ソフトウェア基礎論 20－4
（1987. 2. 13）

属性文法記述言語PANDAの処理系における最適化

馮 安　杉山 裕二　藤井 譲　鳥居 宏次
大阪大学　基礎工学部

属性文法のコピー規則問題を解決するため，著者らは共通属性文法宣言という記述法を提案した．現在，共通属性宣言を用いた属性文法の記述言語PANDAの処理系を開発している．その処理系では，属性文法記述から Prolog プログラムを生成する．本報告では，効率よいPrologプログラムを生成するための最適化手法を紹介する．

最適化は属性文法の最適化とPrologプログラムの最適化からなっている．前者は，属性の必要性を調べて冗長な属性とそれに関連する意味規則を削除する．後者は，等価変換規則を基本知識として導入し，同値クラスという概念を用いて，変数や述語の削除，共通式の重ねあわせ，DCGのfold/unfoldなどを行い，Prologプログラムの実行効率を向上させている．

OPTIMIZATIONS IN A PROCESSOR OF ATTRIBUTE GRAMMARS

An FENG, Yuji SUGIYAMA, Mamoru FUJII and Koji TORII

Faculty of Engineering Science,Osaka University,
1-1 Machikaneyama,Toyonaka,Osaka 560, Japan

（英称住所）

To solve well_known “Copy Rule Problem” of attribute grammars, authors have introduced Common Attributes. Authors are now constructing a processor of a programming language PANDA, which is developed for describing attribute grammars with common attributes. This paper presents some optimization strategies in our processor.

The optimizations improve the attribute grammar and the Prolog program generated by our processor. In the attribute grammar, the necessity of attributes is determined, redundant attributes and semantic rules are eliminated. In the Prolog program, some replacement rules are introduced into the system as primitive knowledge, and equivalent classes of variables are used for deleting redundant Prolog goals.

1. はじめに

属性文法におけるコピー規則問題を解決するために，著者らは共通属性宣言という表記法を提案した[4][7]．それを用いたコンパイラの記述にはコピー規則がわずか10%しかないという結果が得られている．従来の50-70%と比べて，大きく改善されているといえる．現在，共通属性宣言を用いた属性文法の記述言語PANDAとその処理系を開発している．PANDAの処理系では，PANDA記述からPrologプログラムを生成する．本報告では，効率よいPrologプログラムを生成するために，PANDA処理系の行っている最適化を紹介する．

最適化は，属性文法の最適化と生成されたPrologプログラムの最適化からなっている．前者は，共通属性に対応する属性の必要性を調べ，冗長な属性およびそれに関連する意味規則を削除する．後者は，等価変換規則を基本知識として導入して，変数の同値クラスという概念を用いて変数および述語を減らしたり，共通部分式の重ねあわせ，定数の畳み込み，無用代入の削除，DCGのfold/unfoldなどを行い，Prologプログラムの効率化を図っている．

2. 属性文法の最適化

共通属性を用いた記述は，通常の属性文法への変換によって意味定義がなされている[4][7]．その際に，各共通属性が合成属性および相続属性に置き換えられ，また，コピー規則が補われる．そのため，変換後の属性文法では，不必要な属性や属性間のコピー規則が多く現われるという傾向にある．そこで，属性の必要性を調べ，冗長な属性とそれに関連する意味規則を削除して効率を向上させる．まず，例題を用いて，共通属性を用いた属性文法と，それと等価な（通常な）属性文法との関係を示す．

2.1 例題

次のような共通属性を用いた記述を考える．

```
1       COMMON attr;
2       <X> ::= <Y> ";"
3         { 2: attr := attr + 1  }
4       <Y> ::= " ".
```

ここで，1行目は共通属性の宣言で，2行目と4行目は構文規則である．3行目は意味規則で，数字“2:”は意味規則の評価位置が2で，すなわち";"の右隣であることを示している．

共通属性の宣言により，非終端記号Xは相続属性X.I_attrと合成属性X.S_attrを，Yはそれぞれ相続属性X.I_attrと合成属性X.S_attrを持つ．

これらの属性の値は，“上から下へ”，“左から右へ”の意味規則の評価順序に従って渡していく．そのため，まず，意味規則

```
Y.I_attr := X.I_attr
```

