属性つき構文木上での部分木の置換に対応する属性評価の高速化

馮　安　　河野　俊二　　菊野　亨　　鳥居　宏次

大阪大学基礎工学部

あらまし　属性文法では，構文・意味を厳密，かつ，形式的に定めることができるので，属性文法を利用して言語指向システムを記述する試みが多く行われている．特に，インタラクティブシステムの記述に関しては，構文木に含まれる部分木の置換，及び，それに伴う属性値の更新が必要となる．

本報告では，構文木上に存在する非コピー属性の間の接続関係を表現するデータ構造としてコピー木，圧縮木を導入する．次に，それらのデータ構造を利用して，値の評価が必要となる属性だけを対象として，その値をインクリメンタルに計算するアルゴリズムを提案する．

Efficient Attribute Evaluation for Attributed Tree with Multiple Subtree Replacements

An Feng, Shunji Kohno, Tohru Kikuno and Koji Torii

Department of Information and Computer Sciences
Faculty of Engineering Science, Osaka University
1-1 Machikaneyama, Toyonaka, Osaka, Japan, 560

Abstract Attributed grammars are considered to be a good basis for specifying language-based systems, such as editors, because of their ability to describe declaratively a wide variety of context-dependent relationships. However, certain kinds of efficiency problems should be overcomed to apply attributed grammars to such applications that need replacements of multiple subtree of attributed trees. One of the problems is to develop an efficient algorithm to update attributed trees with a large number of attributes.

This paper presents two types of data structures : copy trees and compressed trees to access non-copy attributes, and then proposes an efficient algorithm that evaluates the minimum number of attributes in attributed tree incrementally.

1. まえがき

近年，構造エディタやソフトウェア開発環境支援システムなどのインタラクティブ・システムの開発における属性文法の適用が注目されている[2][4-5][7-10]．属性文法は文脈依存文法と同等な記述能力を持っており，システムの構文・意味を厳密，かつ，形式的に定めることができる．更に，意味情報をインクリメンタルに更新できるため，効率よく意味処理が行えるという特徴がある[8]．

属性文法では意味情報を構文木上の各節点における属性の値として表現する．インタラクティブ・システムにおけるユーザ操作は，構文木に含まれる部分木の置換，及び，それに伴う属性の更新として実現される．通常，構文木に含まれる属性の数が多くなるので，効率よく属性の更新を行うことは基本的，かつ，重要な問題である．

本報告では，従来の属性更新方法をそのまま採用した場合の問題点を説明した後，評価すべき属性の数を最小にする属性評価法を提案する．提案する評価法の特徴を以下にまとめる．

先ず，属性文法においては，意味記述は構文規則ごとに局所的である[11]ので，属性評価中に属性のコピーがしばしば起き，評価アルゴリズムの時間・空間効率が悪くなるという問題がある．それを解決するため，構文木上で離れて位置している非コピー属性の間の接続関係を表現したコピー木を新しく導入する．コピー木を利用して，非コピー属性のみを評価することを可能にする．

次に，属性の評価回数を軽減するためには，部分木の置換の度に属性評価を行うのではなく，複数の部分木置換に対しそれらをまとめて属性評価を行うことが望まれる[3][9]．提案する評価法では，ある属性の値の参照要求が来た時点で必要な属性評価を行うことにしている．属性文法の代表的な部分クラスの１つに非循環属性文法[11]がある．従来はこの非循環属性文法の部分クラスを対象にした評価アルゴリズムが与えられてきた．ここでは，非循環属性文法のクラスそのものを対象としたアルゴリズムを与える．

2. 準備

2.1 属性文法

属性文法[6]は次の①-③を満たす３項組$G=(G_0, A, R)$として定義される．

①文脈自由文法$G_0=(N, T, P, S)$･･･NとTはそれぞれ非終端記号と終端記号の集合である．Pは構文規則の集合で，$S \in N$は開始記号である．

②属性の集合A･･･各$X \in N \cup T$には属性の集合$A(X)$が割り当てられている．$A(X)$は相続属性の集合$IA(X)$と合成属性$SA(X)$の集合に分けられる．属性$a \in A(X)$を$X.a$と表す．

③意味規則の集合R･･･Pに属する各構文規則

$$p:\ X_0 ::= X_1\ X_2\ \cdots\ X_{n_p}$$

$(X_0 \in N,\ X_i \in N \cup T,\ 1 \leq i \leq n_p)$に意味規則の集合$R(p)$が割り当てられる．$R(p)$は，左辺記号$X_0$の合成属性と右辺記号$X_i (1 \leq i \leq n_p)$の相続属性の値を定める．

