データ構造とプログラミング技法
（第１１回）

演算子優先構文解析

木パターンマッチング

同値類別

言語と文法

- 言語：あるルールを満足する記号の集合。有限集合とは限らない。

- 文法：言語を規定するためのルール（句構造文法）：

  非終端記号V：名詞、名詞句、動詞、動詞句など

  終端記号Σ：pen, this, a, is,

  出発記号S: （∈V）

  生成規則P:　名詞句::=冠詞　名詞

  名詞::=pen

  冠詞::=a

算術式を規定する文法の例

V ＝ {S, 生成規則集合 P において〈 〉で囲んだもの}
Σ ＝ {＋, －, ＊, ／, A, … , Z}
P ＝ {1) 〈算術式〉 ::＝〈項〉 | 〈算術式〉〈加減演算子〉〈項〉
2) 〈項〉 ::＝〈因子〉 | 〈項〉〈乗除演算子〉〈因子〉
3) 〈因子〉 ::＝〈変数〉 | (〈算術式〉)
4) 〈加減演算子〉 ::＝ ＋ | －
5) 〈乗除演算子〉 ::＝ ＊ | ／
6) 〈変数〉 ::＝ A | … | Z
}

S ≡〈算術式〉

A ＋ B ＊ C

導出木

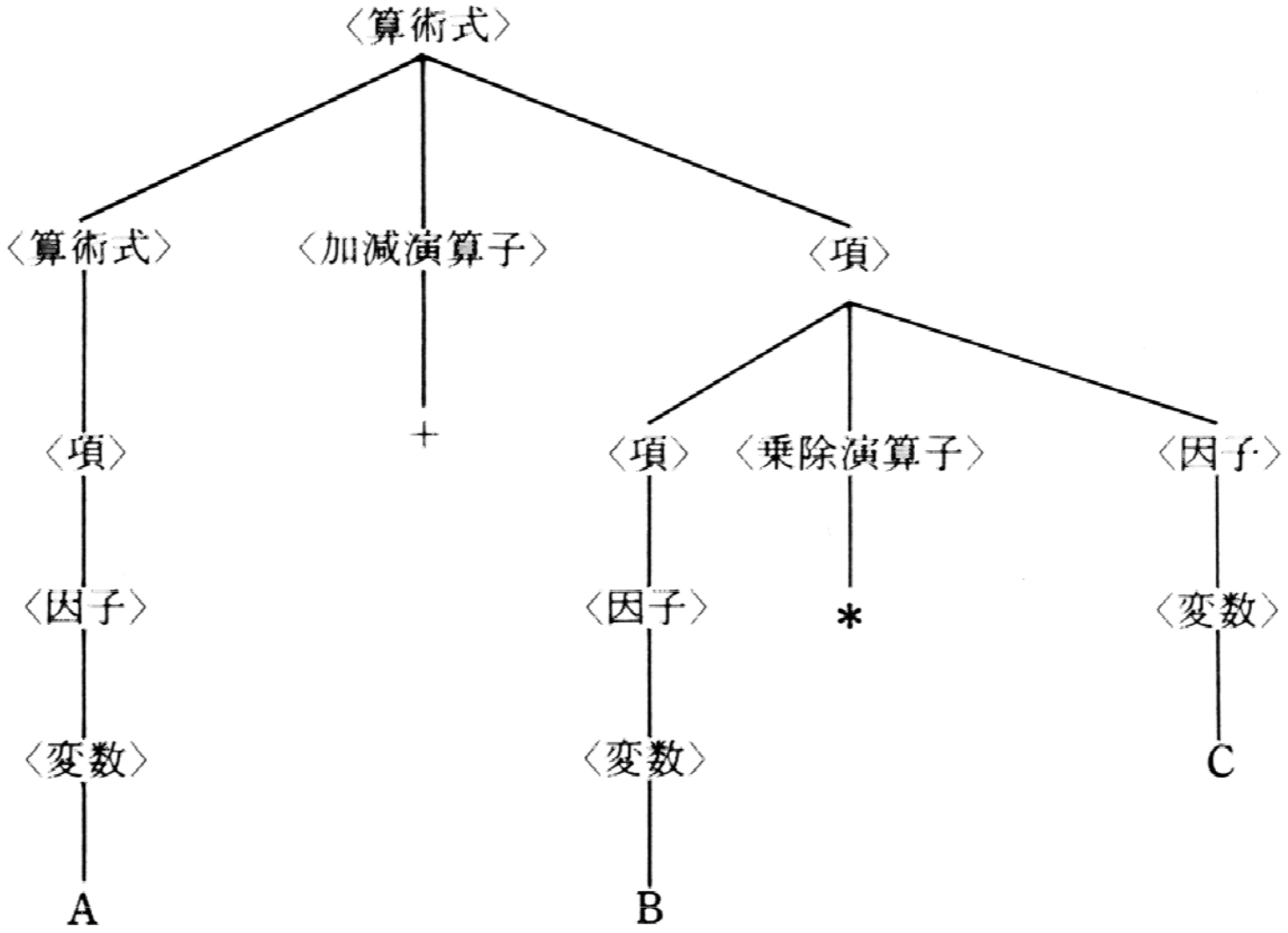


図 4・28　文法 G による算術式 'A + B * C' の導出木

構文木(演算子木)