が2行目の構文規則に，4行目の構文規則に

```
Y.S_attr := Y.I_attr
```

が追加され，3行目の意味規則にある共通属性は次のように置き換えられる：

```
X.S_attr := Y.S_attr + 1
```

上記の例に対応する記述は次のようになる：

```
<X> ::= <Y> ";"
  {  Y.I_attr := X.I_attr ;              ···①
     X.S_attr := Y.S_attr + 1  }.        ···②
<Y> ::= " "
  {  Y.S_attr := Y.I_attr  }.            ···③
```

このようにして得られた記述から，冗長な属性やコピー規則を削除することを考える．意味規則③によりY.S_attrの値がYI_attrといつも等しいので，意味規則②でのY.S_attrをY.I_attrで置き換えてもよい．そうすると，属性Y.S_attrが必要なくなり，記述が次のようになる．

```
<X> ::= <Y> ";"
  {  Y.I_attr := X.I_attr ;
     X.S_attr := Y.I_attr + 1 }.         ···④
<Y> ::= " ".
```

さらに，意味規則④にあるY.I_attrをY.I_attrの意味関数，すなわちX.I_attrに置き換えれば，Y.I_attrの値がどこにも参照されていないので，必要なくなる．結局，非終端記号Yの属性はすべて不要で，記述は次のようになる：

```
<X> ::= <Y> ";"
      { X.S_attr := X.I_attr + 1 }
<Y> ::= " ".
```

2.2 合成属性の削除

非終端記号の集合をV_Nで，共通属性の集合をCAで表す．共通属性を用いた記述には，“a:=a”（a∈CA）のような意味規則は存在しないと仮定する．

［定義1］合成属性X.S_aについて，次の概念を定義する：

(1)Xを根とする構文木で，X.S_aとX.I_aの値が異な

るようなものが存在するとき，またそのときに限り，X.S_aは値が変更されるという．

(2)Xを含む構文木で，X.S_aの値（値が他の属性にコピーされているときは，そのコピー先の属性の値を含む）を参照する意味規則（コピー規則を除く）が存在するとき，またそのときに限り，X.S_aは値が参照されるという．

(3)X.S_aは値が変更され，かつ参照されるとき，またそのときに限り，X.S_aが必須であるという．

X.S_aの値が変更されなければ，X.S_aとX.I_aはいつも等しいので，X.S_aをX.I_aで置き換えることにより，X.S_aを省略できる．また，X.S_aの値が参照されなければ，X.S_aの存在する価値がないので省略できる．従って，必須でない合成属性は，省略可能である．

Xの部分木を含むすべての構文木を書き出して，X.S_aの値の変更，参照を調べれば，X.S_aが必須であるか否かは分かる．しかし残念ながら，殆どの文法ではそのような構文木は無限にあるので，この方法ではX.S_aが必須であるか否かは分からない．もしそのような構文木が有限であったとしても，X.S_aが必須であるか否かはその親Zの属性Z.S_aと兄弟のX1.I_a（値の参照に関して），およびその子Yの属性Y.S_aが必須であるか否か（値の変更に関して）に依存するので，判断するには非常に時間がかかる（図１参照）．

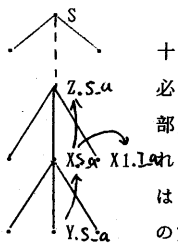

図１

そこで，値の変更のみを調べる十分条件に基づいて属性X.S_aが必須であるか否かを調べる．Xの部分木に共通属性aが陽に定義されていなければ，合成属性X.S_aはいつもX.I_aと等しいので，次の定理が得られる．

[定理１]Xを左辺記号とするすべての構文規則が次の条件(1)および(2)を満たせば，X.S_aは値が変更されない，従って必須でない．

(1)aの値を定義する意味規則をもたない．

(2)右辺記号Y（但し，X≠Y）で，Y.S_aは値が変更されない．

定理１(2)では，左辺記号Xの属性X.S_aの値が変更されるか否かは右辺記号Yの属性Y.S_aの値が変更されるか否かに依存することを表している．例えば，２．

```
common attr : int ;
<simple(main)> ::=<A>
  {  first: attr := 0 ;
     last : write(attr) }.
<A> ::= <B>.
<B> ::= "y" <C>
      {  1: attr := attr + 1}.
<C> ::= <B>.
<C> ::= <D>.
<D> ::= " ".
```
図２