今，属性bの値を属性$a_i (1 \leq i \leq m)$に基づいて定める意味規則を

$$b = f(a_1, \cdots, a_m)$$

の様に記す．この意味規則の右辺をbの意味関数という．特に，右辺がa_1であるとき，この意味規則をコピー規則という．コピー規則の左辺に現われている属性bをコピー属性という．それ以外の属性を非コピー属性という．

構文規則 p における属性間の依存関係を依存グラフ$D(p) = (V_p, E_p)$で表す．

$$V_p = A(X_0) \cup A(X_1) \cup \cdots \cup A(X_{n_p})$$

$$E_p = \{(a_i, b) \mid a_i \text{がbの意味関数の引数である}\}$$

各意味関数の引数a_iが左辺記号X_0の相続属性か右辺記号X_jの合成属性であるとき，Gは正規形であるという．

図１に，簡単なプログラミング言語Lを記述した属性文法を示す．Lでは変数の２重宣言や未宣言変数の使用を誤りと見る．誤りを検出するため，宣言済みの変数の集合を２つの属性STT.in, STT.out，変数の名前をVAR.id，誤りの有無をVAR.errで表す．

```
PRO ::= STT
{ STT.in = φ }
STT1 ::= STT2 STT3
{ STT2.in = STT1.in;
  STT3.in = STT2.out;
  STT1.out = STT3.out }
STT ::= dec VAR
{ STT.out = STT.in∪VAR.id;
  VAR.err = (STT.in∩VAR.id≠φ) }
STT ::= use VAR
{ STT.out = STT.in;
  VAR.err = (VAR.id⊈STT.in) }
STT ::= ε
{ STT.out = STT.in }
```

図１　属性文法

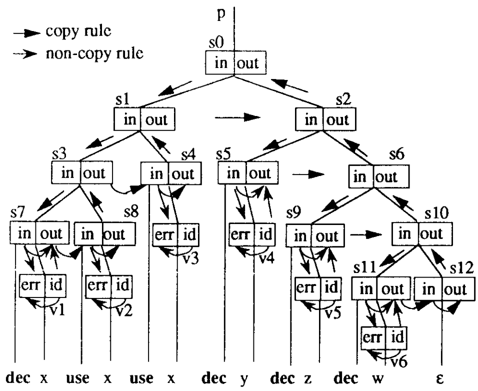

図２　属性つき構文木

Ｇによる文 $s \in T^*$ の解析について説明する．先ずsに対する構文木Tを作成し，次に意味規則に従って各節点の値を定める．後者を属性評価と呼ぶ．全ての属性値が定まった構文木を属性つき構文木と呼ぶ．

Tにおける属性間の依存関係を依存グラフD(T)で表す．D(T)はTの構造に従ってD(p)を合成したものである．

図１の属性文法による文「**dec x use x use x dec y dec z dec w**」の属性つき構文木Tを図２に示す．節点p，s_i，v_iには非終端記号PRO，STT，VARがそれぞれ対応している．各節点tの記号X(例えば，in，out)は属性t.Xを表す．図２ではコピー規則と非コピー規則を２種類の矢印で表している．これらの矢印から構成されるグラフが依存グラフD(T)を表す．

任意の文$s \in L(G)$に対する構文木TのD(T)がサイクルを含まないとき，Ｇは非循環文法（NC-AG）[11]であるという．以下では，属性文法は正規形で，かつ，NC-AGであると仮定する．

Tの属性の値がその意味関数の評価値と等しくなければ，その属性は矛盾しているという．Tの全ての属性が矛盾していなければ，Tを正しい属性つき構文木という．

2.2　更新システム S1，S2

属性値の更新システムを一般に３項組S=(T,λ,M)で定義する．ここで，Tは構文木，λはTに対する操作の集合，MはTに対する属性評価方法の指定である．直感的には，システムはλの操作を適用して構文木Tを更新し，Mに基づいて属性の計算を行う．

有向グラフ$B=(V_B,E_B)$，$C=(V_C,E_C)$と$V' \subseteq V_B$に対し，次の３つの演算を定義する．

和　$B \cup C=(V_B \cup V_C, E_B \cup E_C)$

差　$B-C=(V_B, E_B-E_C)$

射影　$B/V'=(V',E')$，但し，$E'=\{(x,y) \mid$ xからyへのB上の道で，道の途中にV'の要素を含まない道が存在する$\}$

構文木上の節点sに対し，節点sを頂点とするTの極大な部分木をT_sと表す．T_s，$T-T_s$におけるsの属性間の依存関係を表すため，下位グラフLT_s[8]と上位グラフUT_s[8]を定義する．

