ソフトウェア基礎論　16－3
プログラミング言語　4－3
（１９８６．２．７）

属性文法における最適な属性評価順序の決定手法

西野　哲朗

日本アイ・ビー・エム株式会社　サイエンス・インスティチュート

１．　はじめに

従来、属性文法の理論において、訪問系列（visit sequence）の概念を用いた属性評価戦略が研究されてきた［１，５］。　それらの研究は、属性評価時に要求される導出木のノードに対する訪問回数によって、属性文法のクラスを特徴付けるものであった。

そうした属性文法のクラスとしては、次のRiisによる純粋 k-訪問属性文法（pure k-visit attribute grammar［５］）が最も一般的である。：『ｋを正整数とするとき、属性文法 G は、その任意の導出木 t に付随したすべての属性を、 t の各ノードを高々 k 回訪問することにより評価できるときに、純粋 k- 訪問であるという。』

多くの場合、このような属性文法のクラスに関する決定問題は、高いオーダーの時間計算量を要求する。　例えば次のような決定問題が考えられる。

純粋 k-訪問問題：『属性文法 G と任意の正整数 k が与えられたときに、 G が純粋 k-訪問であるか否かを決定せよ。』

この純粋 k-訪問問題は、本質的に指数時間計算量を要求する［４］。

本稿では、従来のこうした研究とは少し視点を変えて、次のような問題について考察する。：『属性文法 G とその導出木 t がひとつ与えられたときに、 t の各ノードに対する訪問回数の総和を最小にするような、t に対する属性評価順序を決定せよ。』

属性文法のクラスを決定することは重要な問題であるが、実際の属性評価という観点からすれば、個々の導出木に対して、どのような属性評価戦略をとればよいかを決定することに興味がある。

本稿で述べるアルゴリズムでは、属性文法 G が与えられると、まず最初に、ある前処理を行う。その後、ひとつの入力文 v（v は、 G の基礎文脈自由文法により生成される。）が与えられると、その入力文の構文解析が完了するのとほぼ同時に、得られた解析木に対する最適な属性評価順序を与える。（図１．参照。）

前処理は、上述の純粋 k-訪問問題の解法と本質的に同値なことを行うため、入力された属性文法のサイズの指数関数のステップ数を必要とする。　しかし、その後の処理は、入力文に対する構文解析アルゴリズムとほぼ同じステップ数しか必要としないので、入力文の長さの３乗のオーダーの時間計算量で行なえる。　したがって、１度前処理を行ってしまえば、その後は任意の入力文に対して、効率良く最適な属性評価順序を決定することができる。（図１．参照。）

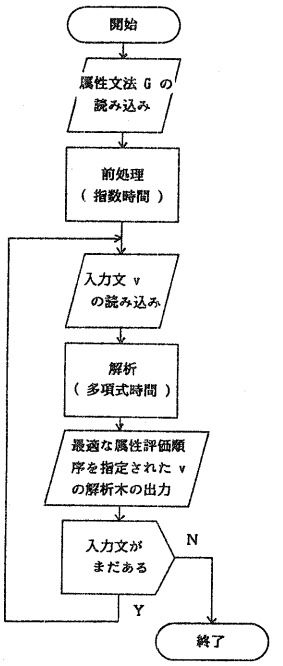

図１．　本稿で述べるアルゴリズムの枠組

２．　用語の定義

本節では、必要となる概念及び用語の定義を、具体例とともに述べる。

定義 2.1.　属性文法 G は、以下の３つの条件を満たす３つ組 $< G_u, A, F >$ である。

(1)　$G_u = (N, T, P, Z)$ は G の基礎文脈自由文法と呼ばれる。　ここに、N, T, P, Z はそれぞれ G_u の、非終端記号全体の集合、終端記号全体の集合、プロダクション全体の集合及び、開始記号である。 P のプロダクション p は　$p : X_0 \rightarrow w_0 X_1 w_1 X_2 \ldots X_n w_n$　と表される。　ここに、$n \geq 0$, $X_j \in N$, $w_j \in T^*$ $(0 \leq j \leq n)$ である。　通常の意味で G_u は既約であると仮定する。

(2) G_u の各非終端記号 X に対し、互いに素な2つの有限集合、継承属性の集合 I(X) と合成属性の集合 S(X) が付随している。 X の属性全体の集合を
$A(X) = I(X) \cup S(X)$ で表す。 $A = \bigcup_{X \in N} A(X)$ を G の属性集合という。 以下のことを仮定する。
$A(X) \neq \phi$, $I(Z) = \phi$, $|S(Z)| = 1$. ここに、$|S|$ は、集合 S の要素の個数を表す。 X の属性 a を a(X) で表し、a がとりうる値全体の集合を V(a) で表す。