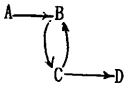

図３

１の例題で，Y.I_attrがX.I_attrからコピーされ，X.S_attrがY.S_attrからコピーされているので，X.S_attrとX.I_attrと等しいか否かは，Y.S_attrとY.I_attrが等しいか否かに依存する．Xを左辺記号とする構文規則の右辺記号である非終端記号Y（X≠Y）の全体の集合をXの合成属性（例えば，X.S_a）に関する依存集合といい，SNDEP(X)（Dependency set of Non_terminals for Synthesized attributes）と書く．図２が示している例では，各SNDEPは次のようになる：

SNDEP(A) = {B}

SNDEP(B) = {C}

SNDEP(C) = {B,D}

SNDEP(D) = {}

SNDEPを用いて，合成属性に関する依存グラフSNDG(Dependency Graph among Non_terminals for Synthesized attributes)を定義する．

[定義２] 合成属性に関する依存グラフSNDG＝(V,E)を次のように定義する：

$$V = V_N$$

$$E = \{ (X,Y) \mid Y \in SDEP(X) \}$$

図２の記述に対応する依存グラフSNDGを図３に示す．

SNDGに，図３のようなサイクルがあるかもしれない．サイクルについて定理２がある．

[定理２]SNDGにサイクル$(X_0,X_1,\cdots X_k,X_0)$があるとすると，すべての$X,Y\in\{X_0,X_1,\cdots X_k\}$ $(X\neq Y)$，$a\in CA$について，

X.S_aは値が変更されるならば，Y.S_aも値が変更される．

[証明]定義２，定理１から次の結論が直ちに得られる：SNDG＝(V,E)では，$(X,Y)\in E$について，Y.S_aの値が変更されるならば，X.S_aの値も変更される．

X_m, $X_n(1\leqq m<n\leqq k)$について，$(X_i,X_{i+1})\in E(m\leqq i$

≦n-1)から，もしX_n.S_aの値が変更されるならば，X_m.S_aの値も変更される．一方，$(X_k,X_O)\in E$，$(X_i, X_{i+i})\in E$(n≦i≦k-1または0≦i≦m-1)から，もしX_m.S_aは値が変更されるならば，X_n.S_aの値も変更される．

ゆえに，サイクル上のすべてのX,Yについて，X.S_aは値が変更されるならば，Y.S_aも値が変更される．

定理２により，サイクル上の非終端記号は，共通属性に対応する合成属性の値が変更されているか否かという観点から見ると互いに等価である．それで，サイクル上の非終端記号の集合を一つのメタ頂点とすることができる．そうすると，依存グラフでのサイクルがなくなる．変換された図をMETA_SNDG＝(MV,ME)とする．例えば，図３にメタ頂点{A},{B,C},{D}を導入すれば，図４(a)のようになる．

PANDA処理系では，共通属性aについて，次のように合成属性の必要性を決める：

(a)初期化：

すべての構文規則pについて，その左辺記号をXとすると，もし共通属性aを定義する意味規則が存在すれば，X.S_aが属するメタ頂点を必要とする．

(b)META_SNDGに必要とされたメタ頂点が存在する間，次の処理を行なう：必要とされたメタ頂点をMとする．

(1)(N,M)∈MEなるすべてのNを必要とする．

(2)Mとそれに隣接する辺をMETA_SNDGから取り除く．

(c)残るすべてのメタ頂点を不要とする．

	(a)	(b)	(c)
METER-SNDG	{A} ↓ {B,C} ↓ {D}	{A} {D}	 {D}
A.S_attr	?	必要	必要
B.S_attr	必要	必要	必要
C.S_attr	必要	必要	必要
D.S_attr	?	?	不必要

図　4

この手続きで，不要とされるメタ頂点に属する非終端記号Xの合成属性X.S_aは値が変更されていないので，省くことができる．

図２の例では，図４(a)(b)(c)のように合成属性の必要性を判断し，D.S_attrが不要であることが分かる．

上述のように，不要とされたすべての属性X.S_aについて，記述からX.S_aの値を代入する意味規則（コピー規則）を取り除き，意味関数の引数であるX.S_aをX.I_aで置き換えれば，等価で簡単な記述が得られる．共通属性を用いた図２の記述を通常の属性文法へ変換すると，結果は図５になる．図５から属性D.S_attrとそれに関する意味規則を削除すると，図６の記述になる．

```
<simple(main)> ::=<A>
  ( A.I_attr := 0 ;
    write(A.S_attr) ).
<A> ::= <B>
  ( B.I_attr := A.I_attr ;
    A.S_attr := B.S_attr ).
<B> ::= "y" <C>
      ( C.I_attr := B.I_attr + 1 ;
        B.S_attr := C.S_attr   ).
<C> ::= <B>
      ( B.I_attr := C.I_attr ;
        C.S_attr := B.S_attr   ).
<C> ::= <D>
      ( D.I_attr := C.I_attr ;
        C.S_attr := D.S_attr   ).
<D> ::= " "
      ( D.S_attr := D.I_attr ).
```

