntax rule: <expr> → <var>[2] + <var>[3]

Semantic rules:

<expr>.actual_type ←
if (<var>[2].actual_type = int) and
(<var>[3].actual_type = int) then int
else real

Predicate:

<expr>.actual_type = <expr>.expected_type

49

屬性文法範例

3. Syntax rule: <expr> → <var>

Semantic rule:

<expr>.actual_type ← <var>.actual_type

Predicate:

<expr>.actual_type = <expr>.expected_type

50

屬性文法範例

4. Syntax rule: <var> → A | B | C
Semantic rule:
<var>.actual_type ← Look-up(<var>.string)

Look-up 函式則為在符號表(symbol table)中，尋找一個變數名稱及其傳回的資料型態

51

屬性文法範例

如何計算屬性值?

1. 如果所有的屬性是繼承的(inherited), 則剖析樹是以由上到下的推導方式
2. 如果所有的屬性是推導的(synthesized), 則剖析樹是以由下到上的推導方式
3. 在許多情形下, 以上兩種屬性皆有使用, 而且是由上到下及由下到上推導方式的合併作法.

52

屬性文法範例

Ex. A=A+B

1. <var>.actual_type ← Look-up(A) [Rule 4]

2. <expr>.expect_type ← <var>.actual_type [Rule 1]

3. <var>[2].actual_type ← Look-up(A) [Rule 4]

<var>[3].actual_type ← Look-up(B) [Rule 4]

4. <expr>.actual_type ← 是 int 或 real [Rule 2]

5. <expr>.expected_type =<expr>.actual_type 是 TRUE 或 FALSE [Rule 2]

53

屬性文法總結

- 屬性文法特性
  - 屬性文法是用來描述一個程式語言的語法及其固定語意.
- 屬性文法優點
  - 屬性文法可用來正式定義一個語言，並可當做編譯器產生系統的輸入.

54

屬性文法總結

- 屬性文法缺點
  - 很難描述一個程式語言所有的語法及固定語意，是因為其複雜度很高及要耗費很多記憶體.
  - 因為要描述其屬性及語意規則相當多，使得屬性文法很難讀得懂及寫得出來.
  - 屬性文法當應用在大型的剖析樹時，其代價相當大.

55

動態語意

- 解釋型語意(interpretive semantics)
- 公理型語意(axiomatic semantics)
- 符號型語意(denotational semantics)

56

解釋型語意

- 意義
  - 又稱為操作型語意(operational semantics) ,在本方法中定義抽象機器，此抽象機器可支援一組簡單的操作並提供一些簡單的資料結構供使用。通常以暫存器或記憶體位置來表示計算機執行的狀態
- 作法
  - 語言的語意在此類型中，被定義成為如何將程式轉換成在抽象機器上執行的碼
- 範例
  - 如PL/1語言的維也納定義語言(Vienna Definition Language)

57

解釋型語意範例

C指令:

for (expr1; expr2; expr3) { … }

操作型語意

```
        expr1;
Loop: if expr2 = 0  goto out
        …
        expr3;
        goto loop
Out:  …
```

58

解釋型語意範例

簡單控制指令元件:

ident=var
ident=ident+1
ident=ident-1
goto label
if var relop var goto label ***relop:*** **>, >=, <, <=, =, <>**

變數設定指令

ident=var bin_op var ***bin_op:*** **+, -, *, /**
ident=un_op var ***un_op:*** **+, -**

59

公理型語意

- 意義
  - 公理型語意提供數學規則來表示程式執行之結果。
- 作法
  - 對於程式語言的每個語法單元均提供一個數學規則來定義。也就是利用數學推論來証明程式的正確性

60

公理型語意範例

前後條件型式: **{P} statement {Q}**

- ***範例1:*** **{b > 10} a := b + 1 {a > 1}**

  一種可能的前條件: {b > 10}

  最不可能的前條件: {b > 0}

  { b > 0} ⊃ {b > 10} or { b > 0} => {b > 10}

- ***範例2:*** **{b < 22} a=b/2-1 {a < 10}**

  {a < 10} →後條件

  b/2-1 < 10 →最不可能的前條件計算

  {b < 22} →前條件

61

公理型語意範例

- ***範例3:*** **{y > 14} x=2*y-3 {x > 25}**

  {x > 25} →後條件

  2*y-3 > 25 →最不可能的前條件計算

  {y > 14} →前條件

- ***範例4:*** **{y > 13-x} x=x+y-3 {x > 10}**

  {x > 10} →後條件

  x+y-3 > 10 →最不可能的前條件計算

  {y > 13-x} →前條件

62

符號型語意

- 意義
  - 符號型語意利用數學函數來定義程式。
- 作法
  - syntax entity → object

63

符號型語意範例

- **十進位數字的 CFG規則:**

  <dec_num> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
  | <dec_num> (0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9)

符號型語意針對上述規則如下**:**

M_{dec}('0') = 0, M_{dec} ('1') = 1, …, M_{dec} ('9') = 9
M_{dec} (<dec_num> '0') = 10 * M_{dec} (<dec_num>)
M_{dec} (<dec_num> '1') = 10 * M_{dec} (<dec_num>) + 1
…
M_{dec} (<dec_num> '9') = 10 * M_{dec} (<dec_num>) + 9

64