(3) P の各プロダクション p に対し、
$S(X_0) \cup I(X_1) \cup \ldots \cup I(X_n)$ のすべての属性を定義する意味規則の集合 F_p が付随している。 属性 $a_0(X_{i_0})$ を定義する意味規則は、
$a_0(X_{i_0}) := f(a_1(X_{i_1}), \ldots , a_m(X_{i_m}))$,
$0 \leq i_j \leq n$, $0 \leq j \leq m$ という形をしている。
ここに、 f は、$V(a_1(X_{i_1})) \times \ldots \times V(a_m(X_{i_m}))$ から $V(a_0(X_{i_0}))$ の中への写像である。 このとき、$a_0(X_{i_0})$ は p において $a_j(X_{i_j})$ ($0 \leq j \leq m$) に依存するという。 集合 $F = \bigcup_{p \in P} F_p$ を G の意味規則集合という。 ■

定義 2.2. (1) $p : X_0 \rightarrow w_0 X_1 w_1 X_2 \ldots X_n w_n$ を属性文法 G のプロダクションとする。 p のプロダクション・グラフ PG(p) とは、p に現れる非終端記号 X_j ($0 \leq j \leq n$) のすべての属性に対応したノードをもち、p において属性 b が属性 a に依存するとき、かつそのときに限り属性 a から属性 b に向かう辺をもつ有向グラフである。

(2) t を属性文法 G の導出木とする。 t の導出木グラフ DTG(t) とは、t において使われた、すべてのプロダクションのプロダクション・グラフを貼合せて得られる有向グラフである。 ■

属性文法 G は、その任意の導出木グラフがループを含まないときに、非循環であるという。

例 2.1. 本稿を通じて、次のような属性文法 $G_0 = \langle G_1, A, F \rangle$ を考える。
G_0 の基礎文脈自由文法 $G_1 = (N, T, P, Z)$ に関しては、
N = { Z, A, B, C, D },
T = { a, b }
とする。 P の定義は後述。
各非終端に付随した属性の集合は以下の通りとする。
S(Z) = { s },
I(A) = { i_1 }, S(A) = { s_1 },
I(B) = { i_1, i_2 }, S(B) = { s_1, s_2 },
I(C) = { i_1, i_2 }, S(C) = { s_1, s_2 },
I(D) = { i_1, i_2, i_3, i_4 },
S(D) = { s_1, s_2, s_3, s_4 }.

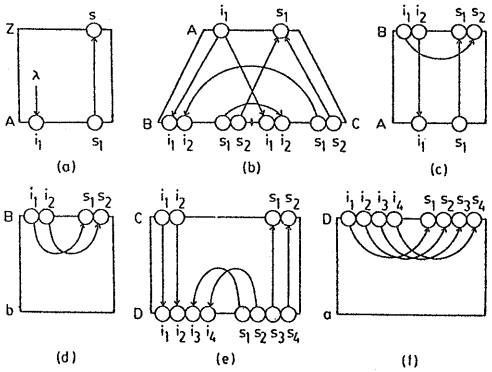

図2. 属性文法 G_0 のプロダクション・グラフ

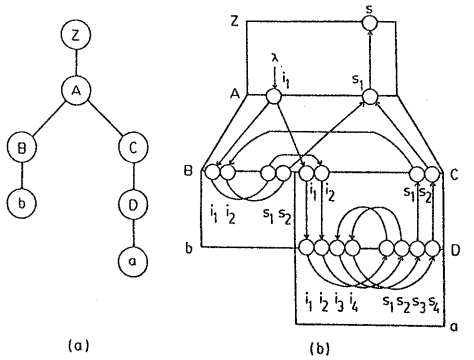

図3. 属性文法 G_0 の (a) 導出木 t 及び、(b) 対応する導出木グラフ

G_1 のプロダクションと、それらに付随した意味規則は、図2にプロダクション・グラフの形で示す。 図3 (a) の G_0 の導出木 t に対する導出木グラフを図3 (b) に示す。 ■

導出木は、そのすべての葉が終端記号でラベル付けられているときに、完全であるという。

定義 2.3. (1) X を属性文法 G の非終端記号とするとき、X の非終端グラフを次のように定義し、NG(X) と表す。: A(X) のすべての属性に対応するノードをもち、X を根のラベルとするある完全な導出木内における、A(X) の属性間のすべての依存関係にちょうど対応する辺をもつ有向グラフ.
NG(X) の各ノードにラベルを付けることにより得られるグラフを、X のラベル付き非終端グラフと呼び、l-NG(X) で表す。

(2) $p : X_0 \rightarrow w_0\ X_1\ w_1\ X_2 \ldots X_n\ w_n$ を属性文法 G のプロダクションとし、$1 \leq j \leq n$ に対し、$NG(X_j)$ を非終端 X_j の任意の非終端グラフとする。 そのとき、