図５

```
<simple(main)> ::=<A>
  ( A.I_attr := 0 ;
    write(A.S_attr) ).
<A> ::= <B>
  ( B.I_attr := A.I_attr ;
    A.S_attr := B.S_attr ).
<B> ::= "y" <C>
      ( C.I_attr := B.I_attr + 1 ;
        B.S_attr := C.S_attr   ).
<C> ::= <B>
      ( B.I_attr := C.I_attr ;
        C.S_attr := B.S_attr   ).
<C> ::= <D>
      ( D.I_attr := C.I_attr ;
        C.S_attr := D.I_attr   ).
<D> ::= " ".
```

図６

２．３　相続属性の削除

ここでは，２．２で不要と判断された合成属性は削除したものとする．相続属性の必要性を検討する前に，まず次の概念を定義しておく．

[定義３]構文規則pを

$$X_O ::= X_1 \cdots X_k \cdots X_{np}$$

とすると，評価位置k（X_kの右隣）で共通属性aが定義されるとは，条件(1)または(2)を満たす．

(1)評価位置kでaへ値を代入する意味規則がある．

(2)X_kが非終端記号で，かつX_k.S_aが存在する．

[定義４]もし非終端記号Xを根とするある部分木にX.I_a（値がコピーされたときはそのコピー先）の値が参照されるとき，またそのときに限り，相続属性X.I_aは必須であるという．

相続属性は（構文木の中で）親からの値を伝える役目をもっており，共通属性に対応する相続属性では，その値が再定義されることはない．従って，必須でない相続属性は省くことができる．

X.I_aの値の参照は，次の３つの場合に限定される：

(1)Xを左辺記号とする構文規則の意味規則で参照される．

(2)Xを左辺記号とする構文規則で，X.S_aへ代入する値として参照される．

(3)Xを左辺記号とする構文規則の右辺記号Yの相続属性Y.I_aに値がコピーされ，Y.I_aの値が参照される．

このことから，直ちに定理３が得られる．

[定理３]相続属性X.I_aが必須である必要十分条件は，Xを左辺記号とする構文規則で，(1)または(2)または(3)を満たすものが存在することである．

(1)aが定義される前に，意味関数の引数として参照される．

(2)aが定義されず，X.S_aが存在する．

(3)次のような構文規則が存在する：

$$X ::= X_1 \cdots X_k \cdots X_{np}$$

$X_k \in V_N$，$X \neq X_k$，かつ評価位置kの前にaが定義されず，X_k.I_aが必須である．

定理３(3)では，X.I_aが必須であるか否かはX_kの属性X_k.I_aが必須であるか否かに依存することを表している．このような非終端記号X_k全体の集合をX.I_aに関する依存集合といい，$INDEP_a(X)$(Dependency setof Non_terminals for Inherited attribute X.I_a) と書く．例えば，図２の$INDEP_{attr}$は次のようである．

$INDEP_{attr}\{A\} = \{B\}$

$INDEP_{attr}\{B\} = \{\}$

$INDEP_{attr}\{C\} = \{B,D\}$

$INDEP_{attr}\{D\} = \{\}$

INDEPを用いて，相続属性に関する依存グラフINDG (Dependency Graph among Non_terminals for Inherited atttributes)を定義する．