$LT_s=D(T_s)/A(s)$

$UT_s=(D(T)-D(T_s))/A(s)$

ここでは構文木Tに対し次のH1～H3を仮定する．

H1　構文木T上のある節点上に論理カーソル(*で表す)が置かれている．

今，論理カーソルが節点rにあるとし，Tの頂点からrへの道上の節点の集合をV1とする．

H2　各$s \in V1$に対しLT_sが計算されている．

H3　各$s \in (V-V1) \cup \{r\}$に対しUT_sが計算されている．

これまでに提案されてきた更新システムは次に示す２つのモデルS1[4][5][8]とS2[3][9]で表現できる．

モデルは，S1=(T1,λ1,M1)を次のように定める．

(1)T1･･･H1～H3の他に次のH4を満たす構文木である．

H4　全ての属性の値が評価されている．

(2)λ1={UP,DOWN,REP}(図３(a)-(c))･･･今，T1の論理カーソルが節点r上にあり，置換すべき部分木Uの頂点がrと同じ非終端記号でラベルつけられているとする．

UP･･･論理カーソルを親節点p(r)の位置へ移す．

DOWN(j)･･･論理カーソルをj番目の子節点r_jへ移す．

REP(U)･･･T_rをUで置換する．

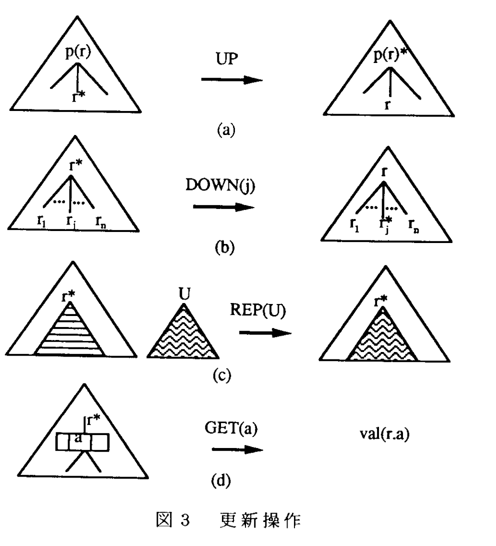

図３　更新操作

(3)M1・・・REP操作を実行する度に属性評価を行い，正しい属性つき構文木を求める．

モデルS2はS2=(T2, λ2, M2)を次のように定める．

(1)T2 = T1

(2)λ2 = λ1

(3)M2・・・k(k≧1)回のREP操作をまとめて属性評価を行い，(k回に1回の割合で)正しい属性つき構文木を求める．

モデルS2では，複数のREP操作の影響をまとめて考慮できるので，評価すべき属性の数がS1より少なくなる可能性がある．従来，モデルS1に関してはNC-AGを対象にした属性評価アルゴリズムが提案された[8]．それに対して，モデルS2に関してはNC-AGのサブクラスを対象したものしか議論されてない[9]．

3．提案する方法

3.1 コピー木

属性文法では意味記述が各構文規則毎に局所的なので，モジュール性が高いという長所がある．しかし，コピー規則が含まれていると，属性評価を行う際に，構文木の1つの節点から他の節点へ属性値をコピーする必要がある．しかも，コピー規則が意味規則全体の55%～75%を占めると言われている[2][11]．

高度の属性評価を実現するには，非コピー属性のみを評価する機構の導入が必要となる[3][11]．ここでは，非コピー属性間の接続関係を表現するために，コピー木という概念を導入する．

T上のコピー属性の集合をCAと表す．非コピー属性a∈A-CAに対し，コピー木copy(a)=(V, E)を次のように定義する．

V={a' | 依存グラフD(T)上でaからa'に至る道 $(a, c_1, \cdots, c_k, a')$が存在する．但し，k≧0，各$c_j$∈CAとする．}

E={(c, d)∈D(T)}

T上の矛盾属性の集合をFAとする．属性a∈Aに対し，フラグfl(a)を導入する．a∈CA-FAならばfl(a)=OFF，そうでなければfl(a)=ONと定める．

図2のTに対するcopy(s7.out)，copy(s4.out)，copy(s11.out)をそれぞれ図4(a)，(b)，(c)に示す．図4中のイタリック文字は属性の値を表し，白丸，黒丸はそれぞれフラグがOFFかONかを示す．

