計算機言語講義資料 第2回
プログラミング言語の構文 (続き)

大阪大学 大学院情報科学研究科
2015年
長谷川亨
t-hasegawa@ist.osaka-u.ac.jp

1

演習問題【1】の回答

【1】$L_1 = \{ a^n b^m \mid n \geq m \geq 1\}$ とする。L_1を生成するCF文法を求めよ。

回答

$G_3 = \{N=\{S\}, \Sigma = \{a, b\}, P=\{S\rightarrow aS, S\rightarrow aSb, S\rightarrow ab\}, S\}$

2

演習問題【2】

L_3={$a^nb^mc^m$, $a^nb^nc^m$ | n, m≥1}とする。L3を生成するCF文法を与え，つぎに終端記号列abcの解析木をすべて求めよ。

解析木

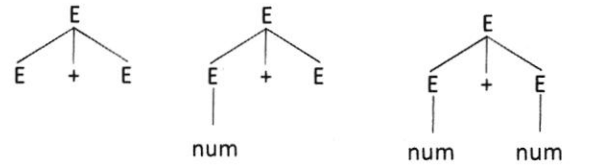
解析木(または導出木, 構文木)は, 導出列を木表現したもの

作り方

- 頂点は開始記号
- 適用した生成規則$A \rightarrow B_1B_2B_3 \cdots B_k$に対して, 頂点をk個作り左から順に$B_1, B_2, B_3, \cdots, B_k$を割り当て
- 以降、再帰的に繰り返し

例　$num + num$

$E \Rightarrow E+E \quad \Rightarrow \quad num+E \quad \Rightarrow \quad num+num$



結合性

結合則と可換則

ある集合Sと演算・において、任意のa, b, c∈Sにおいて、

$(a \cdot b) \cdot c = a \cdot (b \cdot c)$ がなりたつ　結合則

$a \cdot b = b \cdot a$ がなりたつ　可換則

結合性

同一の演算子がならんでいるときに、どの演算を先に実行するか

左の方が先：　左結合的

右の方が先：　右結合的

テキストには記載されていません

5

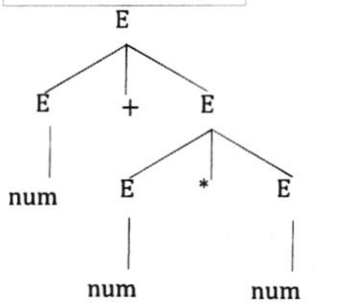
曖昧なCF文法

一つの終端記号列に対して、二つ以上の解析木ができるとき、その文法は**曖昧**である

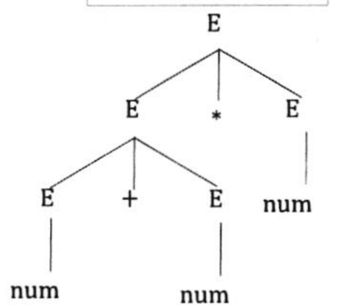
*num+num*num*の解析木

$G = \{N = \{E\}, \Sigma = \{num, +, *, (,)\}$
$P = \{E \rightarrow E+E,\ E \rightarrow E*E,$
$E \rightarrow num,\ E \rightarrow (E)\},$
$S = E\}$

*が+より優先度が高い



+が*より優先度が高い



6

文法の曖昧さの除去

曖昧なCF文法

$G = \{N = \{E\}, \Sigma = \{num, +, *. (,)\}$

$P= \{ E \rightarrow E+E \quad E \rightarrow E*E \quad E \rightarrow num,\ E \rightarrow (E) \}, S=E\}$