終端記号間の関係を、演算子とその引数という形でまとめた木構造。非終端記号は表れない。導出木を簡略化して、得る方法と、直接構文木を得る方法の二つがある。

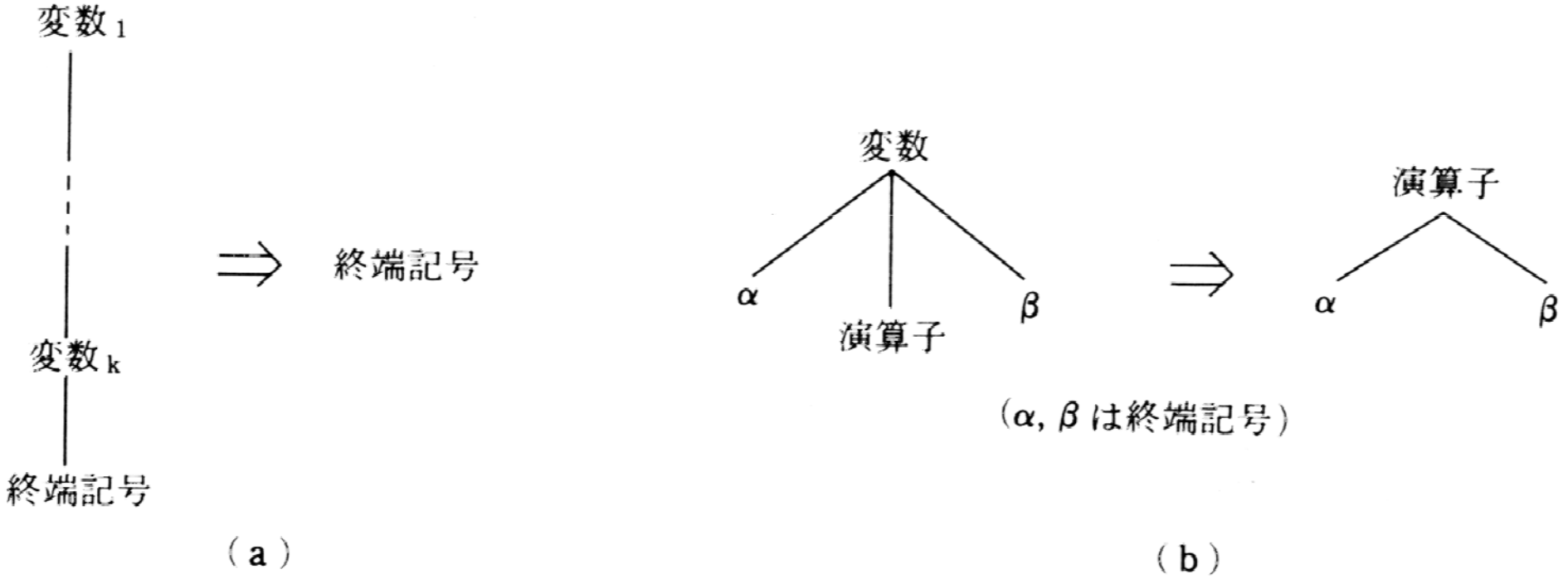


図 4・29　構文木のための簡単化規則

A+B*Cの構文木の例

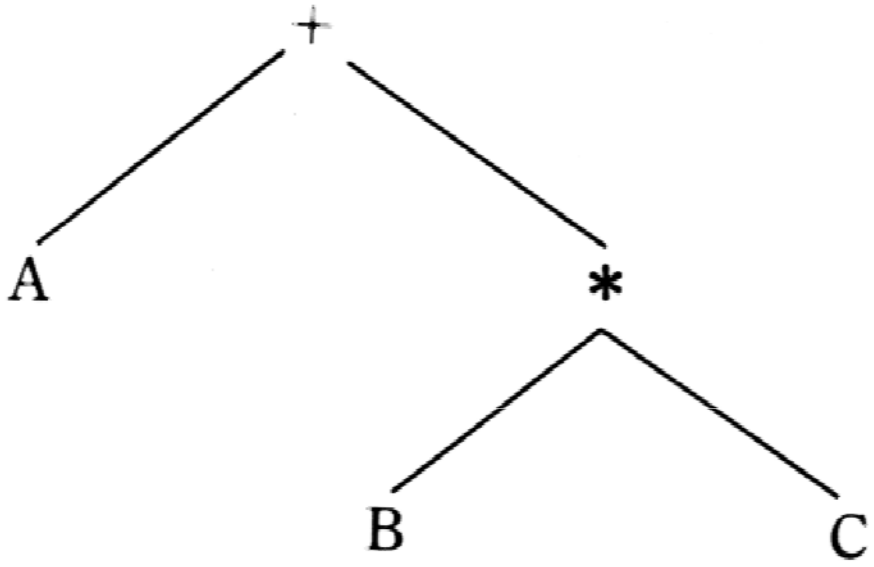


図 4・30　算術式 ‘A ＋ B ＊ C’ の構文木

最左導出、最右導出

導出：出発記号からスタートして、非終端記号を生成規則の左辺→右辺と辿りながら、置き換えていき、最終的に終端記号列を得る過程。