3.2 圧縮木

T上の節点の集合Yに対し，圧縮木[9]をcomp(Y)を次のように定義する．

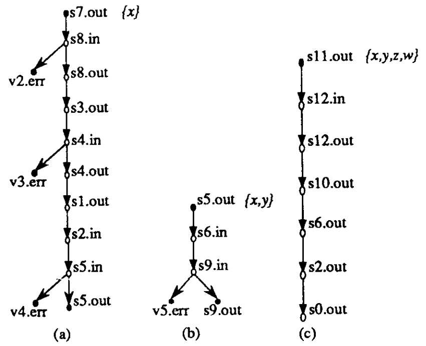

図4 コピー木

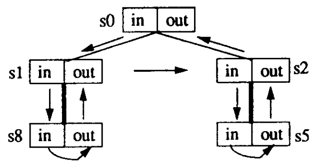

図5 図2に対するcomp({s5, s6})

comp(Y)=T/(Y∪anc(Y)∪chl(anc(Y)))

anc(X)={z | zはT上でx, y(x, y∈X)の最小の共通祖先である}

chl(X)={y | yはT上でx(x∈X)の子節点である}

圧縮木comp(Y)の枝(x, y)に対し，x∈chl(anc(Y))かつy∈anc(Y)ならばこの枝を親子枝，そうでなければ圧縮枝と呼ぶ．

圧縮木comp(Y)の連結部分cont=(CV, CE)に対し，圧縮依存グラフCD(cont)=D(T)/CVを定義する．

図2における圧縮木comp({s5, s8})と圧縮依存グラフを図5に示している．図5では親子枝，圧縮枝をそれぞれ細線，太線で表している．矢印が属性の依存関係を表す．

T_s-T_tにおけるsとtの属性間の依存関係を表すため，子孫グラフDP(s, t)を定義する．

$$DP(s, t)=(D(T_s)-D(T_t))/(A(s)\cup A(t))$$

3.3 提案するモデル S3

モデルS3=(T3, λ3, M3)を次のように定義する．

(1)T3・・・H1～H3の他に，次のH5～H7を満たす構文木である．

H5　フラグがONの各属性の値が評価されている．

H6　各非コピー属性a∈A-CAに対し，copy(a)が求まっている．

H7　T上で矛盾属性を含む節点の集合をYとするとき，圧縮木comp(Y)が構成されている．

(2) λ3={UP,DOWN,REP,GET}・・・UP,DOWN,REPはλ1，λ2と同じ．GET(a)は，属性r.a∈A(r)の値を返す．

(3) M3・・・GET(a)操作を実行する度に，集合AF-CA-NNに属する各属性の値を評価する．ここで，AFはTから正しい属性つき構文木T'への変換に伴って値が変化する属性の集合を表し，NNはD(T)上でr.aへの有向道が存在しない属性の集合を表す．

モデルS3では，フラグがONである属性に対してのみ明示的に値を与える．その他の属性の値は，必要ならばコピー木を利用して計算することができる．

モデルS3で新たに導入したGET操作は操作対象（構文木）の状態（属性値）を参照するためのものである．例えば，宣言済みの変数の集合を表す属性の値を参照することによって，ある変数を新たに宣言すべきか否かを決定できる．従来は，属性評価で得られたエラー情報がユーザに知らされる様になっており，能動的に属性値を参照するという考えは実現されていない．

一般的に，２つのGET操作の間に複数個のREP操作が行われる．モデルS3では，その値が変更されており，かつ，求めたい属性r.aに影響を与える可能性のある非コピー属性だけを評価する．従って，S3で評価する属性の数はS2に比べて少なくなる．

４.では，REP操作に伴う圧縮木とコピー木の更新について説明する．５.では，GET(a)操作を実現するアルゴリズムについて述べる．

４．置換に伴う更新

4.1　REP操作

図3(c)に示す様に，論理カーソルが節点r上にあり，部分木Uがr'を頂点とする部分木であるとする．rとr'は同じ非終端記号でラベルつけられていると仮定する．REP(U)操作は部分操作DとIから構成される．

D操作・・・TからT_rを削除する

I操作・・・節点rにUを挿入する

置換後のrの各属性r.aの値は次のように定めることにする．r.aが相続属性ならばr'.aの値を代入する．一方，r.aが合成属性ならば元のままにする．その結果，REP操作によってrの属性だけが矛盾属性になる可能性がある．例えば，図２のT_{s8}，T_{s5}を，それぞれ，図6(a)，(b)で置換する．このとき，置換後の属性s8.in,s8.out,s5.in,s5.outの値はそれぞれ{},{x},{},{x,y}となる．この中でs8.in,s8.out,s5.inが矛盾属性である．

4.2　コピー木の更新

今，wをコピー木COPY(v_0)(v_0≠w)上の頂点以外の節点とし，vをコピー木copy(u)上の葉以外の節点とする．このとき，次の関数，操作を定義する．

Pred(w)・・・copy(v_0)上の道(v_0,・・・,v_k,w)に対し，j=max{i | fl(v_i)==ON}とするとき，v_jを返す．

Succ(v)・・・集合{vj | copy(u)上の道(v,v_0,・・・,v_k)に対し，j=min{i | fl(v_i)==ON}が成立する}を返す．

Cut(w)・・・wを頂点とするコピー木copy(v_0)の極大部分木を削除する．

Link(w)・・・wはコピー木copy(v_0)の葉であり，かつ，コピー木copy(w)の頂点となっている．その２つのwを重ね合わせる．

On(w)・・・fl(w)をONにする

Off(w)・・・fl(w)をOFFにする

REP(U)操作に伴うコピー木の更新アルゴリズムD_copyとI_copyを図７に示す．D_copyではT_rに対応するコピー木の部分木を削除する．I_copyではUに対応するコピー木を挿入し，矛盾属性のフラグをONにする．

