ソフトウェア工学 53－3
（１９８７. 5. ２７）

インクリメンタルな属性評価の一手法

海尻　賢二

信州大学　工学部

属性文法はプログラミング環境の生成器に対する入力記述言語としてよく用いられている。対話型のプログラミング環境においては素早い応答が要求されるため各種処理（構文解析、意味解析、コード生成、表示等）はインクリメンタルに行う必要がある。本報告ではそのような情報が属性として記述されているとして、属性文法の属性値のインクリメンタルな評価の新しい方法を提案する。この問題についてはＲｅｐｓが次の２つのアルゴリズムを与えている。（１）非循環属性文法に対する前処理をしないアルゴリズム、（２）絶対非循環属性文法に対する前処理をするアルゴリズム。環境の生成時に前処理を行っておけば、実行時の属性の評価は効率的になる。非循環属性文法に対する前処理による属性評価法としてはＣｏｈｅｎのlocal control automata(LCA)による方法がある。本報告ではＬＣＡに対するインクリメンタルな属性評価アルゴリズムについて述べる。

ON INCREMENTAL EVALUATION OF NONCIRCULAR ATTRIBUTED GRAMMARS

Kenji KAIJIRI
Faculty of Engineering, Shinsyu University
500 Wakasato Nagano 380 Japan

Attribute grammars permit context-dependent languages features to be expressed in a modular, declarative fashion, so are a good basis for specifying programming environments. In an interactive programming environment, informations associated with programms must be updated incrementally after every modification. There are two kinds of methods for attribute evaluation: One analyzes the attribute grammar before analyzing programs and generates some control informations(automata) for attribute evaluation, and the others not. A method is presented to augument the Cohen's LCA(Local Control Automata) evaluator for a noncircular attribute grammar into an incremental one. The Cohen's evaluator is the former one.

1。まえがき

属性文法はｋｎｕｔｈによって提案された、文脈自由言語の意味記述のための文脈自由文法の拡張である［1］。最近この属性文法がプログラミング環境の記述に用いられることが増えてきている［2］。統合されたプログラミング環境においては構文のみならず意味の処理も要求される。たとえばプログラミング環境の主要な要素である構文エディタでは静的な意味の解析を必要としている。プログラミング環境の生成器を考える際、静的意味の記述に属性文法がよく用いられている。プログラミング環境、特に構文エディタではプログラムをインクリメンタルに処理する。そのため構文解析、及びその後の静的意味の解析もインクリメンタルに行う必要がある。即ち属性値のインクリメンタルな評価が必要となる。本論文ではこの様な目的のもとで、属性文法の属性値のインクリメンタルな評価について考察する。

属性文法の属性値のインクリメンタルな評価についてはＲｅｐｓが文献［3］［4］において詳しく考察している。そして次の２つのアルゴリズムを与えている。

（1）非循環属性文法に対するO(｜Affected｜)での評価アルゴリズム。

（2）絶対非循環属性文法に対するtree walk evaluator［5］のためのO(｜Affected｜)での評価アルゴリズム。

ここでAffectedは再評価によって値の変わる属性の集合である。

またＹｅｈは文献［6］においてordered attribute grammar［7］と呼ばれる属性文法の部分クラスに対するインクリメンタルな評価法を与えている。

属性文法の属性値の評価を効率的に行うためには何等かの戦略が必要となる。属性文法の前処理によって一種のオートマトンを作成し、そのオートマトンに従って評価を行うという方法がいくつか提案されている。Ｒｅｐｓの２番目のアルゴリズム及びＹｅｈのアルゴリズムはその様な方法に基づいている。但しその対象は絶対非循環属性文法に限られている。ところがＲｅｐｓの１番目のアルゴリズムはそうではない。そのため評価中に情報を計算しながら評価戦略を決定している。非循環属性文法に対するオートマトンによる属性評価法としてはＣｏｈｅｎのlocal control automata(LCA)による方法がある［8］。この方法は前処理をしない方法と比べて、時間複雑性では同じであるが、その動作がオートマトンで決定されるため、より効率的である。本論文ではＬＣＡによる非循環属性文法の属性値のインクリメンタルな評価法を提案する。

2。属性文法

本節では文献［1］［9］に従って属性文法について簡単に述べる。なお細かな定義は［1］［9］に従い省略する。文脈自由文法をG=(V,N,S,P)とする。ここでVは終端語の有限集合、Nは非終端語の有限集合、SはNの要素で開始記号、そしてPは生成規則の有限集合である。開始記号は生成規則の右辺にはあらわれないものとする。