(*) $PG(p)[\ NG(X_1), \ldots, NG(X_n)\ ]$

で、次のように PG(p) を拡張することにより得られる有向グラフを表す。 : 『 $NG(X_j)$ ($1 \leq j \leq n$) 内に属性 a から属性 b に向かう辺があるとき、かつそのときに限り、PG(p) 内に属性 $a(X_j)$ から属性 $b(X_j)$ に向かう辺を付け加える。』 (*) を p の拡張プロダクション・グラフと呼び、EPG(p) で表す。

EPG(p) の各ノードにラベルを付けることにより得られるグラフを、p のラベル付き拡張プロダクション・グラフと呼び、l-EPG(p) で表す。 ■

例 2.2. 例 2.1. の属性文法 G_0 の非終端記号 B の非終端グラフの例を図4に示す。 また、G_0の拡張プロダクション・グラフを図5に、ラベル付き拡張プロダクション・グラフを図6にそれぞれ示す。 ■

定義 2.4. (1) $p : X_0 \rightarrow w_0\ X_1\ w_1\ X_2 \ldots X_n\ w_n$ を属性文法 G のプロダクションとし、$g = EPG(p)$ とする。 そのとき g の還元非終端グラフを次のように定義し、RNG(g) で表す。 : X_0 のすべての属性に対応するノードをもち、g 内に属性 $a(X_0)$ から属性 $b(X_0)$ に向かうパスがあるとき、かつそのときに限り、属性 a から属性 b に向かう辺をもつ有向グラフ。

また、$g = l\text{-}EPG(p)$ のとき、EPG(p) の還元非終端グラフで、その各ノードがすべて、対応する g 内の X_0 の属性に対するノードと同じラベルをもつものを、g のラベル付き還元非終端グラフと呼び、l-RNG(g) で表す。

(2) $p : X_0 \rightarrow w_0\ X_1\ w_1\ X_2 \ldots X_n\ w_n$,
$q : Y_0 \rightarrow v_0\ Y_1\ v_1\ Y_2 \ldots Y_m\ v_m$
をそれぞれ属性文法 G のプロダクションとし、
$g_p = l\text{-}EPG(p)$, $g_q = l\text{-}EPG(q)$ とする。
ある j ($1 \leq j \leq n$) に関して $Y_0 = X_j$ かつ、
$l\text{-}NG(X_j) = l\text{-}RNG(g_q)$ のとき、かつそのときに限り、g_q は X_j において g_p に接続可能であるという。 ただし、$l\text{-}NG(X_j)$ は、EPG(p) を構成するときに、PG(p) に X_j において貼り付けられた非終端グラフ ($NG(X_j)$) に、g_p 内の対応するラベルを付けた、ラベル付き非終端グラフとする。 ■