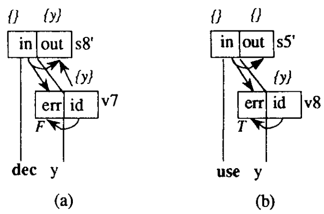

図６　置換する部分木

```
D_copy(T,r) /*コピー木の削除*/
  for a∈A(r)∩CA do
    Cut(a);
  od
end D_copy
I_copy(T,r) /*コピー木の挿入*/
  for a∈A(r)∩CA do
    Link(a);
    if val(a)≠val(Pred(a)) then On(a);
  od
end I_copy
```

図７　コピー木の更新

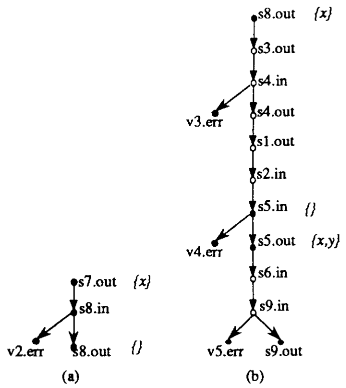

図８　置換後のコピー木

図２に対して図６の置換を行う．これに伴い，図4(a)のコピー木copy(s7.out)から，S8.in〜s8.out，s5.in〜s5.outTの部分木が削除される．最終的に求まるコピー木を図８に示す．図８上では，矛盾属性s8.in,s5.inのフラグがONとなっている．

4.3　圧縮木の更新

構文木Tの節点の部分集合をYとし，Tの圧縮木をcomp(Y)と表す．comp(Y)上の親子枝に対応するT上の枝をL枝と呼ぶ．T上の枝でL枝以外のものをH枝と呼ぶ．T上のH枝だけを含む道をH道と呼ぶ．このとき，次の関数，操作を定義する．

Path(v)・・・節点v_0を通る次の条件を満たすH道を返す．

　但し，H道を節点の系列(s_0，・・・，s_j，・・・，s_k)（s_j=v）と表すとき，始点s_0へ入るH枝，s_kから出るH枝がT上に存在しないものとする．

Head(p)・・・道pの始点を返す．

Tail(p)・・・道pの終点を返す．

Join(p,v,q)・・・２つのH枝(Tail(p),v)と(v,Head(q))をTに追加する．

Split(v)・・・Path(v)で求まるH道からvから出る枝を削除し，２つのH道に分割する．

AddNode(T,t)・・・Tの節点の集合をYとするとき，圧縮木comp(Y∪{t})を構成する．

REP(U)操作に伴う圧縮木の更新アルゴリズムD_compとI_compを図９に示す．D_compではT_rに対応する圧縮木の部分木を削除する．I_compではUに対応する圧縮木を挿入する．このとき，rに矛盾属性が存在するならrを圧縮木に加える．

```
D_comp(T,r) /*圧縮木の削除*/
  s=Head(Path(r)); t=Tail(Path(r));
  Tの圧縮木から枝(s,t)を削除する
end D_Comp

I_comp(T,r,U)  /*圧縮木の挿入*/
  if rに矛盾属性が存在する then  AddNode(T,r);
  Uの圧縮木の頂点をtとする；
  AddNode(T,t)
end I_comp
```

図９　圧縮木の更新

```
REP(U)
  D_comp(T,r);  D_copy(T,r);
  Split(r);                  /*Trの削除*/
  Join(φ,p(r),Path(r'));/*Uの挿入*/
  I_copy(T,r);  I_comp(T,r,U)
end REP
```

図１０　アルゴリズムREP

図２に対し図６の置換を適用したとき構成される圧縮木を図５に示す．

以上より，REP(U)操作に対する処理をまとめると図１０に示すアルゴリズムREPが求まる．

5．属性評価アルゴリズム

5.1　初期化

構文木Tに矛盾属性が存在する場合には，属性r.aの値を求める前に，r.aに影響を与える全ての矛盾属性の評価をしなければならない．

矛盾属性を含むT上の全ての節点の集合をYとする．Y∪{r}における属性の間の依存関係は圧縮依存グラフCD(comp(Y∪{r})で表すことができる．グラフCD(comp(Y∪{r})の中で，r.aへ到達可能な極大な部分グラフをM'とする．更に，M'の中で矛盾属性から到達可能な極大な部分グラフをMとする．Mはr.aと矛盾属性の間の依存関係を表しており，以降，初期依存グラフと呼ぶ．