最左導出：未置換の非終端記号のうち、左にあるものから順に置きかえる方法。

生成規則の例：

<算術式>::=<項>|<項>+<算術式>

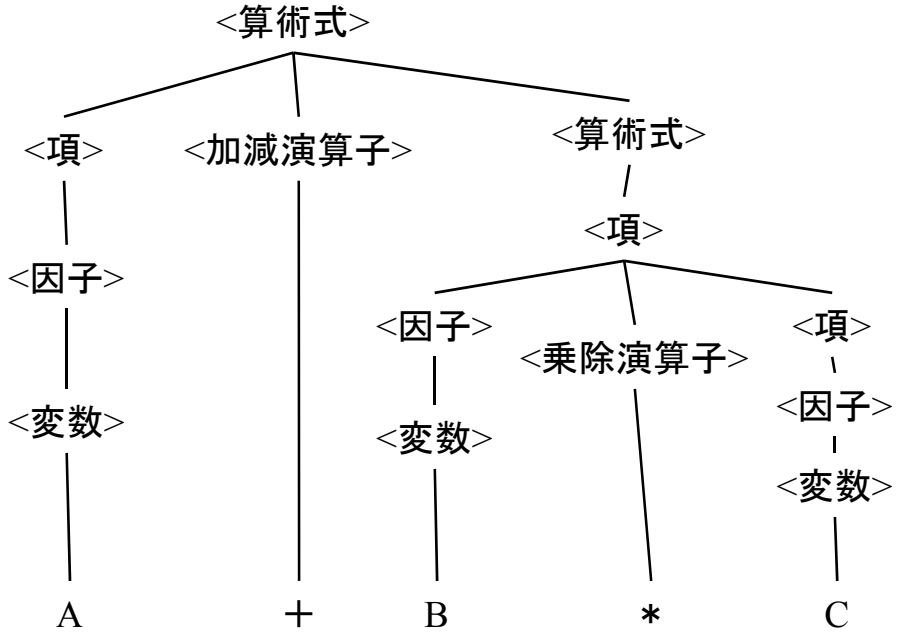
最右導出：未置換の非終端記号のうち、右にあるものから順に置きかえる方法。

生成規則の例：

<算術式>::=<項>|<算術式>+<項>

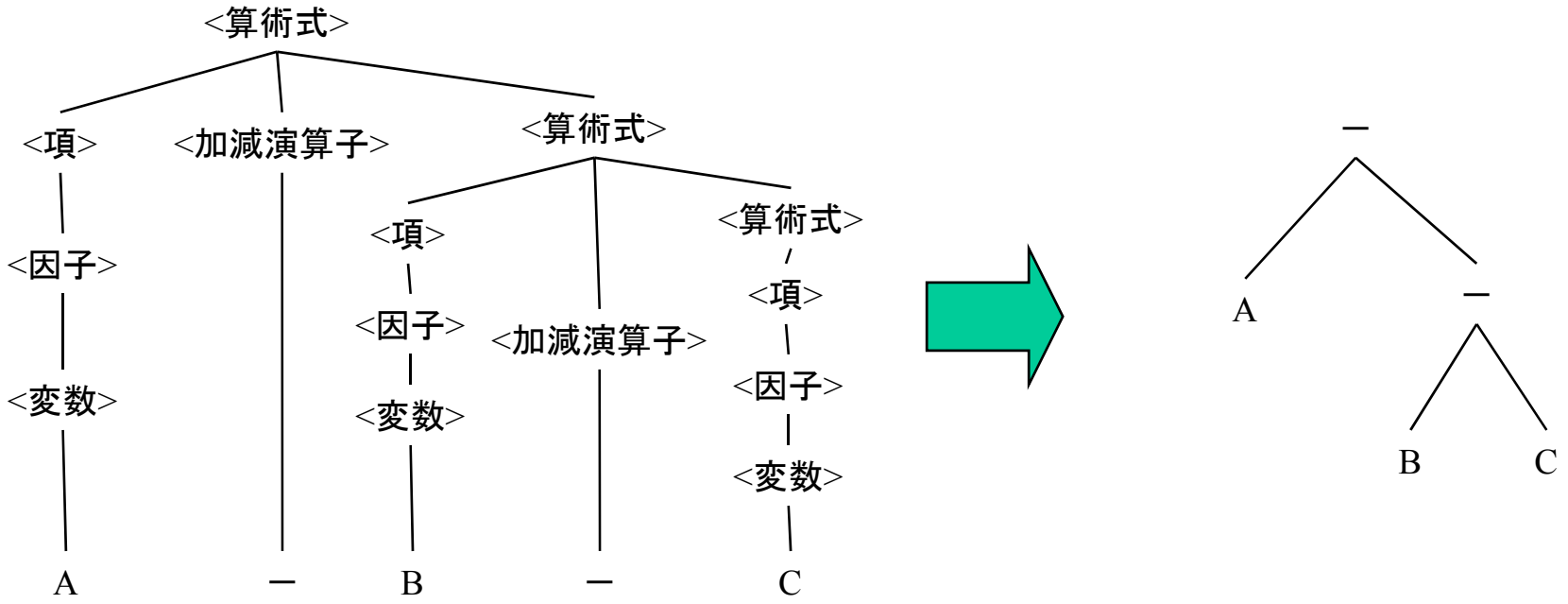
最左導出の例

<算術式>::=<項>|<項><加減演算子><算術式>
<項>::=<因子>|<因子><乗除演算子><項>
<因子>::=<変数>|(<算術式>)
<加減演算子>::=＋|－
<乗除演算子>::=＊|／
<変数>::=A |…|Z



最左導出がうまくいかない例

<算術式>::=<項>|<項><加減演算子><算術式>
<項>::=<因子>|<因子><乗除演算子><項>
<因子>::=<変数>|(<算術式>)
<加減演算子>::=＋|－
<乗除演算子>::=＊|／
<変数>::=A |…|Z



演算子優先構文解析

演算子間に優先順位を設け、生成規則の右辺から左辺に辿ることができる部分を見つける構文解析法。

- 導出木ではなく、直接構文木が得られる。
- 明示的な生成規則を持たない。
- 最右導出が実現できる。
- 木構造を下から上に作る。（上昇型構文解析）
- 2つのスタックを用いた単純な処理