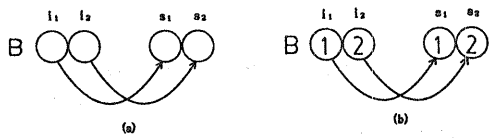

図4. 属性文法 G_0 の非終端記号 B の (a) 非終端グラフ及び、(b) ラベル付き非終端グラフ

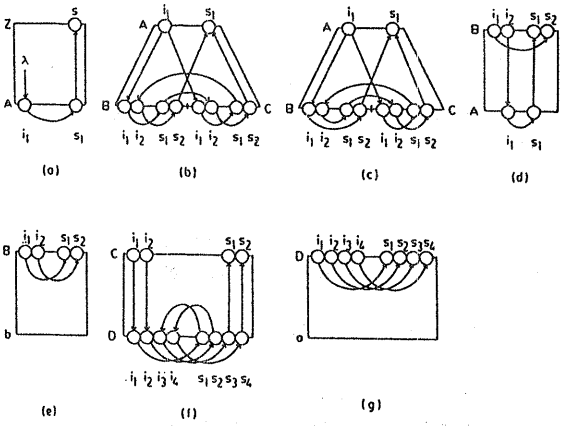

図5. 属性文法 G_0 の拡張プロダクション・グラフ

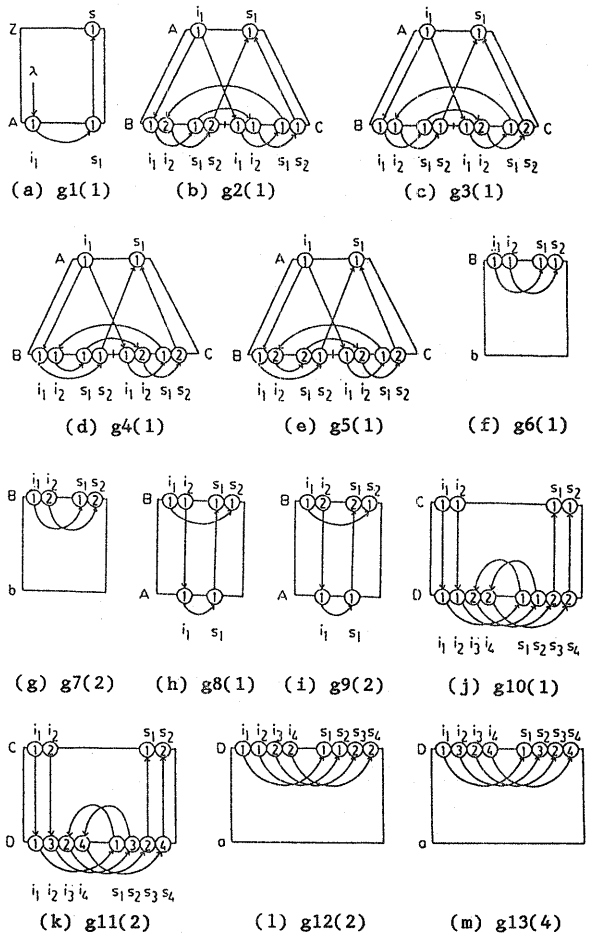

図6. 属性文法 G_0 のラベル付き拡張プロダクション・グラフ

例 2.3.　1．図７（a）に、図５（d）の拡張プロダクション・グラフに対する還元非終端グラフを示す。また、図６の g9(2) に対するラベル付き還元非終端グラフを示す。

2．図６において、ラベル付き拡張プロダクション・グラフ g8(1) は、B においてラベル付き拡張プロダクション・グラフ g4(1) に接続可能である。 ■

３．　属性評価順序の決定問題

t を、図３（a)に示したような、例 2.1. の属性文法 G_0 の導出木とする。　次のような問題を考えてみよう。

『 t の各ノードに対する訪問回数の総和を最小にして、t に付随したすべての属性を評価する属性評価順序を決定せよ。』

t の導出木グラフは、図３（b）のようになる。t に付随したすべての属性は、図８に示した２通りの順序で評価することができる。　この２通りの場合それぞれについて、t の各ノードに対する訪問回数の総和を求めてみると、図８（a）の順序では７回、(b) の順序では９回となる。　したがって、上の問題に対する解答は、図８（a）に示される属性評価順序となる。

次節で、属性文法 G とその導出木 t が与えられたときに、上記の決定問題を解くアルゴリズムを示す。

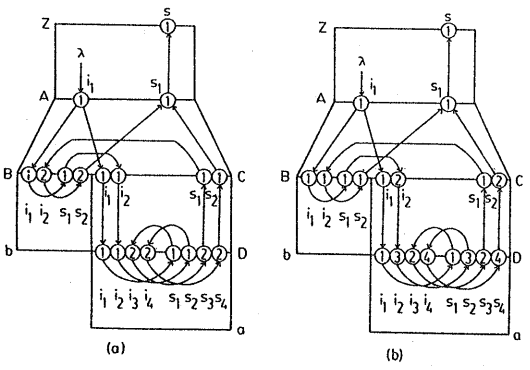

図８．　図３（a）の導出木 t に対する２通りの属性評価順序

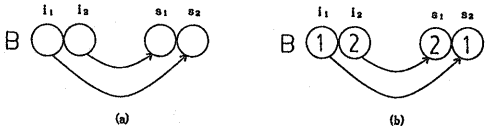

図７．　還元非終端グラフ及び、ラベル付き還元非終端グラフの例

４．　解法アルゴリズムの内容

前述の通り、以下で述べる属性評価順序決定アルゴリズムは、前処理部と解析部に分かれている。最初に、前処理部の手続きを示す。　各ステップの具体的な手続きは複雑なので、付録にまとめて掲載した。　興味のある方は、対応する手続きを参照されたい。

アルゴリズム 4.1.
　　属性評価順序決定のための前処理

入力　　非循環属性文法 G

出力　　G の拡張プロダクション・グラフの接続規則

手法

１．　G のすべての拡張プロダクション・グラフを構成する。
（ 詳細は、付録の BUILD_EPG 参照。BUILD_NG 及び PASTE_REDUCE が ｻﾌﾞﾙｰﾁﾝ として使われる。）

２．　１．で構成された各拡張プロダクション・グラフのそれぞれのノードに、そのノードが表す属性が評価されるときの、導出木のノードに対する訪問番号（ visit number ）をラベル付ける。　ひとつの拡張プロダクション・グラフに対して何通りかの可能なラベリングが存在するときは、各ラベリングに対してひとつのラベル付き拡張プロダクション・グラフを作る。
（ 具体的には、G のひとつの拡張ﾌﾟﾛﾀﾞｸｼｮﾝ･ｸﾞﾗﾌ に対して、付録の非決定性手続き BUILD_ORDER が成功したときに得られるすべての ﾗﾍﾞﾘﾝｸﾞ関数による ﾗﾍﾞﾙ付き拡張ﾌﾟﾛﾀﾞｸｼｮﾝ･ｸﾞﾗﾌ の集合を構成する。）