Mを構成する効率よいアルゴリズムInitialを図１１に示す．先ず，変数について説明する．

構文規則p：$X ::= X_1,\cdots,X_n$を考える．関数EC(X)と$\overline{EC}(X_j)$($1 \leqq j \leqq n$)を次式のように定義する．

```
Initial(T,r.a,M)
  if r.a∈CA then b=Pred(r.a) else  b=r.a;
  AddNode(T,Node(b));
  M=LT_Node(b) ∪ UT_Node(b) の内,
        bへ到達可能な極大部分グラフ;
  V=Mにおける節点の集合;
  while s∈Vが存在する do
    Node(s)をndとする;
    if V∩SA(nd)≠φ
          && out_deg_comp(Y)(nd)>0 then
      Dis = LT_nd;
      if ndから出る枝が圧縮枝である then
        その圧縮枝を(nd,nd')とする;
        E = DP(nd,nd')∪LT_nd';
      else E = EC(nd);
    if V∩IA(nd)≠φ && in_deg_comp(Y)(nd)>0 then
      Dis = UT_nd;
      if ndに入る枝が圧縮枝である then
        その圧縮枝を(nd',nd)とする;
        E = DP(nd',nd)∪UT_nd;
      else E = ‾EC‾(nd);
    Eの内,
        Mへ到達可能な極大部分グラフをExpとする;
    Expにおける節点の集合を ND とする;
    M=(M-Dis)∪Exp;  V=V∪ND-A(nd);
  od
  S={c | in_deg_M(b)==0 }
  while d∈Sが存在する do
    S = S-{d};
    while out_deg_M(d)>0 do
      dから出る1つの枝(d,e)を削除する;
      if in_deg_M(d)==0 && dが矛盾属性でない then
        S=S∪{d};
    od
  od
end Initial
```

図11 アルゴリズム Initial

```
GET(a)
  Initial(T,r.a,M);
  S={b | in_deg_M(b)==0 }
  while c∈Sが存在する do
    S = S-{c};
    if f1(c)==TRUE then
      if val(c)≠Sem_func(c) then  ExpandDG(c);
      if c∈CA then Off(c);
      else val(c)=Sem_func(c);
    fi
    while out_deg_M(c)>0 do
      cから出る枝を(c,d)とする;
      Mから枝(c,d)を削除する;
      if in_deg_M(d)==0 then S=S∪{d};
    od
  od
  comp(Y)から矛盾属性を含まない節点を削除する;
  return Sem_func(r.a)
end GET

Sem_func(a)
  aの意味関数をf(b_1,...,b_n)とする;
  for (j=1; j<n; j++) do
    if b_j∉CA then v_j = val(b_j);
    else  v_j = val(Pred(b_j));
  od
  return f(v_1,...,v_n)
end Sem_func

ExpandDG(b)
  while Mに含まないc∈Succ(b)が存在する do
    現時点でのTの圧縮木をT1とする;
    AddNode(T,Node(c));
    現時点でのTの圧縮木をT2とする;
    CD(T2-T1)の内,
        Mへ到達可能な極大部分グラフをEとする
    M=(M-CD(T1-T2))∪E;
    入力次数が0であるEの節点をS に加える
  od
end ExpandDG
```