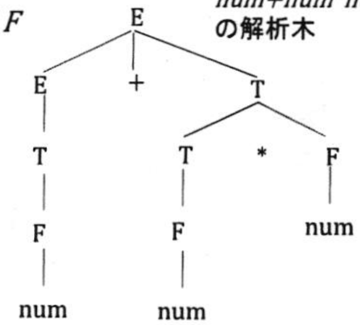
曖昧さを除去したCF文法の例

$G3 = \{ N=\{E, T, F\}, \Sigma = \{num, +, *, (,) \}, P, E \}$

$P= \{ E \rightarrow E+T | T \quad T \rightarrow T*F \,/\, F$

$F \rightarrow (E) \,/\, num \}$

*num+num*num*
の解析木



曖昧さを除去するための規則

(1) +より*が高い優先順位

(2) +は左結合的

(3) *は左結合的

曖昧さのない生成規則

曖昧なCF文法*G*　例3.4

$G = \{N = \{E\}, \Sigma = \{=, +, -, *, /, \uparrow, (,), num\},$

$P, E\}$

$P= \{ E \rightarrow E=E$

$E \rightarrow E+E | E-E$

$E \rightarrow E*E | E/E$

$E \rightarrow E \uparrow E$

$E \rightarrow (E)$

$E \rightarrow num \ \}$

曖昧さのない生成規則

式の生成規則の集合から、曖昧さを無くす方法
- 二項演算子の優先順位を決定する
- 優先順位の低い順でi(1≦i≦n)番目の二項演算子をΘ_i とする
- Θ_iを直接生成する生成規則の左辺をE_iとする
  - Θ_iが左結合的であれば　$E_i \to E_i\ \Theta_i E_{i+1} | E_{i+1}$
  - Θ_iが右結合的であれば　$E_i \to E_{i+1} \Theta_i E_i | E_{i+1}$
  - Θ_iが非結合的であれば
    $E_i \to E_{i+1} \Theta_i E_{i+1} | E_{i+1}$

9

曖昧さのない生成規則

曖昧さのないCF文法*G'*

演算子の優先順位 $\uparrow > \{*, /\} > \{+, -\} > =$

*、/、+、-は左結合的、↑は右結合的、=は非結合的

$G' = \{N' = \{E_i | 1 \leqq i \leqq 5\},$

$\Sigma = \{=, +, -, *, /, \uparrow, (,), num\}, P', E_1\}$

$P' = \{ E_1 \to E_2 = E_2 \,|E_2$

$E_2 \to E_2 + E_3 \,|E_2 - E_3 \,|E_3$

$E_3 \to E_3 * E_4 | E_3 / E_4 \,|E_4$

$E_4 \to E_5 \uparrow E_4 | E_5$

$E_5 \to (E_1) \,|num \ \}$

10

曖昧さのない生成規則

演習問題【3】 例3．4の曖昧際の無い生成規則を用いて、

*num-num*num↑num↑num*の解析木を与えよ。

C言語におけるif文の曖昧さの除去

無曖昧なCF文法を構成するアルゴリズムは存在する

しかしながら、通常のプログラミング言語の構文は必ずしも無曖昧な文法で記述されていない

C言語のif文の文法を以下とする。

<文>→if (<式>) <文>

<文>→if (<式>) <文> else <文>

if (b) if (b') s else s' を生成する構文木は？

C言語におけるif文の曖昧さの除去 (続き)

elseがどちらのifに対応するかに応じて、2通りの解釈がある ➡ 上記の文法は曖昧

if (b) { if (b') s else s'}

if (b) { if (b') s } else s'

C言語では、elseはこれに一番近いifと対応させる規則を導入することで、曖昧さを除去

if (b) { if (b') s else s'} が正しい解釈

13

構文図式

14

拡張CF文法

拡張CF文法

G=(N, Σ,P,S)の四つ組

N 非終端記号の有限集合,

Σ 終端記号の有限集合,

P 生成規則の有限集合,

S∈N 開始記号

生成規則の右辺の記述に正規表現(非決定性有限オートマトン)を許す点が、CF文法との差

生成規則は *A→e* 、 *e*は*(N ∪Σ)*の正規表現

15

正規表現

アルファベット$A=\{a_1, a_2, ..., a_n\}$上の正規表現

- *Φ*は正規表現である。これは空集合 *Φ* を表す。
- a_j(*A*の任意の要素)は正規表現である。これはa_jのみからなる集合$\{a_j\}$を表す。
- XとYが正規表現ならば、
  - *X*/*Y*も正規表現である。これはXの表す集合とYが表す集合の和集合を表す。
  - *XY*も正規表現である。これはXに含まれる記号列にYに含まれる記号列をつなげてできる記号列の集合$\{ab/a \in X, b \in Y\}$を表す。
  - *X**も正規表現である。これは*X*に含まれる記号列を0個以上つなげて出来る文字列の集合$\cup\{Xn/n \geqq 0\}$を表す。
  - 上記の帰納的導出によって構成される記号列のみが正規表現である。