３．　接続可能性を考慮して、２．で構成されたラベル付き拡張プロダクション・グラフの接続規則を構成する。
（ 詳細は、付録の BUILD_RULE 参照。）

手続きの第１ステップは、Knuth［３］による属性文法の非循環性チェック・アルゴリズムと本質的に同じ処理を行うので、G のサイズの指数時間を必要とする［２］。

プロダクション・グラフ内でのノードの訪問順序は、付録の手続き BUILD_ORDER の中で与えることができるが、複雑になるので本稿では省略した。

例 4.1.　アルゴリズム 4.1. による処理の内容を示すために、前節で定義した属性文法 G_0 に、このアルゴリズムを適用した場合について考察する。

１．　G_0 の拡張プロダクション・グラフを構成すると、図５のようになる。

２．　図５の拡張プロダクション・グラフから構成された、ラベル付き拡張プロダクション・グラフを図６に示す。　例えば、図５ (b) の拡張プロダクション・グラフに対しては、可能なラベリングが２通り存在するので、各ラベリングに対応して図６ (b), (c) のグラフが作られる。

３．　接続可能性を考慮すると、図６のラベル付き拡張プロダクション・グラフの接続規則は図９のようになる。$ は、接続規則の開始記号である。 ■

```
$ -> g1(1)
g1(1) -> g2(1) | g3(1) | g4(1) | g5(1)
g2(1) -> g7(2) g10(1)
g3(1) -> g6(1) g11(2)
g4(1) -> g8(1) g11(2)
g5(1) -> g9(2) g11(2)
g6(1) -> b
g7(2) -> b
g8(1) -> g2(1) | g3(1) | g4(1) | g5(1)
g9(2) -> g2(1) | g3(1) | g4(1) | g5(1)
g10(1) -> g12(2)
g11(2) -> g13(4)
g12(2) -> a
g13(4) -> a
```

図９．　図６．のラベル付き拡張プロダクション・グラフの接続規則

g をプロダクション $X_0 \rightarrow w_0\ X_1\ w_1\ \ldots\ X_n\ w_n$ に対する拡張プロダクション・グラフとする。　図９の規則において、非終端の中に含まれるカッコで囲まれた数は、g における X_0 への訪問回数になっている。（この値は、付録の BUILD_RULE において関数 f により評価される。）　例えば、g9(2) という非終端は、そのグラフにおいて、G_0 の非終端 B への訪問回数が２回であることを表している。構成の仕方から、ここで得られる接続規則は、文脈自由である。

次に解析部の手続きを示す。　解析部では、アルゴリズム 4.1. で得られた G のラベル付き拡張プロダクション・グラフの接続規則を用いて、入力文の構文解析を行う。

アルゴリズム 4.2.
属性評価順序決定のための解析

入力
ａ．　アルゴリズム 4.1. で得られた G のラベル付き拡張プロダクション・グラフの接続規則 R
ｂ．　入力文 v（G の基礎文脈自由文法によって生成される文）

出力
最適な属性評価順序が指定された v の解析木

手法
１．　接続規則 R を用いて入力文 v を構文解析し、v の可能なすべての解析木の集合を作る。

２．　１．で得られた v のすべての解析木の中から、各ノードに対する訪問回数の総和が最小のものを選び出す。　すなわち、それらの解析木のうちで、非終端に含まれるカッコで囲まれた数の総和が最小のものを選び出せばよい。

アルゴリズム 4.2. の第１ステップは、効率の良い構文解析アルゴリズム（CKY-アルゴリズム等）を用いれば、入力文の長さの３乗のオーダーのステップ数で実行できる。

例4.2. アルゴリズム 4.2. の処理の内容を、再び例2.1. の属性文法 G_0 の場合について考察する。 入力文としては、v = ba を考える。

1. 入力文 v を図9の接続規則によって構文解析すると、図10のような2つの解析木を得る。

2. 図10の2つの解析木のうちから、最適な属性評価順序を表すものを選び出す。 すなわち、図10のそれぞれの解析木に対して、そのすべてのノードに対する訪問回数の和を計算し、その総和が小さい方をとればよい。 訪問回数の総和を求めるには、それぞれの解析木の非終端に含まれるカッコで囲まれた数の和を求めればよい。 図10の解析木について上のような和を求めると、解析木 (a) については7、(b) については9を得るので、入力文 v に対しては、解析木 (a)で表される属性評価順序をとるのが最適であることがわかる。 その評価順序は、図8(a) の評価順序と一致している。 (図10 (a) の解析木は、図8 (a) の導出木グラフを表したものである。) ■