文脈自由文法に対してつぎのような意味規則を付け加えたものが属性文法である。個々の非終端語Xに対して属性の有限集合A(X)を対応づける。A(X)は２つの排反する集合S(X)とI(X)に分けられる。I(X)の要素を相続属性(Inherited attribute)，S(X)の要素を合成属性(Synthesized attribute)と呼ぶ。A(X)の任意の要素をａとする時、これをX.ａとも書く。Ｐはｍ個の生成規則からなるとし、ｐ番目の生成規則を

$$X_{p0} \rightarrow X_{p1} X_{p2} \ldots X_{pk}$$

と表す。以下では特にあい昧さのない限り、各生成規則の右辺の長さをｋとする。意味規則f_{pja}はすべての$1 \leq p \leq m$について$j=0$の時は$S(X_{p0})$の要素、$j>0$のときは$I(X_{pj})$の要素に対して定義される。f_{pja}は$A(X_{pi})$の要素である属性の値から属性ａの値を与える写像である。$X_{pj}.b$がf_{pia}の引数である時、$X_{pi}.a$は$X_{pj}.b$に依存するという。生成規則においてはX_{pi}とX_{pj}は同じものでありうる。そこで生成規則が文脈上から明かな場合、属性$a \in A(X_{pi})$を(a,i)とも書く。

解析木に属性を付加したものを意味木と呼ぶ。解析木のノードのラベルをＸとすると、意味木はA(X)もラベルとして持つ。

文法　G=<{S A B},{1 0},S,P>
生成規則　P:
0:S→1 0 A
　{S.a=A.a+A.b;　A.c=A.a}
1:A→1 A 0 B
　{$A_1.a=A_2.a$; $A_1.b=A_2.b+B.a$;
　B.d=A_1.c; B.c=1; A_2.c=1}
2:A→1 0 0
　{A.a=1; A.b=2}
3:A→1 0
　{A.a=1; A.b=A.c}
4:B→1 0
　{B.a=B.c}
5:B→0 1
　{B.a=B.d}

属性 A:

I(S)={ }	S(S)={a}
I(A)={c}	S(A)={a b}
I(B)={c d}	S(B)={a}

図１　属性文法の例［8］

3。Local Control Automataによる属性の評価［8］

［定義］生成規則ｐの依存グラフD_p

次の様に定義される方向性グラフを生成規則ｐの依存グラフと呼ぶ。D_pの節はすべての$0 \leq i \leq k$に対するA(X_{pi})の要素である。節(ａ，ｉ)から(ｂ，ｊ)への枝があるとは(ｂ，ｊ)が(ａ，ｉ)に依存するということである。［定義終］

依存グラフは生成規則ｐ内での属性の直接的な依存関係を表現している。

［定義］意味木Ｔの複合依存グラフD(T)

Ｔの各ノードで適用されている生成規則に対する依存グラフをＴに従って統合したものをＴの複合依存グラフと呼ぶ。［定義終］

複合依存グラフは意味木Ｔにおけるすべての属性のインスタンス間の直接的な依存関係を表現している。

［定義］意味木Ｔのノードｑにおける特徴グラフC_q

Ｔのｑを根とする部分木をT_qとする。C_qはつぎの様に定義される方向性グラフである。T_qのラベルをＸとするとC_qの節はA(X)の要素である。ａをI(X)の要素、ｂをS(X)の要素とする。T_qにおいてX.ａからX.ｂへの路があるならば(ａ，ｂ)はC_qの枝である。［定義終］

特徴グラフはｑにおける属性のインスタンス間(相続属性と合成属性)の依存関係を表現している。

［定義］利用可な属性のインスタンス

属性のインスタンスの値と、その意味規則を評価した値とが等しい時、その属性のインスタンスは利用可であると言う。［定義終］

意味木Ｔの属性評価とはＴ上のすべての属性のインス

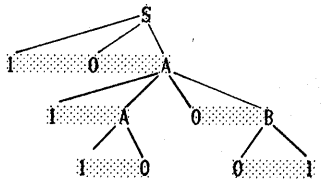


図２　入力10110001に対する解析木

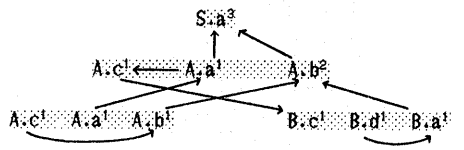


図３　入力10110001の解析木に対する複合依存グラフ
（右肩の数値は属性の値）

タンスを利用可とする事である。

Ｃｏｈｅｎらは意味木Ｔ上での属性の評価順を決定するために生成規則毎にLocal Control Automaton(LCA)と呼ぶ有限オートマトンを導入した。生成規則ｐに対するＬＣＡをA_pと書くことにする。A_pの状態はこの状態に入った時に利用可能な、生成規則ｐの適用されているノードにおける属性のインスタンスをあらわしている。またA_pからの遷移は次の３つの命令を表現している。

（１）ｃａｌｌ(ａ,ｋ)：生成規則ｐの適用されているノードにおける意味関数f_{pk}の評価を指示する。即ち属性のインスタンス(ａ,ｋ)の値を計算する。

（２）ｔｒａｎｓｆｅｒ(０)：現在のＬＣＡの動作を中断し、親のノードのＬＣＡの動作を再開する。これは現在のノード以下の属性の評価が終了したか、またはその評価のために属性値の確定していない相続属性の値が必要である事を示している。後者の場合は意味木の他の部分の評価の後、再びこのＬＣＡが活性化される。これを親ノードの走査と呼ぶ。