16

構文図式

文法を表現する方法の一つ
- 非終端記号は□に、終端記号は○に囲まれる
- 拡張CF文法の図式表現

構文図式の例

拡張CF文法

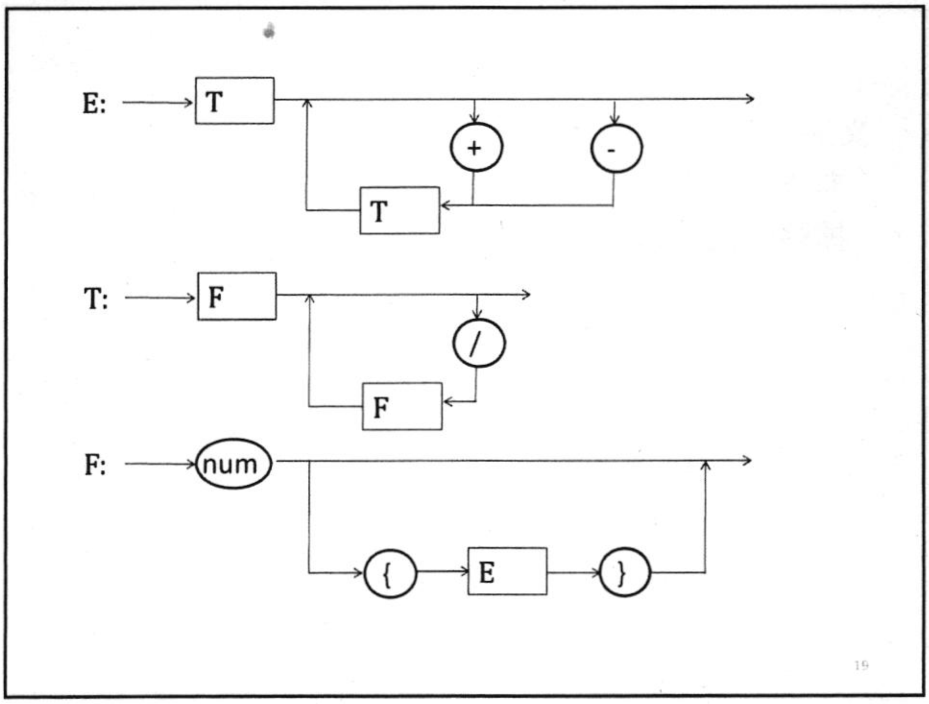
G={N={E,T,F}, Σ={num, +, -, /, '(', ')'}, P, E}

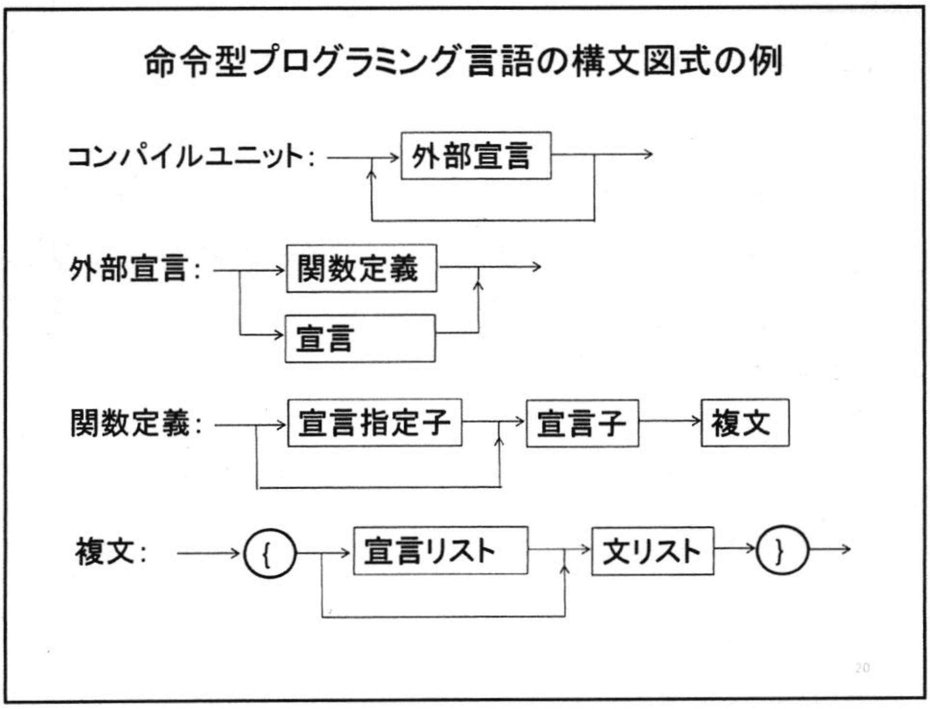
P={E→T ((+|-)T)   T→F(/F)*   F→num | '(' E ')'*

(,), | と*は正規表現を記述するための(メタ)記号。
非終端記号ではない

非終端記号から生成される非終端記号列の例

F, F/F, F/F/F, F/F/F, ...