演算子順位表

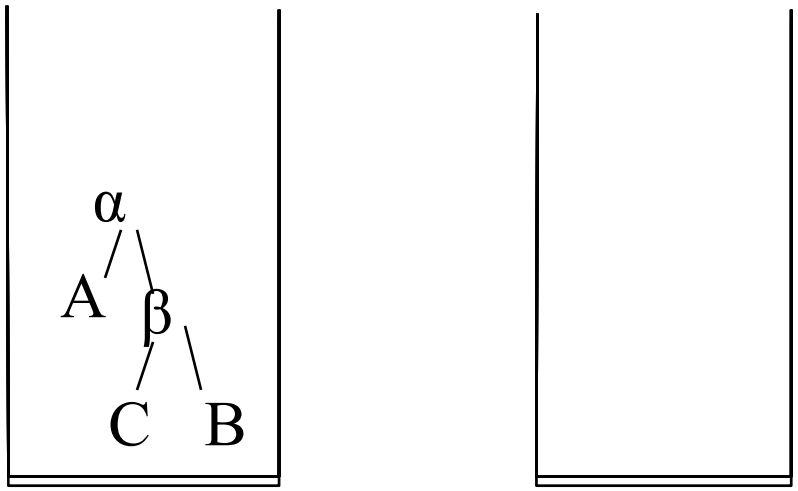
表 4・1　演算子順位表（α：左の演算子，β：右の演算子）

α \ β	+, −	*, ／	(	)
+, −	$\gtrdot$	$\lessdot$	$\lessdot$	$\gtrdot$
*, ／	$\gtrdot$	$\gtrdot$	$\lessdot$	$\gtrdot$
(	$\lessdot$	$\lessdot$	$\lessdot$	$\doteq$
)	$\gtrdot$	$\gtrdot$		$\gtrdot$

$$A\alpha B\beta C$$

αの方がβよりも優先順位が高いとき、還元（生成規則を右辺から左辺に辿る操作）を行う。

演算子優先構文解析



演算子優先構文解析のアルゴリズム(a)

```
procedure operator-check (var S, O : stack; x : operator);
  [if (empty-stack(O)) or (table (top (O), x) = '⋖') then Q ⇦ x
   else if (top(O) = '(') then delete-top(O)
        else [make-tree(S, O); operator-check(S, O, x)]
  ];
procedure make-tree(var S, O : stack);
    [r ⇦ O; β ⇦ S; α ⇦ S;
     create-tree(r, α, β);
     S ⇦ pointer(r)];
```

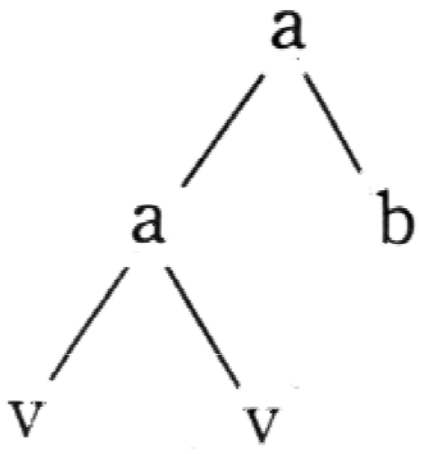
(a) 構文解析のための副手続き

演算子優先構文解析のアルゴリズム(b)

```
create-stack(S) ;
create-stack(O) ;
get-token(x) ;
while (x ≠ null) do
  [if (x = 変数) then S ⇦ x
   else operator-check(S, O, x) ;
   get-token(x)
  ].
```

(b) 構文解析の主手続き (部分)