（３）ｔｒａｎｓｆｅｒ(ｋ),ｋ＞０：現在のＬＣＡの動作を中断し、ｋ番目の子供のノード(生成規則ｐの右辺のｋ番目の要素に対応するノード)のＬＣＡの動作を開始(再開)する。これを子供ノードの走査と呼ぶ。

A_pは$S(X_{p0})$と$I(X_{pi})(1 \leq i \leq k)$の評価を行う。ＬＣＡの状態は現在動作中である事をしめすactive stateと、中断中であることを示すsuspend stateの２種類に分類される。ＬＣＡはｔｒａｎｓｆｅｒ命令によりactive stateからsuspend stateへ遷移する。最終状態と初期状態もまたsuspend stateである。ノードに対して生成規則は一意的に決まるのであるから以下の記述においてあい味さのない限り、ノードという表現で、ノードに適用されている生成規則のインスタンスを表す。

Ordered attributed grammar及びtree walk evaluatorでは属性の評価順は意味木の構造には関係なく、生成規則毎に決っている。しかしＬＣＡでは属性の評価順は生成規則だけでなく意味木に依存する。そこでＬＣＡの動作関数と遷移関数を次の様に表現する。

Active stateに対する関数

動作関数　Action_a(machine,state,graph)

あるＬＣＡ(machine)のある状態(state)に対する動作を表す。Machineとstateのみで決まる場合とノードの特徴グラフ(graph)を必要とする場合とがある。

遷移関数　Goto_a(machine,state,graph)

あるＬＣＡ(machine)のある状態(state)に対する次の状態を表す。Machineとstateのみで決まる場合とノードの特徴グラフ(graph)を必要とする場合とがある。

Suspend stateに対する関数

遷移関数　Goto_s(machine,state,attr)

あるＬＣＡ(machine)のある状態(state)に対する次の状態を表す。条件遷移の時はtransmitted setである属性集合(attr)を利用する。

Suspend stateが(再)活性化される時には、それが親のノードからであれば新たな相続属性が、子供のノードからであれば新たな合成属性が利用可能なものとしてわたされる。これをＣｏｈｅｎはtransmitted setとよんでいる。Transmmited setがどのようなものであるかは意味木の構造による。評価時には各ノードのＬＣＡ間のパラメータのような形で受け渡される。

アルゴリズムの記述の便宜上属性の集合Ａに対して次の２つの演算(／,＊)を定義する。

$A/k = \{a \mid (a,k) \varepsilon A\}$

$(A/k)*n = \{(a,n) \mid (a,k) \varepsilon A\}$

演算／は生成規則に束縛された属性表現(ａ,ｋ)を文法記号に束縛された表現ａに変換する。＊はその逆である。

アルゴリズム１　ＬＣＡによる意味木の評価

[入力] 意味木Ｔ

[出力] すべての属性値が利用可な意味木Ｔ

[変数]

node：現在評価中の意味木Ｔのnon_leaf　node

node.state：suspend状態である時、その状態を記憶する

node.graph:nodeの特徴グラフ

[関数]

Machine(node)：nodeに適用されている生成規則のインスタンスに対応するＬＣＡ

Production(node)：nodeで適用されている生成規則のインスタンス

AvailableAttr(node,state):状態stateにあるノードnodeで利用可な属性の集合。ＬＣＡにおいては状態が決まればその時に利用可能な属性の集合は一意的にかつ静的に決まる。よってこの関数は表引きによって実現できる。

Parent(node):nodeの親ノード

Child(node,k):nodeのｋ番目の子供のノード

[手順]

```
begin
  Tの各ノードの特徴グラフを作成;
  Tの各ノードの状態を初期状態に設定;
  node=Tの根ノード;  machine=Machine(node);
  state=machineの初期状態(node.state);  attr=空;
  loop
    if state = active_state then
      動作=Action_a(machine,state,node.graph);
      if 動作 = call(a,k) then
        属性(a,k)の評価;
        state=Goto_a(machine,state,node.graph);
      else  (*  transfer(k)  *)
        node.state
```

```
        =Goto_a(machine,state,node.graph);
      if k=0 かつ node=根ノード then
        評価終了;
      elseif k=0 then
        nodeは親ノードのｎ番目の子供である;
        attr=(AvailableAttr(node,state)/0)¥n;
        node=Parent(node);machine=Machine(node);
      else
        attr=(AvailableAttr(node,state)/k)¥0;
        node=Child(node,k);
        machine=Machine(node);
      endif;
      state=node.state;
    endif;
  else
    state=Goto_s(machine,state,attr);
  endif;
endloop;
end.
```