E:
T
+
-
T
T:
F
/
F
F:
num
{
E
}
19


命令型プログラミング言語の構文図式の例
コンパイルユニット:
外部宣言
外部宣言:
関数定義
宣言
関数定義:
宣言指定子
宣言子
複文
複文:
{
宣言リスト
文リスト
}
20

式の補足

式の補足

式

演算子(operator)と被演算子(operand)から構成

「*a+b+c*という式を、計算機でどのように実行するか」

計算機では、一度にひとつの計算しかしない

$(a+b)+c$ と $a+(b+c)$ の2通り

テキストには記載されていません

式の補足 (続き)

演算子と優先順位

プログラミング言語毎に，演算子の強弱，あるいは優先順位が決められている

$a+b*c$が、 $a+(b*c)$ または $(a+b)*b$ の

どちらのように括弧をつけるかが決まる

演算子*が演算子+より優先順位が高い場合、

$a+(b*c)$ と解釈される

テキストには記載されていません

23

式の (続き)

【演習問題4】 C言語の演算子の優先順位(優先度)を調べよ。(復習)

「プログラミング言語C」のP65 表2-1

テキストには記載されていません

24

式の木表現

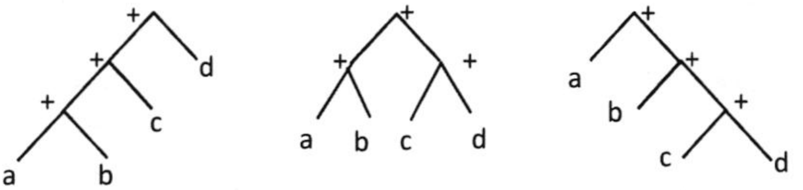
- 式を木で表現することは、評価法(評価順)を考える上で有用
- 木表現
  - 被演算子= 葉接点
  - 演算子= 内点(内部接点)
  - 2項演算子 =子接点を2つ持つ演算子
  - 単項演算子 =子接点を1つ持つ演算子

テキストには記載されていません

25

式の木表現

$a+b+c+d$ の木表現　　3通り



$((a+b)+c)+d$　　$(a+b)+(c+d)$　　$a+(b+(c+d))$

線形表現

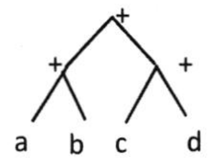
括弧を使うことによって、木の構造の情報を保存して線形表現できる

テキストには記載されていません

26

式の木表現

木表現は計算順序をかなり良く表現するが，完全ではない．例えば，中央の木表現では，*a+b* と *c+d* のどちらを先に評価するかを指定していない



$(a+b)+(c+d)$

テキストには記載されていません

27

木表現から線形表現へ

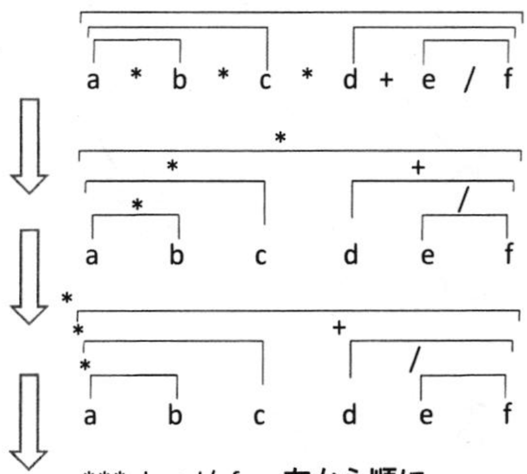
ポーランド記法

例) $a*b*c*d+e/f$

演算子を括弧の上へ

全ての演算子を括弧の前方に寄せる

前方に演算子を下す



***abc+d/ef　左から順に

テキストには記載されていません

28

ポーランド記法

ポーランド記法

前置記法・・・演算子はその対象とする被演算子の前に置く

後置記法・・・演算子はその対象とする被演算子の後に置く

「各演算子のとる被演算子数は、演算子毎に決まっていること」

この条件のもとで，式の演算順序は一意に決まり，括弧は必要としない

括弧無し表現と呼ばれる

(注) 後置記法は逆ポーランド記法と呼ばれる

数学での記法は，中置記法を用いることが多い

テキストには記載されていません

29

中置記法の木表現を辿ってポーランド記法を求める

前順・行きがけ順 (preorder)