木パターン照合問題とは



（a） p_1

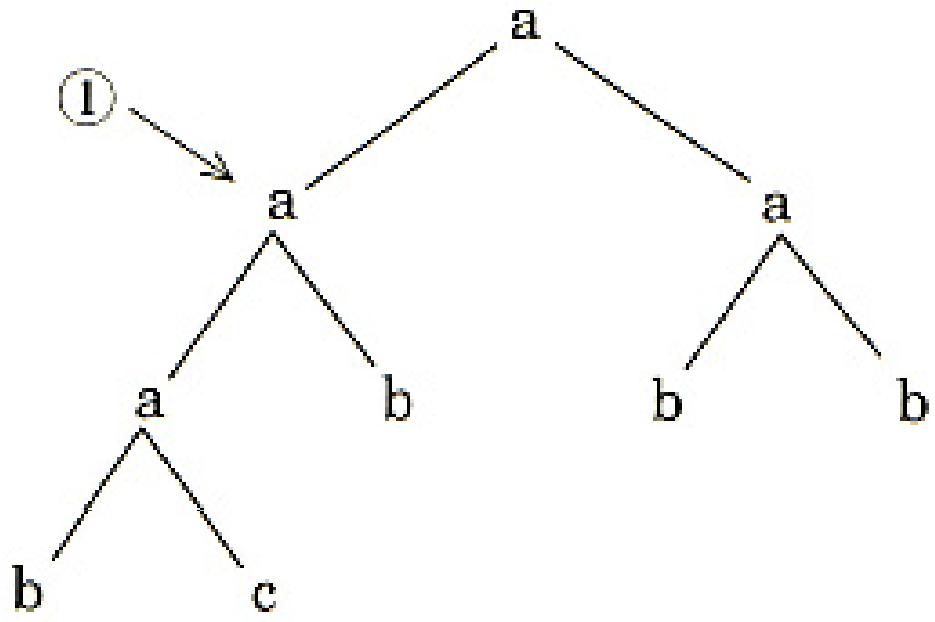


図 4·17　Σ−木の例

木パターン照合の方法

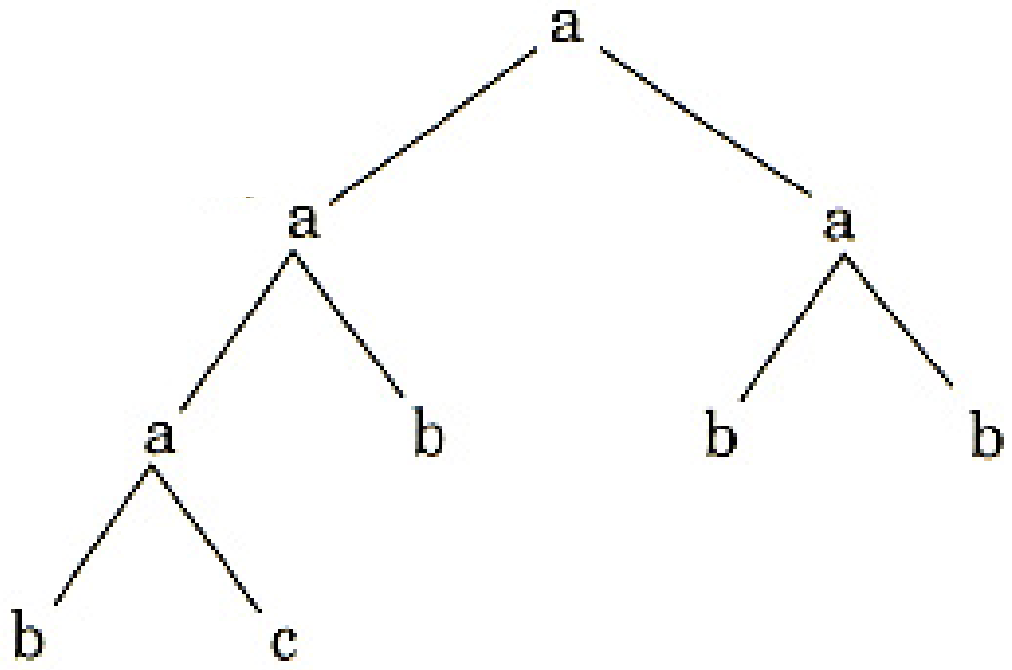
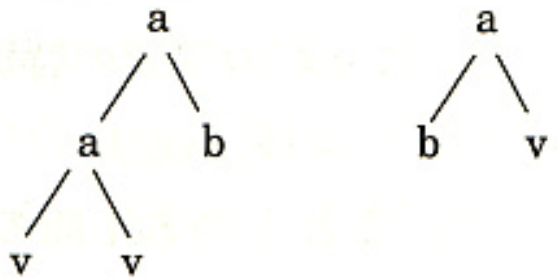