アルゴリズム１は０(ｌＴｌ)でＴの評価を行う。図４に生成規則S->10Aに対するＬＣＡを，又図５に入力10110001に対する属性評価の流れをそれぞれ示す。図５においては各状態で新しく利用可と成った属性のインスタンスのみ示している。

```
state 0      SUS0[{}]0  初期状態
  GOTO_S(A0,0)=1
state 1      [{}]0
  ACTION_A(A0,1)=transfer(3)
  GOTO_A(A0,1)=2
state 2      SUS3[{}]0
  GOTO_S(A0,2,{(a,3) (b,3)})=3
  GOTO_S(A0,2,{(a,3)})=7
state 3      [{(a,3) (b,3)}]0
  ACTION_A(A0,3)=call(c,3)
  GOTO_A(A0,3)=4
state 4      [{(a,3) (b,3) (c,3)}]0
  ACTION_A(A0,4)=call(a,0)
  GOTO_A(A0,4)=5
state 5      [{(a,3) (b,3) (c,3) (a,0)}]0
  ACTION_A(A0,5)=transfer(0)
  GOTO_A(A0,5)=6
state 6      SUS0[{(a,3) (b,3) (c,3) (a,0)}]0
  最終状態
state 7      [{(a,3)}]0
  ACTION_A(A0,7)=call(c,3)
  GOTO_A(A0,7)=8
state 8      [{(a,3) (c,3)}]0
  ACTION_A(A0,8)=transfer(3)
  GOTO_A(A0,8)=9
state 9      SUS3[{(a,3) (c,3)}]0
  GOTO_S(A0,9,{(a,3) (c,3) (b,3)})=4
```

図４　生成規則　S->10A　に対するＬＣＡ　A0

４．ＬＣＡによる属性のインクリメンタルな再評価

すべての属性のインスタンスが利用可な意味木をＴとする。Ｔのあるノードｒを根とする部分木Trをｑを根とする部分木Tqによって置換したとする。置換された意味木をＴ'とする。Ｔ'の属性のインスタンスをすべて利用可にすることについて考察する。

前処理に基づく属性評価では意味木のすべてのノードが状態を持つ。そしてこの状態に従って評価器は評価順を決定する。Tqは最初から評価を始めるのであるからTqのすべてのノードの状態は初期状態に設定すればよい。しかし(Ｔ－Tq)は途中からの評価であるから、Tqの根ノードを最初に評価しようとした時点での状態に再設定する必要がある。Ｒｅｐｓのtree walk evaluatorに基づく方法では各ノード毎に確定する状態系列を遡ることにより再設定を行っている(子供のノードの状態系列は親ノードの状態系列より導くことができる。そのため再設定は親子関係より局所的に行なえる。)

ＬＣＡによる評価器は非循環属性文法に対して適用可であり、そのためＬＣＡによる属性の評価においては評価の順序は各ノードが持つ状態、特徴グラフ、親又は子供のノードから渡されるtransmitted set、そして各ノードに対応するＬＣＡによって決定される。そこでこれら

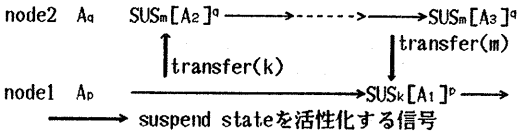
```
A0    0 SUS0[]
      1 []
      2 SUS3[]..[]
A1      0 SUS0[]
        1 []
        2 [(c,4)]
        3 [(c,4) (c,2)]
        4 SUS2[(c,4) (c,2)]
A3        0 SUS0[]...[(c,2)]
          4 [(c,0)]
          5 [(c,0) (a,0)]
          6 [(c,0) (a,0) (b,0)]
          7 SUS0[(c,0) (a,0) (b,0)]
A1      4 SUS2[--]...[(a,0) (b,0)]
       10 [-- (a,2) (b,2)]
       11 [-- (a,2) (b,2) (a,0)]
       13 SUS0[-- (a,2) (b,2) (a,0)]
A0    2 SUS3[]...[(a,0)]
      7 [(a,3)]
      8 [(a,3) (c,3)]
      9 SUS3[(a,3) (c,3)]
A1      13 SUS0[--]...[(c,3)]
        20 [-- (c,0)]
        15 [-- (c,0) (d,4)]
        16 SUS4[-- (c,d) (d,4)]
A5        0 SUS0[]...[(d,4)]
          1 [(d,0)]
          2 [(d,0) (a,0)]
          3 SUS0[(d,0) (a,0)]
A1      16 SUS4[--]...[(a,0)]
        21 [-- (a,4)]
        22 [-- (a,4) (b,0)]
        23 SUS0[-- (a,4) (b,0)]
A0    9 SUS3[--]...[(b,0)]
      5 [-- (b,3)]
      6 [-- (b,3) (a,0)]
     10 SUS0[-- (b,3) (a,0)]
```

図５　入力10110001に対するＬＣＡを利用した属性評価の流れ

を利用した状態の再設定についてまづ考える。

Transfer(m)によるsuspend stateをm-th suspend stateと呼ぶ。Active stateを［A］^{p}と，m-th suspend stateをＳＵＳ$_{m}$［A］^{p}と表す。ここでAは各状態で利用可な属性集合を表す。

