ソフトウェア基礎論　18−3
（１９８６・９・３０）

Ｄｅｆｉｎｉｔｅ　Ｃｌａｕｓｅ　Ｇｒａｐｈ　Ｇｒａｍｍａｒ：
論理プログラミングのためのグラフ文法

塩谷　勇　　中村　克彦
（東京電機大学　理工学部）

２次元パタンなどの解析，認識，表示に対して，グラフ文法はひとつの有力な基盤となりうるものと期待される．この報告では，論理プログラミングにもとづいたひとつのグラフ文法(Definite Clause Graph Grammar: DCGG)を提案し，その文法の能力と構文解析法について述べる．論理プログラミングに適合したより能力の大きなグラフ文法を定義するため，この文法はhypergraphにもとずいて定義される．Hypergraphにおいては，各辺(hyper-edge)がふたつ以上の接点をも接続できるように一般のグラフの辺が拡張されている．この文法の各生成規則は Horn節であり，hypergraph の辺を部分 hypergraph に書き換える規則を表し，グラフの導出は resolution によって行われる．文法によって導出されるは言語に対しては，文脈自由型言語に対するuvwxy定理に相当するPumping Lemmaと呼ばれる命題が成立し，この命題からこのグラフ文法の言語ではないいくつかのグラフのクラスが示される．構文解析に関しては，この文法をもとにして入力(hyper)グラフを認識するPrologプログラムに変換するための２，３の方法が示されている．

Definite Clause Graph Grammar: A Graph Grammar based on Logic Programming

Isamu SHIOYA and Katsuhiko NAKAMURA

Department of Systems Engineering, Tokyo Denki University, Hatoyama-machi, Saitama-ken, 350-03, Japan

The graph grammar can be considered to be an important base for analyzing recognizing, and displaying two-dimewnsional patterns. We propose a new class of graph grammars, that are called Definite Clause Graph Grammars(DCGG) and based on logic programming, and discuss their capabilities and the parsing methods. We employ hypergraphs instead of ordinary graphs for defining a more powerful graph grammar by which capabilities of the logic programming is fully utilized. Each production rule is a Horn clause, and represents rewriting of a hyper-edge to a sub-hypergraph. The derivation of DCGG is defined by the resolution. We proved a proposition called the pumping lemma which is corresponding to the "uvwxy theorem" for context-free languages, and by which several classes of graphs are shown not to be definite clause graph languages. Some methods are described for transforming a DCGG into a Prolog program that recognizes the input (hyper-)graphs.

1. はじめに

知識情報処理のいくつかの分野において，入力情報のもつ構造を調べ，これがあるクラスに属しているか否かを決定するプロセスが必要とされる．入力情報のもつ構造の解析は，この情報のもつ“意味”を把握するための基礎となる．自然言語の文に対する構文解析とこれにもとづく意味の解析および図形の構文的な解析[Fu 1982]は，このようなプロセスの代表的な例である．

グラフは，広範囲の情報に含まれる構造を表現する強力な方法を与えることから，一般的な情報の構造の解析を目的として，これまでに各種のグラフ文法が提案されている[Claus et al 1979, Ehrig et al 1983, Vigna and Ghezzi 1978,大川 1980]．これらのグラフ文法は記号列のクラス（言語）を定義するための形式文法[Aho and Ullman 1972]をグラフに拡張したものであり，各々の生成規則は部分グラフの書き換えによって定義される．

この報告では，論理プログラミングにもとづいたひとつのグラフ文法(Definite Clause Graph Grammar, DCGGと呼ぶ)を提案し，その文法の能力と構文解析法について述べる．この文法は，Kowalskiによる文脈自由型文法の規則を表現したHorn節からなり，その言語を認識する論理プログラム[Kowalski 1979]を拡張した形式をもっている．

従来の文脈自由型のグラフ文法では，各生成規則の左辺は単一の頂点または辺に制限されている．この制限を緩め，かつ，論理プログラムに適合したグラフ文法とするために，われわれはグラフを拡張したhypergraphにもとづいた形式化を行う．すなわち，この文法の各生成規則はHorn節によって表され，hypergraphの辺(hyper-edge)を部分hypergraphに書き換える規則を表す．この文法によるグラフの導出はresolutionにもとづいて行われ，実際の構文解析は文法をPrologプログラムに変換して行われる．

2. Hypergraph

われわれは，まずグラフのひとつの拡張であるhypergraph[Berge 1976]を定義する．Hypergraphは通常のグラフの辺を任意個の頂点を接続することができるようように拡張したhyper-edgeの集合からなる．ここでは，hyper-edgeは関数記号を含まない述語項（または，単位節）によって定義される．ここで取り扱うhypergraphは複数個の同一のhyper-edgeを含むことを許したmultiple hypergraphである．

[定義１] Directed labeled multiple hypergraph（以下，単に hypergraph と呼ぶ）G は，

$$p(a_1,a_2,\ldots,a_n), \qquad n \geq 1$$

の形式のhyper-edgeと呼ばれる述語項の有限のmultisetである．述語記号 p をこのhyper-edgeのラベル，各引数 a_i を頂点と呼ぶ．ただし，

(1)Multiset（または，bag）とは，複数個の同一の要素を含むことを許した集合である．Multisetの和(∪)，差(-)などが集合に対する概念を拡張して定義できる．