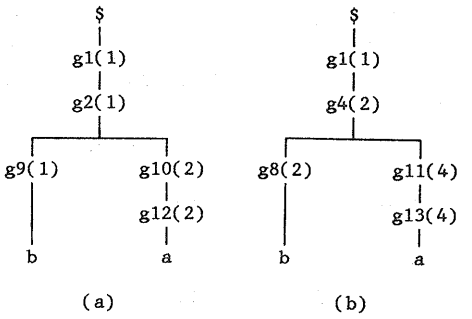

図10. 図9. の接続規則による入力文 v = ba の2通りの解析木

5. 最適性条件の変更

前節までは、入力文の解析木に対する、次のような条件Sに関し最適な属性評価順序を決定することを考えた。

条件S : 解析木の各ノードに対する訪問回数の総和を最小にする。

しかし、属性評価順序の最適性に関する条件は、他にも例えば次のようなものが考えられるであろう。

条件M : 解析木のひとつのノードに対する最大訪問回数を最小にする。

条件A : 解析木の各ノードに対する平均訪問回数を最小にする。

条件M及び、Aに関し最適な属性評価順序を決定したい場合にも、前述のアルゴリズムは、アルゴリズム 4.2. の第2ステップを変更するだけで、全く同様に適用できる。

条件Mに関しては、アルゴリズム 4.2. の第2ステップを次のように変更すればよい。

2'. 1. で得られた v のすべての解析木の中から、条件Mに関し最適なものを選び出す。 すなわち、それらの解析木のうちで、非終端に含まれるカッコで囲まれた数の最大値が最も小さいものを選び出せばよい。

また、条件Aに関しては、次のように変更すればよい。

2". 1. で得られた v のすべての解析木の中から、条件Aに関し最適なものを選び出す。 すなわち、それらの解析木のうちで、非終端に含まれるカッコで囲まれた数の総和を、その解析木のノード数で割った値が最小のものを選び出せばよい。

6. おわりに

本稿では、属性文法 G とその解析木 t が与えられたときに、指定された訪問回数に関する条件に関して最適な、 t に対する属性評価順序を決定する手法について述べた。 そこでは、最適性の条件として特に3つの条件を取り上げたが、その他の訪問回数に関する条件についても、前述のアルゴリズムに対する若干の修正で、対応できるものと思われる。

文献

1. J. Engelfriet and G. Filé, Simple Multi-Visit Attribute Grammars, JCSS 24 (1982), 283-314.
2. M. Jazayeri, W. F. Ogden and W. C. Rounds, The Intrinsically Exponential Complexity of the Circularity Problem for Attribute Grammars, CACM 18 (1975), 697-706.
3. D. E. Knuth, Semantics of Context-Free Languages, Math. Systems Theory 2 (1968), 127-145. Correction : Math Systems Theory 5 (1971), 95-96.
4. T. Nishino, The Intrinsically Exponential Comlexity of the k-visit Property Problem for Attribute Grammars (in preparation).
5. H. Riis and S. Skyum, k-visit Attribute Grammars, Math Systems Theory 15 (1981), 17-28.

付録

本文のアルゴリズム 4.1. の各ステップに対する詳細な手続きを、以下に示す。 以下では、g をグラフとするとき、g のノード全体の集合を V(g) で、辺全体の集合を E(g) で表すこととする。また a 以上 b 以下の自然数の集合を [a,b] で表す。

```
1     Procedure BUILD_NG ( G : noncircular AG,
                           { NG[X] | X ∈ N } ):
2  main: begin
3           for each X in N  do NG[X] := φ end;
4           ProdSet := { p | p : X0-> w0, X0∈N , w0∈T* };
5           repeat
6              convergence := true;
7              for each X in N do
8                 NEW(NG[X]) := false;
9              end;
10 inner:      for each p∈ ProdSet do
11                if n = 0 then
12                   if PG(p) ∉ NG[X0] then do   /* p : X0 -> w0 */
13                      NG[X0] := NG[X0]∪{PG(p)};
14                      NEW(NG[X0]) := true;
15                      convergence := false
16                   end
17                   else
18                      for each (g1,...,gn)∈ NG[X1]X...X NG[Xn] do
19                       /* Let p be the production of the form */
20                       /* X0-> w0X1w1... wn-1Xnwn( n ≥ 0 ).    */
21                         PASTE_REDUCE( PG(p),g1,...,gn,A(X0),g );
22                         if g ∉ NG[X0] then do
23                            NG[X0] := NG[X0]∪{g};
24                            NEW(NG[X0]) := true;
25                            convergence := false
26                         end
27                      end
28             end; inner
29             ProdSet := { p | p : X0-> w0X1w1... wn-1Xnwn such
30                          that for each j∈[1,n], NG[Xj] ≠ φ
31                          and for some j, NEW(NG[Xj])=true }
32          until convergence
33       end main
```