あるノードnode1とそのｋ番目の隣人node2を与える。node1はnode2のm番目の隣人とする。node1で適用されている生成規則をｐ、node2で適用されている生成規則をｑとする。A$_{p}$はtransfer(k)によりＳＵＳ$_{k}$［A$_{1}$］^{p}に入り、かつＳＵＳ$_{m}$［A$_{2}$］^{q}にあるA$_{q}$を活性化する。その後A$_{q}$はtransfer(m)によりＳＵＳ$_{m}$［A$_{3}$］^{q}に入り、かつＳＵＳ$_{k}$［A$_{1}$］^{p}にあるA$_{p}$を活性化する。この時ＳＵＳ$_{k}$［A$_{1}$］^{p}をＳＵＳ$_{m}$［A$_{2}$］^{q}のactivating stateと呼ぶ。活性化には２つの場合がある。１つは子供のノードの活性化であり、ｋ＞０かつｍ＝０である。この時ＳＵＳ$_{k}$［A$_{1}$］^{p}がｎ番目のk-th suspend stateであれば、ＳＵＳ$_{0}$［A$_{2}$］^{q}はｎ番目の0-th suspend stateである。もう１つは親のノードの活性化であり、ｋ＝０かつｍ＞０である。この時はＳＵＳ$_{0}$［A$_{1}$］^{p}がｎ番目の0-th suspend stateならばＳＵＳ$_{m}$［A$_{2}$］^{q}はｎ－１番目のm-th suspend stateである。状態の再設定とはある状態に対するactivating stateを求める事である。



次の関数ActivatingState(node,n,m,state)はnodeにおけるmachineのn番目のm-th suspend stateのactivating stateを求める。

関数SuspendState(node,n,k)はnodeに適用されているＬＣＡの、nodeにおけるｎ番目のk-th suspend stateを求める。ＬＣＡの遷移には条件遷移が含まれている。従ってｎ番目のk-th suspend stateを求めるためにはそのノード上で初期状態から疑似的に遷移を行う必要がある。Ｒｅｐｓのtreewalk evaluatorに基づく方法では条件遷移がないため、単にノード毎(または生成規則毎)の命令系列を順に走査するだけでよい。

関数Exit(node,state)は状態stateにあるノードnodeからの疑似的なexit遷移を行う。ＳＵＳ$_{k}$［A］^{p}のtransmitted setは親ノード(子供ノード)を経由しての走査の結果，新たに利用可となったＩ(X$_{p0}$)の要素(ｋ＝０の時)又はS(X$_{pk}$)の要素(ｋ＞０の時)である。この走査によって何が新たに利用可となるかは現在利用可な属性集合Ａと意味木Ｔの構造による。ノードｓの親ノードの走査に関連した、属性の依存関係を示すものに上位依存グラフｓ.Ｃ［４］がある。同様にノードｓの子供ノードの走査に関連した属性の依存関係を示すものに下位依存グラフｓ.Ｃがある。属性ａが利用可でかつａからｂへの直接の路がｓ.Ｃに存在すれば、状態ＳＵＳ$_{0}$［A］^{p}にあるノードｓよりの親の走査により新たにｂが利用可となる。又ａからｂへの直接の路が（ｓのｋ番目の子供のノード）.Ｃに存在すれば状態ＳＵＳ$_{k}$［A］^{p}にあるノードｓよりのｋ番目の子供の走査により新たにｂが利用可となる。

```
関数 ActivatingState(node,n,m,state)
begin
  if m=0 then
    nodeはParent(node)のｋ番目の子供であるとする;
    state=SuspendState(parent(node),n,k)
  else
    state=SuspendState(Child(node,m),n-1,0)
  endif;
  return state;
end ActivatingState.

関数 SuspendState(node,n,k)
begin
  machine=Machine(node);
  state=machineの初期状態;
  h=n;
  while not(h=1 かつ stateがk-th suspend state) do
    if state=k-th suspend state then
      h=h-1
    endif;
    state=Exit(node,state);
    while state≠suspend state do
      state=Goto_a(machine,state,node.graph)
    endwhile;
  endwhile;
  return state;
end SuspendState.

関数 Exit(node,state)
begin
  stateをＳＵＳ$_{k}$［A］$^{p}$とする;
  attr=AvailableAttr(node,state);
  T={};
  if k>0 then   (＊ 疑似descendant scan ＊)
    for all 相続属性ａεＩ(X$_{pk}$) do
      if a#kεattr かつ (a,b)はchild(node,k).Ｃの枝
      then
        Ｔ＝ＴＵ{b};
      endif;
    endfor;
  else          (＊ 疑似ancestor遷移 ＊)
    for all 合成属性ａεＳ(X$_{p0}$) do
      if a#kεattr かつ (a,b)はnode.Ｃの枝 then
        Ｔ＝ＴＵ{b};
      endif;
    endfor;
  endif;
  state=Goto_s(Machine(node),state,T);
  return state
end Exit.
```