[定義５]相続属性に関する依存グラフINDG(a)＝(V,E)（a∈CA）を次のように定義する：

$V = V_N$

$E = \{ (X,Y) \mid Y \in IDEP_a(X) \}$

２．２と同様に，依存グラフINDG(a)にメタ頂点を導入する（その図をMETA_INDG(a)とする）．

すべての共通属性aについて，次のように各非終端記号Xの相続属性X.I_aの必要性を調べる．

(a)初期化：

左辺記号がXで，①または②を満たすような構文規則が存在すれば，X.S_aが属するメタ頂点を必要とする．

①aが定義される前に，意味関数の引数として使われている．

②aが定義されず，X.S_aが必要である．

(b)META_INDG(a)に必要とされたメタ頂点が存在する間，次の処理を行なう：必要とされたメタ頂点をMとする．

(1)(N,M)∈MEなるすべてのNを必要とする．

(2)Mとそれに隣接する辺をMETA_INDG(a)から取り除く．

(c)残るすべてのメタ頂点を不要とする．

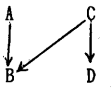

図７

	(a)	(b)	(c)
METER-INDG (attr)	(A) (C) ↓ ↘ ↓ (B) (D)	(A) (D)	(D)
A.I_attr	?	必要	必要
B.I_attr	必要	必要	必要
C.I_attr	必要	必要	必要
D.I_attr	?	?	不必要

図８

例２のINDG(attr)と属性の必要性判定過程はそれぞれ図７，図８に示されている．これより，D.I_attrが必要でないことが分かる．

不要とされたメタ頂点に属する非終端記号Xのすべての属性X.I_aについて，記述からX.I_aを定義する意味規則を取り除き，意味関数の引数として現れているX.I_aをその意味関数で置き換えれば，等価で簡単な記述が得られる．図６から属性D.I_attrとそれに関する意味規則を削除すると，図９の記述になる．

```
<simple(main)> ::=<A>
  { A.I_attr := 0 ;
    write(A.S_attr) }.
<A> ::= <B>
  { B.I_attr := A.I_attr ;
    A.S_attr := B.S_attr }.
<B> ::= "y" <C>
      { C.I_attr := B.I_attr + 1 ;
        B.S_attr := C.S_attr   }.
<C> ::= <B>
      { B.I_attr := C.I_attr ;
        C.S_attr := B.S_attr   }.
<C> ::= <D>
      { C.S_attr := C.I_attr }.
<D> ::= " ".
```

図９

3. PROLOG最適化

Prologプログラムの変換に関する研究は，unfold/foldという等価変換を中心としてなされてきた[2][6]．我々は，この研究を踏まえて，効率よいPrologプログラムの生成を目的とした最適化手法の検討を行ってきた．

ここでは，Prolog処理系が行っているPrologプログラムの最適化手法について述べる．PANDA処理系では，unfold/foldを基本とし，いくつかの等価変換規則に基づいて，変数やProlog項を減らし，静的に評価できる部分をコンパイル段階で評価するなどによってプログラムの効率を向上させている．

3．1 等価規則

PANDA処理系は，Prologプログラムの変換のため，以下のような等価規則を基本知識として利用する．

・同値規則

(1)plus(X,0,Y)が真ならば，XとYが等しい．

(2)times(X,1,Y)が真ならば，XとYが等しい．

(3)concat(X,[],Y)が真ならば，XとYが等しい．

(4)concat([],X,Y)が真ならば，XとYが等しい．

(5)X=Yが真ならば，XとYが等しい．

(6)X is Yが真ならば，XとYが等しい．

(7)null(X,Y)が真ならば，XとYが等しい．

・交換則

(8)plus(X,Y,Z)ならば，plus(Y,X,Z)

(9)times(X,Y,Z)ならば，times(Y,X,Z)

・結合則

(10)(X+Y)+Z ⇔ X+(Y+Z)

(11)(X*Y)*Z ⇔ X*(Y*Z)

但し，“+”，“*”それぞれは述語plus,timesのインフィクス表示である（以下も同様）．

・分配則

(12)(X+Y)*Z ⇔ X*Z+Y*Z

・逆関数規則

(13)plus(X,Y,Z) ⇔ minus(Z,X,Y)

(14)times(X,Y,Z) ⇔ div(Z,X,Y)

但し，(10)～(12)は，処理系に組込んで実現する．そのほかの等価変換規則は，prologで記述し，fold/unfoldによって実現する．

3．2 同値変数の統合

共通属性を用いても，属性文法記述の10%程度のコピー規則が残る．これは，属性が非終端記号毎にあるということに起因する．もしPrologプログラムで，それらの属性を同じ変数に対応させれば，変数の数が減り，また，値のコピーも必要なくなる．そこで，生成されたPrologプログラムにおいて，変数を（その値により）同値クラスに分割し，各同値クラスに一つの領域名を与え，そして，各領域名へ値を代入するProlog項を一個だけにする．この方法は，共通式の除去にも利用できる．例えば，次のようなProlog記述を考える．

```
X1 is Y * 12 ,
X2 is X1,
X3 is Y * 12
```