図 4·17　Σ−木の例

$p_1 = a(a(v, v), b)$,　　$p_2 = a(b, v)$

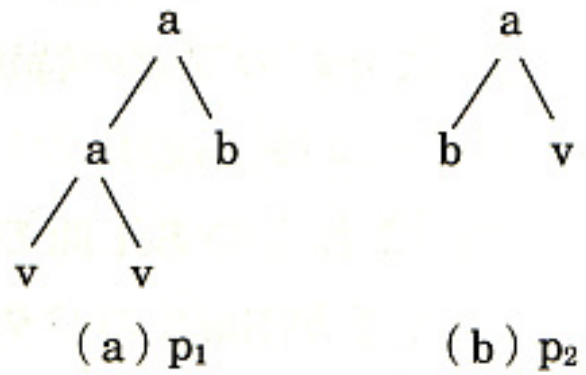


(a) p_1　　(b) p_2

図 4·18　Σ∪{v}−木の例

木パターン照合の準備

$p_1 = a(a(v, v), b)$,　　$p_2 = a(b, v)$



(a) p_1　　(b) p_2

図 4·18　ΣU {v} -木の例

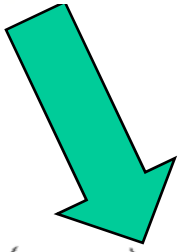
Set 1 = {v}

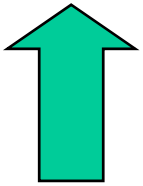
Set 2 = {b, v}

Set 3 = {a(v, v), v}

Set 4 = {a(b, v), a(v, v), v}

Set 5 = {a(a(v, v), b), a(v, v), v}





{v, b, a(v, v), a(b, v), a(a(v, v), b)}

木パターン照合の準備（続き）

表 4.1　節 a の照合表

第1子＼第2子		Set 1	2	3	4	5
Set	1	3	3	3	3	3
	2	4	4	4	4	4
	3	3	5	3	3	3
	4	3	5	3	3	3
	5	3	5	3	3	3