再評価または状態の再設定においては親または子供のノードの走査をスキップして遷移をシミュレートする。親又は子供のノードからの復帰の際ＬＣＡはtransmitted setを返し、それに従いsuspendであった状態は条件的に遷移し新たなactive stateに変わる。そのためtransmitted setを求める必要がある。ノードnodeの状態stateにおけるtransmitted setは、stateのactivating stateに対応する利用可な属性である。次の関数TransmittedSet(node,state)はノードnodeの状態stateにおけるtrans

mitted setを求める。

```
関数 TransmittedSet(node,state)
begin
  stateはnodeのn番目のm-th suspend stateである;
  wstate=ActivatingState(node,n,m,state);
  wnode=nodeのm番目の隣人;
  nodeはwnodeのｋ番目の隣人である;
  return (AvailableAttr(wnode,wstate)/k)*m;
end  TransmittedSet.
```

再評価にあたっては意味木Ｔのすべてのノードの属性の再評価は必要でない。そこでreactivatedなる集合を用意して、再評価の必要なノードはこの集合の要素とする。又再評価にあたってはノードに対応するＬＣＡの状態の再設定も必要となる。手続きAddReactivated(node,k)はノードnodeのk番目の隣人を集合reactivatedに加え、ノードの状態に対応するactivating stateにｋ番目の隣人の状態を設定する。

```
手続き AddReactivated(node,k)
begin
  if  k=0  then
    nodew=Parent(node);
    nodeはnodewのn番目の子供であるとする;
    nodew.state
          =ActivatingState(node,0,n,node.state);
  else
    nodew=Child(node,k);
    nodeの状態はn番目のk-th suspend stateである;
    nodew.state
          =ActivatingState(node,n,k,node.state);
  endif;
  reactivated=reactivated∪{nodew};
end AddReactivated.
```

以上の手続き、関数を利用すれば、ＬＣＡによる意味木のインクリメンタルな再評価アルゴリズムは次の様に実現できる。T_qの各ノードはまだ一度も走査されていないので、対応するＬＣＡの初期状態に設定する。Ｔ－T_qの各ノードは再評価の必要なもののみ状態の再設定が必要となる。そこでAddReactivatedで集合reactivatedに加える時に状態の再設定も行う。手続きReevaluate(node)はnodeを基点として属性の再評価を行う。

アルゴリズム２　ＬＣＡによる意味木のインクリメンタルな再評価

［入力］
　意味木ＴにおいてT_rをT_qで置換したもの(Ｔ')

［出力］Ｔ'に対して属性値を再評価したもの。

［手順］

```
begin
  Ｔ'の各ノードの上位及び下位依存グラフを作成;
  Tqの各ノードの状態を初期状態に設定;
  ノードｑの相続属性の値=ノードｒの相続属性の値;
  Tqの他の属性の値をnullとする;
  qのすべてのancestorノードの特徴グラフを空とする;
  node=q;
  machine=Machine(node);
  state=machineの初期状態(node.state);
  attr=TransmittedSet(node,state);
  reactivated={Tqの非終端ノード} ;
  Reevaluate(node);
end.
```

```
手続き Reevaluate(node)
begin
  repeat
    動作=Action_a(machine,state,node.graph);
    if state = active_state then
      case 動作 of
      call(a,k):
        属性(a,k)の評価;
        if 新しい属性≠古い属性 かつ k番目の隣人
           はreactivatedに含まれない then
          AddReactivated(node,k);
        endif
        state=Goto_a(machine,state,node.graph);
      transfer(k):
        if Goto_a(machine,state,node.graph)=最終状
           態 then
          reactivatedからnodeを除く;
        endif
        node.state
            =Goto_a(machine,state,node.graph);
        if k=0 then (*  to  parent  *)
          if node=ルートノード then
            評価終了; (*  この場合はすべての属性
            の再評価により終了  *)
          else
            nodew=Parent(node);
            nodeは親ノードのｎ番目の子供である;
            if nodewはreactivatedに含まれない then
             (*  再評価は不要なので親ノードをスキ
              ップする  *)
              attr=TransmittedSet(node,state)*n;
            else (*  再評価必要  *)
              machine=Machine(nodew);
              attr=(AvailableAttr(node,state)/0)*n
              node=nodew;
            endif
          endif
        else (*  to child  *)
          nodew=Child(node,k);
          if  nodewはreactivatedに含まれない  then
          (* k番目の子供のノードをスキップする *)
            attr=TransmittedSet(node、state)*0;
          else
            machine=Machine(nodew);  node=nodew;
            attr=(AvailableAttr(node,state)/k)*0;
          endif
        endif;
        state=node.state;
      endcase;
    else
      state=Goto_s(machine,state,attr);
    endif
  until reactivated = 空;
end  Reevaluate.
```