[注] 手続き BUILD_NG は、属性文法 G の各非終端記号 X∈N に対して、非終端グラフ NG(X) 全体の集合 NG[X] を構成し出力する。NEW は、集合 NG[X] が更新されたとき、かつそのときにかぎり真となる述語とする。

```
1     Procedure PASTE_REDUCE ( PG(p)       : production graph,
2                              gi,i [1,n] : nonterminal graph,
3                              A(X0) : set of all the attributes of X ,
4                              g     : nonterminal graph ):
5        begin
6           V(e) := V(PG(p));
7           E(e) := E(PG(p));
8           for each j∈[1,n] do
9              for each (a,b)∈E(gj) do
10                if (a(Xj),b(Xj)) ∉ E(PG(p)) then
11                   E(e) := E(e)∪{(a(Xj),b(Xj))}
12             end
13          end;
14          V(g) := A(X0);
15          for each oriented path from a(X0) to b(X0) in e do
16             E(g) := E(g)∪{(a,b)}
17          end
18       end
```

[注] p : X0 -> w0 X1 w1 ... Xn wn を属性文法 G のプロダクションとする。手続き PASTE_REDUCE は、最初にプロダクション・グラフ PG(p) と非終端グラフ gi = NG(Xi) (1 ≤ i ≤ n) から拡張プロダクション・グラフ e = PG(p)[g1, ... , gn] を構成し、さらにその還元非終端グラフ g = RNG(e) を構成して出力する。

```
1     Procedure BUILD_EPG ( G : noncircular AG ):
2        begin
3           BUILD_NG( G, { NG[X] | X ∈ N } );
4           for each p∈P do /* p : X0 -> w0 X1 w1 X2 ... Xn wn ( n ≥ 0 ) */
5              if n = 0
6                 then EPG[p] := {PG(p)} /* p : X0 -> w0 */
7                 else for each (g1,...,gn)∈NG[X1]  ...  NG[Xn] do
8                    EPG[p] := EPG[p]∪{ PASTE(PG(p),g1,...,gn) }
9                 end
10          end
11       end
```

[注] 手続き BUILD_EPG が、本文の アルゴリズム 4.1. の第1ステップに対応する主手続きである。ここでは上記の手続き BUILD_NG から出力された非終端グラフの集合 NG[X], X∈N から、属性文法 G の各プロダクション p∈P に対して、拡張プロダクション・グラフ全体の集合 EPG[p] を構成する。PASTE(PG(p), g1, ... , gn) = PG(p)[g1, ... , gn] とする。

```
1     Procedure BUILD_ORDER ( g : extended production graph,
2                             f_g : labeling function );
          /* f_g is defined only on A(X_0). */
3        begin
4           for each i∈[0,n] do
5              C_i := 1 /* initialization of visit counters */
6           end;
7           Args := { i(X_0)∈I(X_0) | f_g(i(X_0)) = 1 };
8           repeat
9              j := CHOICE[0,n]; /* nondeterministic step */
10             if j = 0 then do
11                SNS := { s(X_0)∈S(X_0) | ∃n∈Args s.t.
                          (n,s(X_0))∈E(g) };
12                if SNS = φ
13                   then stop  /* fail */
14                   else do
15                      if SNS ≠ { s(X_0) | f_g(s(X_0)) = C_0}
16                         then stop  /* fail */
17                         else Args = Args ∪ {i(X_0)∈I(X_0) | f_g(i(X_0)) = C_0+ 1 }
18                   end
19                end
20             else do
21                INS := Undef[I(X_j),Args];
22                if INS = φ
23                   then stop  /* fail */
24                   else do
25                      for each i(X_j)∈INS do
26                         f_g(i(X_j)) = C_j
27                      end;
28                      SNS := Undef[S(X_j),INS];
29                      if SNS ≠ φ
30                         then do
31                            for each s(X_j)∈SNS do
32                               f_g(s(X_j)) = C_j
33                            end;
34                            C_j = C_j + 1;
35                            Args := Args ∪ SNS
36                         end
37                   end
38             end
39          until f_g is defined on every node in g
40       end
```