変数X1，X2，X3の値が等しいので，X1，X2，X3が同じ同値クラスに属する．その同値クラスの領域名をXとすると，記述は次のようになる：

```
X is Y * 12
```

この種の最適化を一般の述語に対するものに拡張するため，“一意述語”の概念を導入する．

［定義６］述語$P=A(X_1,\cdots,X_n)$において，任意のn-1個の引数（$X_1,\cdots,X_{j-1},X_{j+1},\cdots X_n$とする）の値を固定すれば，Pを真とする（残りの）引数X_jの値が一意に決まるとき，Pを一意述語と呼ぶ．

明らかに，述語“is”は一意述語である．上述の定義から，次の定理が直ちに得られる．

［定理４］$A(X_1,\cdots,X_n)$を一意述語とする．もし項$A(Y_1,\cdots,Y_n)$も$A(Z_1,\cdots,Z_n)$も真で，かつi=1,…,j-1,j+1,nに対して，Y_iとZ_iの値が等しければ，Y_jとZ_jも等しい．

“is”が一意述語であることから，上記の例に対して，この定理が適用でき，X1，X3の値が等しいことがいえる．現在処理系に組込んである一意述語は，“=”，“name”，“is”，“concat”，“append”，“plus”(+)，“minus”(-)，“times”(*)，“div”(/)，“null”などである．

PANDA処理系は，次の順で処理を行う：

(1) 同値規則（３．１参照）で変数の同値クラスの初期化を行う．

(2) 交換則，結合則，分配則，逆関数規則（３．１参照）で記述を変換し，一意述語の性質を用いて同値クラスを求める．

(3) 各同値クラスに一つの領域名を与え，記述での変

PROLOG記述	Z1 is Z2, plus(Z1,X1,W), minus(W,Z2,X2), name(Y1,[X1]), name(Y2,[X2])	Z1 is Z2, plus(X1,Z1,W), plus(X2,Z2,W), name(Y1,[X1]), name(Y2,[X2])	plus(X,Z,W), name(Y,[X])
同値クラス	{X1},{X2} {Y1},{Y2} {Z1,Z2} {W}	{X1,X2} {Y1,Y2} {Z1,Z2} {W}	X = {X1,X2} Y = {Y1,Y2} Z = {Z1,Z2} W = {W}

図１０

数を領域名で置き換え，冗長なProlog項を削除する．

例えば，次の記述を考える．

```
Z1 is Z2 ,
plus(Z1,X1,W),
minus(W,Z2,X2),
name(Y1,[X1]),
name(Y2,[X2])
```

この記述に関する最適化処理(1)(2)(3)は，図１０で示されている．ここで，(1)では，規則(6)を利用した．(2)では，規則(8)，規則(13)を用いて，記述を変形し，“plus”が一意述語であることから，X1,X2が等しいことは分かる．そして，“name”が一意述語であることから，Y1,Y2が等しいといえる．

３．３　共通部分式の重ねあわせ

プログラム変換において，効率化を図る重要な戦略の一つに，同じ結果を得ることが分かっている場合を再度実行することを避けること，即ち再計算の回避がある．PANDA処理系では，共通式については，３．２の最適化で除去できる．二つの項にある共通部分式の場合は，重ねあわせの処理を行っている．例えば，

```
Y is A + B + C ,
Y is A + B + D
```

を

```
Z is A + B  ,
Y is Z + C ,
Y is Z + D
```

へ変換する．

以上の例では，基本ブロック内に対してであるが，条件文の文脈においても行う．例えば，

```
((Z>0,!,X1 is f(0),Y is g(X1));
                X2 is f(1), Y is f(X2))
```

の場合は，まず，変数名を変える：

```
((Z>0,!,X is f(0),Y is g(X));
                X is f(1), Y is f(X))
```

そして，文脈にある共通属性を外に出し，次のように変換する：

```
((Z>0,!,X is f(0)); X is f(1)) , Y is f(X)
```