属性(a,k)を評価した後、新しい属性と古い属性が等しくなく、かつk番目の隣人が集合reactivatedに含まれていなければ、k番目の隣人をreactivatedに加える。$X_{r1}=X_{qk}$とするとノードqの属性(a,k)のインスタンスはノードrの属性(a,1)のインスタンスと同じものである。そこで(a,k)が更新されれば(a,1)も更新されたことになり、それを含むノードrのすべての属性の再評価が必要となる。故にrをreactivatedに加える。k=0の時はrはqの親ノードであり、k>0の時はrはqのk番目の子供のノードである。

Transfer命令により最終状態(suspend state)に入ったノード(LCA)は再びactiveになることはない。そのノードの属性の評価は終了している。そこで集合reactivatedから除く。こうしてreactivatedが空になれば再評価は終了する。

Transferにおいて再活性化するノードがreactivatedに含まれていなければそのノードの属性の再評価は必要ない、即ちすべての属性は矛盾なく利用可であることを示している。そこで評価をスキップする。評価をスキップするとはtransferによって入るsuspend stateから即、次のactive stateへ遷移することである。Suspend stateから次のactive stateへ遷移する際今まで評価していたノードからのtransmited setを必要とする。Transmitted setは現在のノードのI(X_{p0})又はS(X_{pi})である。そこで関数TransmittedSetによりこれを求める。

本アルゴリズムは最初T_qのノードをreactivatedに入れ、実行中に再評価の必要なノードが見つかるとreactivatedに追加する。そしてreactivatedに入っているノードに対応するLCAのみ働かせて再評価を行う。このため影響を受けるノードの集合をAffectedとすると、実行時間の複雑さはO(|Affected|)である。

5。改善策

アルゴリズム2は再評価において各ノードの状態、上位依存グラフ、そして下位依存グラフを利用する。しかし毎回すべてのノードについて再計算が必要な訳ではない。そのため状態については、($T-T_q$)のノードはreactivatedに加えられるもののみ、その時にAddReactivatedで再設定している。同様な事が上位依存グラフ、下位依存グラフについても言える。T_rをT_qで置換した意味木T'においては、

qから根へ至る路上のノード．．．．上位依存グラフ

qから根へ至る路上以外のノード．．下位依存グラフ

のそれぞれが元の意味木のものと変わらない。そこでnodeに各グラフの無矛盾フラグなるものを付加する。上位依存グラフに対してはnode.supflag、下位依存グラフに対してはnode.subflagをそれぞれ用意する。そしてこのflagがOFFの時、各グラフは意味木と矛盾しているとする。初期化においては変化しうるnodeについてそのflagをOFFにする。そして動作時に関数ActivatingStateで検査し、必要ならば再計算し、flagをONにする。

状態の再設定についても無駄がある。対応するsuspend stateを求める際、そのノードへのアクセスが2度目であっても関数SuspendStateでは再度初期状態より疑似遷移を行い状態をみつける。これは無駄である。そこでnode.stateflagを用意し、これがONの時は利用可、OFFの時は利用不可とする。ONの時はその状態を起点として次の対応する状態を求める。あるノードをk-th suspend stateにしてスキップしたとすると、次回スキップするためにアクセスする時もk-th suspend stateである。以下に改善したアルゴリズム及び関連する関数をしめす。

アルゴリズム3　LCAによる意味木のインクリメンタルな再評価

[入力]

意味木TにおいてTrをTqで置換したもの(T')

[出力] T'に対して属性値を再評価したもの。

[手順]

```
begin
  for all node in T' do
    if node in Tq then
      nodeの上位依存グラフと下位依存グラフの作成;
      node.supflag=node.subflag=ON;
      node.state=初期状態;
      node.stateflag=ON;
    else
      node.stateflag=OFF;
      if nodeはqから根への路上にある then
        node.supflag=ON;
        node.subflag=OFF;
      else
        node.supflag=OFF;
        node.subflag=ON;
      endif;
    endif;
  endfor;
  ノードqの相続属性の値=ノードrの相続属性の値;
  Tqの他の属性の値をnullとする;
  node=q;
  machine=Machine(node);
  state=machineの初期状態(node.state);
  attr=TransmittedSet(node,state);
  reactivated= {Tqの非終端ノード} ;
  Reevaluate(node);
end.

関数 ActivatingState(node,n,m,state)
begin
  if m=0 then
    if parent(node).subflag=OFF then
      parent(node).subordinate=下位依存グラフ;
      parent(node).subflag=ON;
    endif;
    nodeはParent(node)のk番目の子供であるとする;
    state=SuspendState(parent(node),n,k)
  else
    if child(node,m).supflag=OFF then
      child(node,m).superior=上位依存グラフ;
      child(node,m)=supflag=ON;
    endif;
    state=SuspendState(Child(node,m),n-1,0)
  endif;
  return state;
end ActivatingState.

関数 SuspendState(node,n,k)
begin
  machine=Machine(node);
  if node.stateflag=OFF then
    state=machineの初期状態;
```

```
  else
    state=node.state;
    stateはnodeのm番目のk-th suspend state;
    n=n-m;
  endif;
  l=n;
  while not(l=1 かつ stateがk-th suspend state) do
    if state=k-th suspend state then
      l=l-1
    endif;
    state=Exit(node,state);
    while state!=suspend state do
      state=Goto_a(machine,state,node.graph)
    endwhile;
  endwhile;
  node.state=state;
  node.stateflag=ON;
  return  state;
end  SuspendState.
```