[注] gをﾌﾟﾛﾀﾞｸｼｮﾝ p : X_0 -> w_0 X_1 w_1 ... X_n w_n に対する拡張ﾌﾟﾛﾀﾞｸｼｮﾝ･ｸﾞﾗﾌとする。手続きBUILD_ORDER は、$A(X_0)$ の属性に対応するﾉｰﾄﾞ上でのみ定義されたﾗﾍﾞﾘﾝｸﾞ関数 f_1 を、g 内における属性間の依存関係と矛盾を起こさないように、V(g) 上に拡張する非決定性手続きである。
Undef[A(X),Args] で、A(X) の属性のうち、まだ対応する g のﾉｰﾄﾞにﾗﾍﾞﾙが付いておらず、かつ Args の属性を用いて評価できるもの (定数が代入される場合も含める) 全体の集合を表す。ここでは、属性とその属性に対応する拡張ﾌﾟﾛﾀﾞｸｼｮﾝ･ｸﾞﾗﾌ内のﾉｰﾄﾞを同一視している。g 内の依存関係や、すでに与えられている $S(X_0)$ のﾉｰﾄﾞのﾗﾍﾞﾘﾝｸﾞと矛盾を起こすような訪問系列を選ぶと、手続きは失敗して停止する。属性文法 G のすべてのﾗﾍﾞﾙ付き拡張ﾌﾟﾛﾀﾞｸｼｮﾝ･ｸﾞﾗﾌを得るには、次のようにすればよい。: (1) Z -> w_0 X_1 w_1 ... X_n w_n の形をしたすべてのﾌﾟﾛﾀﾞｸｼｮﾝに対する拡張ﾌﾟﾛﾀﾞｸｼｮﾝ･ｸﾞﾗﾌ g と、その $S(X_0)$ のﾉｰﾄﾞ (仮定から1個しかない) に1をﾗﾍﾞﾙ付けるﾗﾍﾞﾘﾝｸﾞ関数に対して、BUILD_ORDER を実行し、手続きを成功に導くような、すべての g のﾗﾍﾞﾘﾝｸﾞ関数を構成する。(2) X_i ($1 \le i \le n$) を左辺にもつすべてのﾌﾟﾛﾀﾞｸｼｮﾝ (X_i -> v_0 Y_1 v_1 ... Y_m v_m の形とする) に対する拡張ﾌﾟﾛﾀﾞｸｼｮﾝ･ｸﾞﾗﾌと、(1)で得られた $A(X_i)$ のﾉｰﾄﾞに対するすべてのﾗﾍﾞﾘﾝｸﾞに対して BUILD_ORDER を実行し、手続きを成功に導くような、すべてのﾗﾍﾞﾘﾝｸﾞ関数を構成する。(3) Y_j ($1 \le j \le m$) を左辺にもつすべてのﾌﾟﾛﾀﾞｸｼｮﾝに対する拡張ﾌﾟﾛﾀﾞｸｼｮﾝ･ｸﾞﾗﾌと、(2)で得られた $A(Y_j)$ のﾉｰﾄﾞに対するすべてのﾗﾍﾞﾘﾝｸﾞに対してBUILD_ORDER を実行し、手続きを成功に導くような、すべてのﾗﾍﾞﾘﾝｸﾞ関数を構成する。(4) 以下同様のことを、いかなるﾌﾟﾛﾀﾞｸｼｮﾝの拡張ﾌﾟﾛﾀﾞｸｼｮﾝ･ｸﾞﾗﾌに対しても、新しいﾗﾍﾞﾘﾝｸﾞが得られなくなるまで繰り返す。

```
1     Procedure BUILD_RULE
2        begin
3           RULES := { $ -> g(f(g)) | g is an EPG(p) for some  p : Z -> α };
4           for each p∈P do
            /* Let p be X_0-> w_0X_1w_1... w_{n-1}X_nw_n */
5              for each (g_0,...,g_n)∈EPG[X_0] X ... X EPG[X_n] do
6                 if n = 0
7                    then RULES := RULES  { g_0(f(g_0)) -> w_0| g_0∈EPG[p] }
8                    else if all g_i,i∈[1,n] can paste together
                             with g_0 at X_i then
9                              RULES := RULES ∪
                                  {g_0(f(g_0)) ->w_0 g_1(f(g_1)) w_1...w_{n-1} g_n(f(g_n)) w_n}
10             end
11          end
12       end
```

[注] 手続き BUILD_RULE は、接続可能性を考慮して、上記の手続き BUILD_ORDER を用いて構成された、拡張ﾌﾟﾛﾀﾞｸｼｮﾝ･ｸﾞﾗﾌの接続規則の集合 RULES を構成する。ここで構成される接続規則は、その構成の仕方から文脈自由である。$ は接続規則の開始記号である。BUILD_RULE の中で使われる評価関数 f は、ﾗﾍﾞﾙ付き拡張ﾌﾟﾛﾀﾞｸｼｮﾝ･ｸﾞﾗﾌに、そのﾗﾍﾞﾙから計算される評価値を割当てる。g をﾌﾟﾛﾀﾞｸｼｮﾝ X_0 -> w_0 X_1 w_1 ... X_n w_n に対する拡張ﾌﾟﾛﾀﾞｸｼｮﾝ･ｸﾞﾗﾌとするとき、

f(g) = ($A(X_0)$ の要素に対応するﾉｰﾄﾞに付けられたﾗﾍﾞﾙの最大値)

と定義される。すなわち f は、g に対し、g における X_0 への訪問回数を割当てる。