３．４　定数の畳み込み

最適化の戦略として，静的評価可能な部分はコンパイル段階で評価するということがある．定数の畳み込みはその一例である．PANDA処理系では，定数を畳み込む際，３．１で述べた変換規則のほかに，次の等価規則も用いる：

```
(15)concat(A,B,C),concat(C,D,E)
          ⇒concat(B,D,C),concat(A,C,E)
```

concatは次のように定義されている：

```
①concat([],Y,Y).
②concat([X1 | X2],Y,[X1 | Z) :- concat(X2,Y,Z).
```

また，定数間の演算の実行やPrologのfold/unfoldによって変換する．例えば，

```
    concat(A,[a],C),concat(C,[b],E)
⇒concat([a],[b],C),concat(A,C,E)      (規則(15))
⇒concat([],[b],Z),concat(A,[a | Z],E) (②unfold)
⇒concat(A,[a,b],E)                    (①fold)
```

また，

```
      X is ((5+Y)+10)*10
⇒     X is ((Y+5)+10)*10              (規則(8))
⇒     X is (Y+(5+10))*10              (規則(10))
⇒     X is (Y+15)*10
⇒     X is Y*10+15*10                 (規則(12))
⇒     X is Y*10+150
```

３．５　無用代入の削除

変数への単なる代入を取り除き，変数をその代入値で置き換える．例えば，

```
X = [] ,
Y is 10 ,
b(X,Y)
```

を

```
b([],10)
```

に変換する．

３．６　ＤＣＧのｆｏｌｄ／ｕｎｆｏｌｄ

DCG記述のfold/unfoldについては，効率を向上させる一般的な変換方法はまだ分かっていない．現在，PANDA処理系では，次のような特別な場合だけ，DCG記述のfold/unfold変換を行う．

```
    Z --> [],Z.                          ...①
    Z --> Z,[].                          ...②
empty--> [].                             ...③
```

例えば，

```
  empty,a(A,B),empty
⇒ [],a(A,B),[]                    (③unfold)
⇒ a(A,B),[]                       (①fold)
⇒ a(A,B)                          (②fold)
```

4. むすび

以上，PANDA処理系で行っているいくつかの最適化を紹介してきた．PL/0コンパイラのPANDA記述の場合，生成されたPrologプログラムから見る各最適化の効果は，図１１が示している．属性の削除と同値変数の統合は，Prologプログラムの効率化に著しい効果を上げている．

参考文献：

(1)M.Alan & A.O.Richard: “A Polymorphic Type System fir Prolog” ,Artif.Intell.２３(1984).

(2)R.M.Burstall & J.Darlington: “A Transformation System for Developing Recursive Programs” JACM, ２４, 1(1977).

(3)R.Farrow: “Generating a Pratical Compiler from an Attribute Grammar” ,IEEE Trans. software Eng. １,4,pp.77-93(1984).

(4)馮，杉山，鳥居：“属性文法の表記法について”，昭61年後期情処学全大，７Ｄ－4.

(5)D.E.Knuth: “Semantics of Context-free Languages” ,Math.Syst.Theory,２,2,pp.127-145(1968).

(6)中川，中村：“Prolog 等価変換エディタと変換戦略”，情処学論，２６，5,pp.905-912(1985).

(7)杉山，馮，藤井，鳥居，前野：“属性文法に基づく言語PANDAとDCGへの変換”，信学技報，ＳＳ８６-5.

(8)F.Pereira & D.Warren: “Definite Clause Grammar for Language Analysis” ,Artif.Intell. １３,pp.231-278(1980).

(9)N.Wirth: “Algorithm + Data structure = Programs” ,Prentice-Hall(1976).

		(1)	(2)	(3)	(4)	(5)	減少率(%)
属性（個）	660	361					45.3
長さ(行)	1762	1182	974	952	942	939	46.7
変数（個）	2384	2085	1814	1751	1747	1740	27.0
CONSULT:							
時間(s)	56.70	38.10	33.42	31.71	31.38	32.32	43.0
空間(bytes)	58624	47820	42456	42168	42028	41876	28.6
実行：							
時間減少(%)		8.3	9.0	6.5	2.7	1.9	25.6
空間減少(%)		12.0	7.2	0.4	0.15	0.05	20.0

ここで，

(1) 属性の削除

(2) 同値変数の統合・共通部分式の重ねあわせ

(3) 定数の畳み込み

(4) 無用代入の削除

(5) DCGのFOLD/UNFOLD

図１１