図13 評価アルゴリズムGET

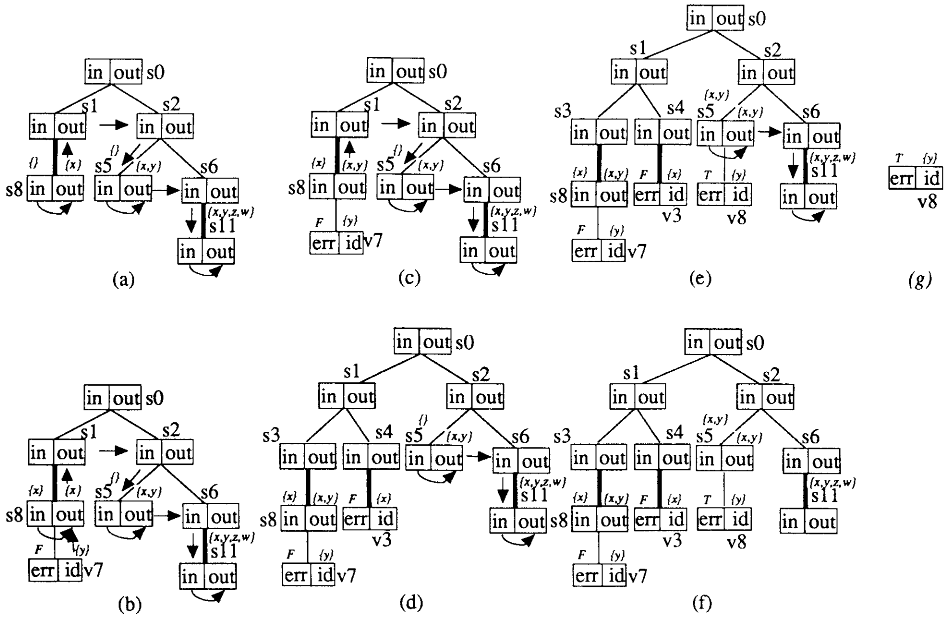

図１２　属性評価過程

$$EC(X) = D(p) \cup LT_{x1} \cup \cdots \cup LT_{xn} \quad (1)$$

$$\overline{EC}(x_j) = D(p) \cup UT_X \cup LT_{x1} \cup \cdots \cup LT_{xj-1} \cup LT_{xj+1} \cup \cdots \cup LT_{xn} \quad (2)$$

Node(a)・・・属性aが含まれている節点を表す．

次に，アルゴリズムの概要について述べる．r.aがコピー属性なら，Node(Pred(r.a))を圧縮木comp(Y)に加える．一方，r.aが非コピー属性ならr.aを圧縮木comp(Y)に加える．そして，得られた圧縮木から，初期依存グラフMを構成する．

図２に図６の置換を適用して求まる属性つき構文木に対し，Initial(T,s6.out,M)を実行した結果を図１２(a)に示す．無向枝の部分が圧縮木を，有向枝の部分が初期依存グラフMを表す．

5.2　アルゴリズムGET

図１３に属性評価アルゴリズムGET(a)を示している．GET(a)では，Initialで求めたグラフMにおける半順序関係に従って属性評価を行う．

Mの上で入力次数が0である節点は，矛盾属性に依存しないので，直ちに評価可能である．そのような節点に対応する属性cに対して，cの値を関数Sem_funcを利用して計算する．計算の前後でcの値が変化するならば，Succ(c)の各属性が矛盾属性となる可能性があるので，ExpandDG(c)を呼び出す．

ExpandDGではSucc(c)の各属性を含む節点を圧縮木に加える．引き続き，r.aへ到達可能な節点をMに加える．

この時点で，cは正しい属性となっているので，cから出る枝をMから削除する．更に，もしcがコピー属性であれば，フラグをOFFにする．最後に，矛盾属性を含まない節点を圧縮木から削除する．

図６の置換後の構文木に対し，論理カーソルがs6にある場合を考える．このときGET(out)に対する評価過程を図１２に示す．入力次数が0である属性s8.inから

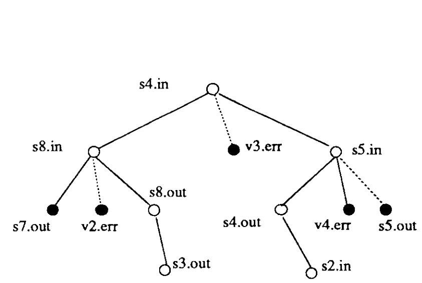

図１４　コピー木の表現

評価を開始する．s8.inの値が変更されるので，Succ(s8.in)={v7.err,s8.out}を含む節点v7とs8を圧縮木に加える（図１２(b)）．属性s8.outの値が変更された後（図１２(c)），v3を圧縮木に加える．コピー木copy(s8.out)（図8(b)）に基づいて，コピー属性{s3.out,s4.in,s4.out,s1.out,s2.in}を飛ばして，矛盾属性s5.inへ直接に移る（図１２(d)）．属性s5.outの値が変更しないので，圧縮枝(s6,s11)は分割を行わず，コピー属性を飛ばしs11.outの値を評価する（図１２(f)）．最後に，圧縮木から矛盾属性を含まない節点を削除して，図１２(g)の圧縮木が得られる．

６．計算複雑度

以降では，次の記号を用いる．

U･･･属性全体の集合

V･･･置換操作によって値が変化する属性の集合

W･･･矛盾属性の集合

X･･･非コピー属性の集合

Y･･･求めるべき属性に影響を与える属性の集合

$u=|U|$, $v=|V|$, $w=|W|$, $x=|X|$, $z=|U-X|$,

$\alpha=|V\cap X\cap Y|$, $\beta=|W\cap Y|$, $\gamma=|V\cap X|$

6.1　データ構造

コピー木と構文木を直接に実現すると，コピー木の更新（4.2）及び構文木の更新（4.3）の計算時間はO(log u)となる．これらの操作を効率よく実現するため，Sleatorらによって提案された自己調整２分木(self-adjusting binary tree)[12]を利用する．