6。むすび

本論文では非循環属性文法のオートマトンLCAによる評価法の、インクリメンタルな属性評価への適用について述べた。実際のプログラミング環境への適用にあたっては上位及び下位依存グラフの効率的な計算法，解析木のインクリメンタルな構成法［１０］等の問題がある。今後これらの問題を解決した上で、統合化されたプログラミング環境への適用により本アルゴリズムを評価したい。

参考文献

［１］Knuth,D.E.:Semantics of Context-Free Languages, Math. Syst. Th., Vol.2, No.2(1968), pp.127-145

［２］Reps,T.W. and Teitelbaum,T.:The Synthesizer Generator, Sigplan, Vol.19, No.5(1984), pp.42-48

［３］Reps,T.W.:Generating Language-Based Environments, MIT Press 1984

［４］Reps,T.W., Teitelbaum,T., and Demers,A.:Incremental Context-Dependent Analysis for Language-Based Editors, ACM Trans. on PLAS, Vol.5, No.3(1983), pp.449-477

［５］Kennedy,K. and Warren,S.:Automatic Generation of Efficient eveluators for Attribute Grammars, 3rd POPL(1976), pp.19-21

［６］Yeh,D.:On Incremental Evaluation of Ordered Attributed Grammars, BIT, Vol.23(1983), pp.308-320

［７］Kastens,K.:Ordered Attributed Grammars, Acta Informatica, Vol.13(1980), pp.229-256

［８］Cohen, R. and Harry, E.:Automatic Generation of Near-Optimal Translators for Non-Circular Attribute Grammars, 6-th POPL(1979), pp.121-134

［９］佐々政孝：属性文法，コンピュータソフトウェア，Vol.3, No.4(1986), pp.73-91

［１０］海尻賢二：直構文エディタのためのインクリメンタルパース，電子情報通信学会論文誌Ｄ　採録決定

付録 属性の評価に関係する基本グラフ(関係の表現)

$D(T_s)=(V_s,E_s)$とする。

下位依存グラフ　s.C：ノードｓの相続属性と合成属性の間でのノードｓの属性を経由しないT_sにおける依存関係を表す。

s.C＝$D(T_s)/V_s$

上位依存グラフ　s.C：ノードｓの合成属性と相続属性の間での、ノードｓの属性を経由しない$(T-T_s)$における依存関係を表す。

s.C=$(D(T)-D(T_s))/V_s$

グラフについての演算(－と／)を次の様に定義する。但し$A=(V_A,E_A)$、$B=(V_B,E_B)$とする。

$A-B=(V_A,E_A-E_B)$

$A/V'=(V',E')$　但しV'はV_Aの部分集合である。

$E=\{(u,v)\mid u、v\in V'$　かつ　V'の要素を含まないｕからｖへの路がＡにある$\}$

```
A0    0 SUS0[]
      1 []
      2 SUS3[]..[]
A1      0 SUS0[]
        1 []
        2 [(c,4)]
        3 [(c,4) (c,2)]
        4 SUS2[(c,4) (c,2)]
ここから再評価を始める
A2        0 SUS0[]...[(c,2)]
          5 [(c,0)]
          6 [(c,0) (a,0)]
          7 [(c,0) (a,0) (b,0)]
(b,0)の値が不一致であるから親ノードをreactivatedに加える
          8 SUS0[(c,0) (a,0) (b,0)]
A1      4 SUS2[--]...[(a,0) (b,0)]
       10 [-- (a,2) (b,2)]
       11 [-- (a,2) (b,2) (a,0)]
       13 SUS0[-- (a,2) (b,2) (a,0)]
A0    2 SUS3[]...[(a,0)]
このノードはreactivatedに含まれていないので評価はスキップする（親ノードの走査のスキップ）
      9 SUS3[(a,3) (c,3)]
A1      13 SUS0[--]...[(c,3)]
        20 [-- (c,0)]
        15 [-- (c,0) (d,4)]
        16 SUS4[-- (c,d) (d,4)]
A5        0 SUS0[]...[(d,4)]
このノードはreactivatedに含まれていないので評価はスキップする（子供のノードの走査のスキップ）
          3 SUS0[(d,0) (a,0)]
A1      16 SUS4[--]...[(a,0)]
        21 [-- (a,4)]
        22 [-- (a,4) (b,0)]
(b,0)の値が不一致であるから親ノードをreactivatedに加える
        23 SUS0[-- (a,4) (b,0)]
A0    9 SUS3[--]...[(b,0)]
      5 [-- (b,3)]
      6 [-- (b,3) (a,0)]
     10 SUS0[-- (b,3) (a,0)]
```

図６　入力101100001に対するＬＣＡを利用した属性評価の流れ