1.根ノードを調査する。
2.もしあれば、左の部分木を前順走査する。
3.もしあれば、右の部分木を前順走査する。

ポーランド記法

中間順・通りがけ順 (inorder)

1.もしあれば、左の部分木を中間順走査する。
2.根ノードを調査する。
3.もしあれば、右の部分木を中間順走査する。

中置記法

後順・帰りがけ順 (postorder)

1.もしあれば、左の部分木を後順走査する。
2.もしあれば、右の部分木を後順走査する。
3.根ノードを調査する。

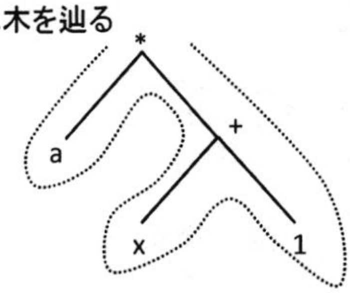
逆ポーランド記法

テキストには記載されていません

30

式の表現(続き)

- ポーランド記法　前置表現 ：前順に木を辿る
  **a +x 1*
- 逆ポーランド記法 後値表現：後順に木を辿る
  *ax1+**
- 中置記法：中間順の木を辿る
  a(x+1)*



テキストには記載されていません

31

式の表現(続き)

【演習問題5】

中置記法で記述した下記の式を、ポーランド記法ならび逆ポーランド記法で記述した式を求めよ。

$a*b*c*(d+e/f)$

32

計算機言語講義資料 第2回補足

ポーランド記法

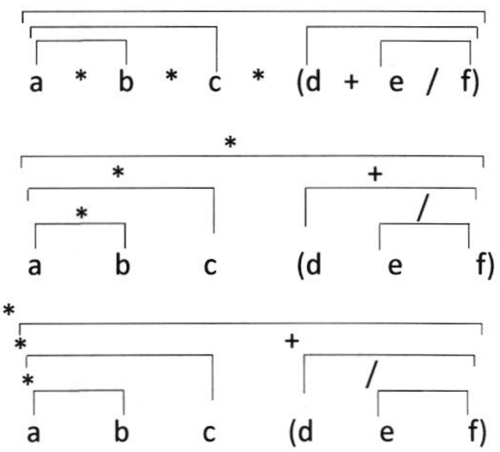
中置記法

*a*b*c*(d+e/f)*

演算子を括弧の上へ

全ての演算子を括弧の前方に寄せる

前方に演算子を下す



***abc+d/ef　左から順に

第2回配布資料 P28の修正

1

ポーランド記法と逆ポーランド記法

中置記法　式(1)　*a*b*c*(d+e/f)*

中置記法で記述した式 (1) の式木i

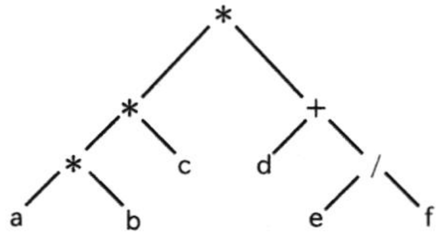


図1　式 (1) の式木

- ポーランド記法による式

  $* * * \, abc + d/ef$

- 逆ポーランド記法による式

  $ab * c * def / + *$

2

逆ポーランド記法で表記した式の計算
スタックによる計算方法

逆ポーランド記法で記述した数式を左から順番に処理

初期条件：スタックは空

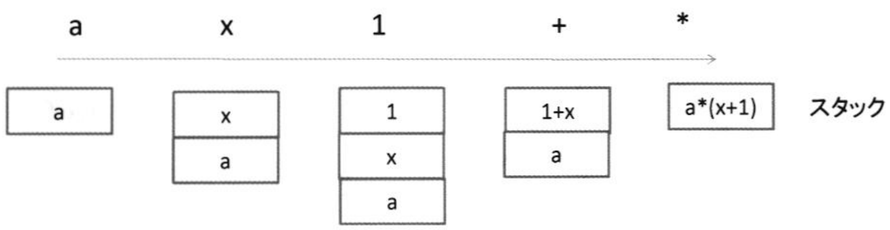
- 数値ならば、スタックに プッシュ
- 演算子ならば、
  - スタックの１番上の数値を ポップ(その数値をXとする)
  - スタックの１番上の数値をYとして、YとXを演算
  - 計算結果をYに代入
- 数式がなくなったとき、スタックには唯一の数値が残る
  - それが数式の計算結果

3

逆ポーランド記法　スタックを用いた計算例

a x 1+*

a * (x + 1)の　逆ポーランド記法



4