コピー木に対する自己調節２分木による実現について説明する．コピー木copy(a)における内部節点で，かつ，フラグがONである節点の集合をON(a)とする．copy(a)を部分木の集合{Sub(s)｜s∈ON(a)}に分割する．ここで，Sub(s)=(V,E)は次のように定義される．

V={x｜copy(a)上に次の条件を満たす道$(s, c_1, \cdots, c_k, x)$が存在する．各c_j $(1 \leqq j \leqq k)$に対しfl(c_j)==OFFが成立し，かつ，xはcopy(a)の葉であるかfl(x)==ONである．}

E={(x,y)｜(x,y)がcopy(a)上の枝である．}

このとき各Sub(s)を自己調整２分木で表現する．図４(a)のコピー木copy(s7.out)に対する自己調節２分木を図１４に示す．このように実現すれば，コピー木に関する各操作はO(log u)で行える．

同様に，構文木Tを自己調整２分木で実現すれば，構文木に関する操作はO(log u)で実現できる．一方，圧縮木の依存グラフはO(m*log u)で実現できる．

6.2　評価

操作REP(U)では，4.2と4.3で述べた基本操作をO(1)回呼び出すので計算時間はO(log u)となる．操作GET(a)ではMの各属性に対し4.2と4.3の基本操作をO(1)回実行する．Mの最大サイズは$O(\alpha+\beta)$なので，GET(a)の計算時間が$O((\alpha+\beta)\log u)$となる．

１つの属性の値を保存するための必要なメモリ容量をsとする．構文木上の値を保存するための空間計算量はO(xs)である．圧縮木とコピー木に必要な空間計算量は O(z) である．故に，全体の空間計算量はO(xs+z)である．

複数の部分木の置換を許した場合の評価アルゴリズムとしては，これまでにRepsらによるアルゴリズム（アルゴリズムR）[9]，Hooverによるアルゴリズム（アルゴリズムH）[3]が知られている．これらのアルゴリズムはモデルS2に基づくものである．本報告で提案したアルゴリズムをアルゴリズムFと表して，それらの計算複雑度に関する比較を表１に示す．

表１　計算複雑度の比較

	適用クラス	空間計算量	時間計算量	
			部分木の置換	属性値の更新
アルゴリズムR	NC-AGのサブクラス	O(us)	O(log u)	O((v+w)log u)
アルゴリズムH	NC-AG	O(xs+zs)	O(u)	O(γu+log u)
アルゴリズムF	NC-AG	O(xs+z)	O(log u)	O((α+β)log u)

7．むすび

本報告では，構文木を動的に更新するシステムに対して，評価すべき属性の数を最小にする属性評価アルゴリズムの提案を行った．本アルゴリズムでは非循環属性文法のクラスを対象にしている．

現在，提案した評価アルゴリズムの実験的な評価，及び，アルゴリズムの正当性に関する形式的な議論を進めている．更に，将来の計画としては，本アルゴリズムを複数のユーザが協同でソフトウェアを開発する環境[4][7]の記述に適用する予定である．

文　献

[1]馮，杉山，藤井，鳥居："共通属性をもつ属性文法とそのPROLOGによる処理系"，信学論，J70-D，7，pp. 1311-1319(1987).

[2]馮，大野，井上，菊野，鳥居:"属性文法による構造化分析法の形式的記述"，信学技報，COMP 88-3，pp. 21-30(1988).

[3]R. Hoover:"Dynamically bypassing copy rule chains in attribute grammars", Proc.13th POPL, pp.14-25(1986).

[4]G. E. Kaiser, S. M. Kalpan and J. Micallef: "Mutiuser, distributed language-based environments", IEEE Software, 4, 6, 58-67(1987).

[5]片山卓也："属性文法に基づくソフトウェア自動生成システム構成法の研究"，昭63年文部省科研(一般研究(A))報告書(1989).

[6]D. E. Knuth: "Semantics of context-free languages", Math. Syst. Theory, 2, 2, pp. 127-145(1968).

[7]P. M. Lu, S. S. Yau and W. Hong:"A formal methodology using attributed grammars for multi-processing-system software development", J. Inform. sci., 30, pp. 79-123(1983).

[8]T. Reps: "Generating Langauge-Based environments", The MIT Press(1986).

[9]T. Reps, C. Marceau and T. Teitelbaum:"Remote attribute updating for language-based editors", Proc. 13th POPL, pp. 1-13(1986).

[10]D. Ridjanovic and M. Brodie:"Defining database dynamics with attribute grammars", Information Processing Letters, 14, 3, pp. 132-138(1982).

[11]佐々政孝: "属性文法", コンピュータソフトウェア, 3, 4, pp. 73-91(1986).

[12]D. D. Sleator and R. E. Tarjan: "Self-adjusting binary search trees", JACM, 32, 3, pp. 652-686 (1985).