木パターン照合

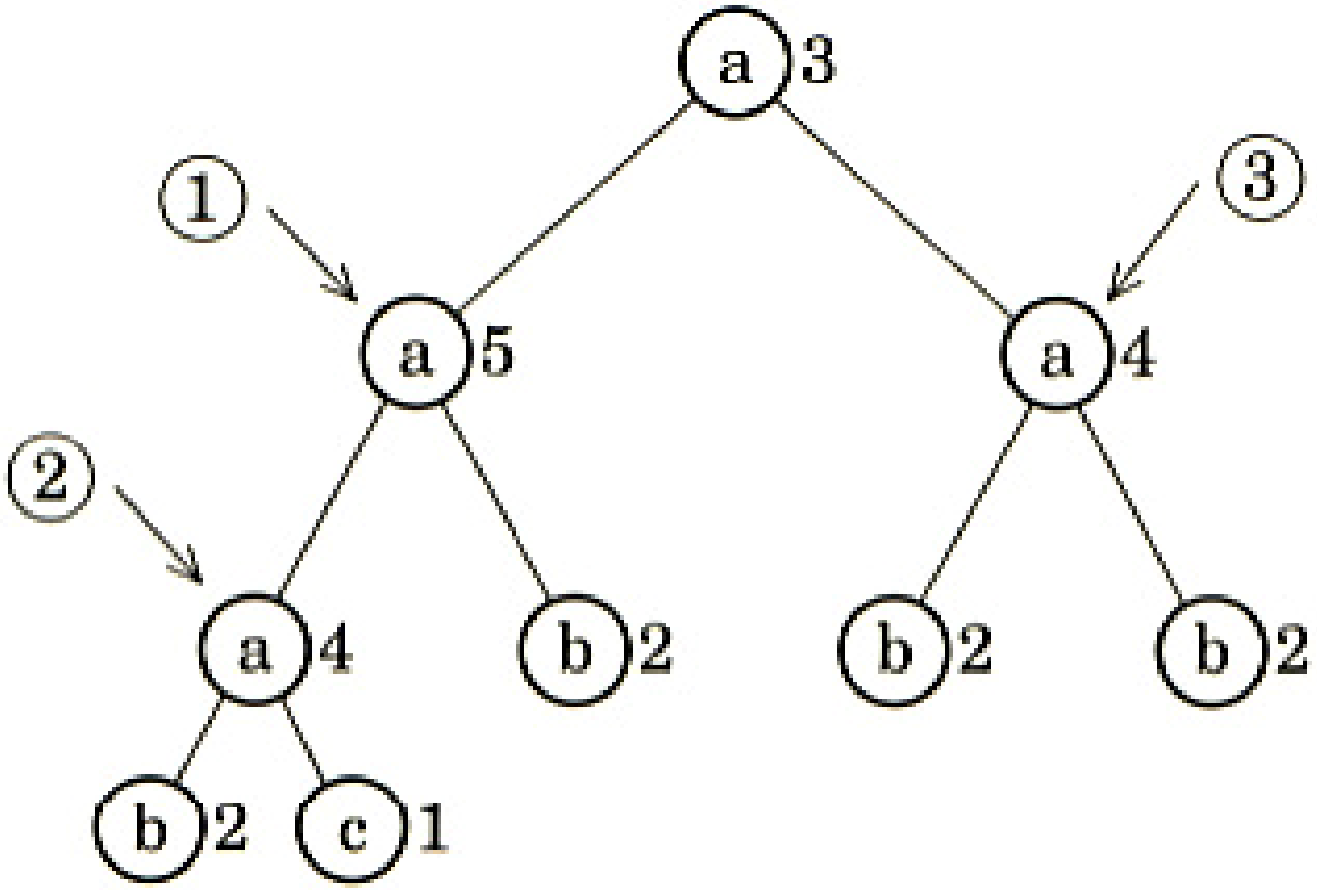


図 4·19　Set 番号の割当て後の Σ-木

同値類への分類

S={a,b,c,d,e,f,g}

(e,g), (d,f), (b,d), (c,e)

```
void group(int x, int y)
{   while (parent[x] != NIL)
        x = parent[x];
    while (parent[y] != NIL)
        y = parent[y];
    if (x != y)
        parent[y] = x;
}
```

図 4·20　x と y を同じグループにする同値類別プログラム
(parent [MAX] は NIL で初期化されているものとする)

同値類への分類結果

S={a,b,c,d,e,f,g}

(a,b),(e,g), (d,f), (b,d), (c,e)

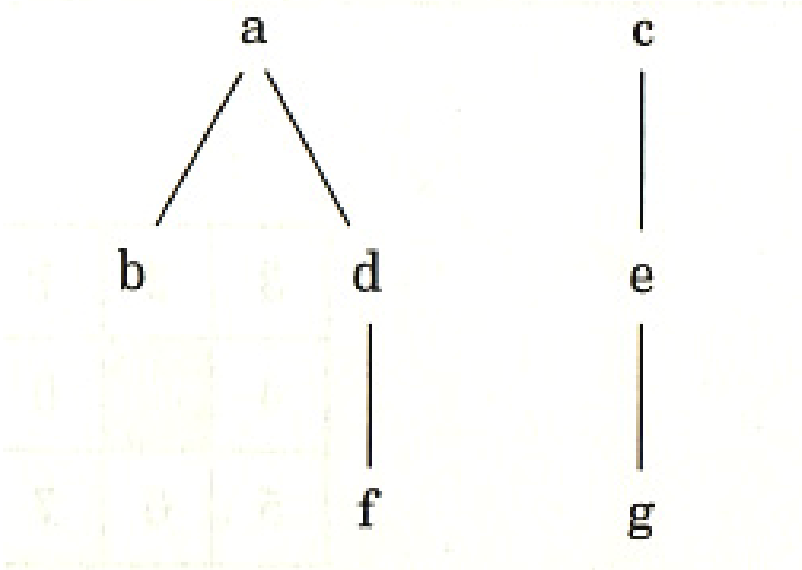


図 4･21　グループ化の結果