(2)ラベルに対する引数の個数は固定されており，ラベルから引数の個数は一意に定まる．

(3)引数は定数または変数であり，ともに頂点を表す．

Hypergraph G のすべての頂点の集合を V(G)，G のラベルの集合を C(G) によって表す．

一般のラベル付き有向グラフはhyper-edgeの引数の個数がすべて１または２であるhypergraphによって表される．このとき，hyper-edge p(a) は p が頂点 a のラベルを，また，p(a,b)は p が有向辺 (a,b) のラベルであることを表す．

[例１] 図１(a)のグラフは hypergraph {p(a,b), p(a,b), q(b,c)} によって表わされる．{p(a,b,c,d), q(c,d,e)}はhypergraphであり，３個以上の頂点をもったhyper-edgeを含んでいる．このようなhyper-edgeは頂点の順序を除いて，図１(b)のように表される．

[定義２] Hypergraph G, H に対して写像 $f:V(G) \rightarrow V(H)$ が存在して

$$\{p(f(a_1), \ldots, f(a_n)) \mid p(a_1, \ldots, a_n) \in G\} \subseteq H$$

が成立するとき，G は H に（頂点に関して）準同型であるといい，f を準同型写像と呼ぶ．G, H に対して，G からH への準同型写像 f が存在し，逆写像f^{-1}もまた準同型であるならば，G と H は同型であるといい，f を同型写像と呼ぶ．G, H に対して，G から H への全射でかつ同型でない準同型写像が存在するとき，H は G の縮退であるという．

われわれは，以下において，同型な hypergraphは同値であるとみなす．

[例２] Hypergraph G = {p(a,b), p(b,c)} に対して，H1 = {p(a,b), p(b,b)}, H2 = {p(a,a), p(a,b)}, H3 = {p(a,b), p(b,a)}, H4 = {p(a,a), p(a,a)}は G の縮退となっている（図２）．

3. Definite Clause Graph Grammar

[定義３] Definite Clause Graph Grammar(DCGG)は記号の組 (R,I)で表される．ここで，

(1) 論理プログラム R は次の形式の節の有限集合である．

$$P :- Q_1, Q_2, \ldots, Q_n. \qquad n \geq 1$$

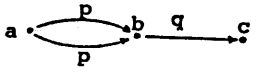
ただし，P, Q_i は関数記号を含まない（変数または定数）述語項である．

(2)初期hypergraph I は hypergraph である．

生成規則の左辺に現れる述語項のラベルを非終端ラベル，それ以外のラベルを終端ラベルと呼ぶ．

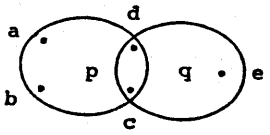
R 中の節は左側の句（hyper-edge）が右側のhypergraphから構成されていることを意味している．ただし，変数は通常の全称記号で導入される変数とは異なり，名前の異なる変数は相異なる頂点を表している．

つぎに，文法からのhypergraphの導出(derivation)を節のresolutionによって定義する．変数の集合 V から頂点を表す変数および定数の集合 W 上への部分関数 s を代入と呼ぶ．Hypergraph G の例(instance) Gsとは，すべての変数 $x \in V$ に対して x が s において定義されるならば，x を s(x) に置き換えて得られ



{p(a,b), p(a,b), q(b,c)}

図１(a)　グラフの例



{p(a,b,c,d), q(c,d,e)}

図１(b)　Hypergraphの例

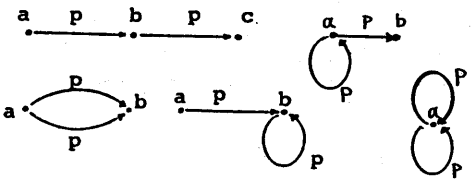
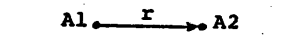
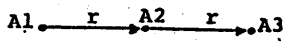


図２　Hypergraphの縮退した例





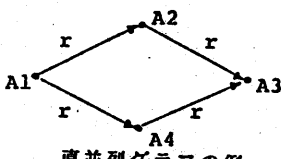


図３　直並列グラフの例

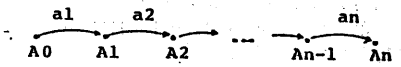
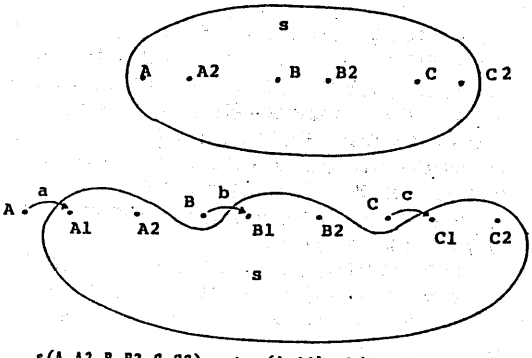


図４　記号列 a1 a2 ... anの線形グラフによる表現



s(A,A2,B,B2,C,C2) --> a(A,A1), b(B,B1), c(C,C1),
s(A1,A2,B1,B2,C1,C2)

図５　記号列 $a^n b^n c^n$ の生成規則

るhypergraphである．

［定義４］　文法 (R,I) に対して hypergraph G から H が導出されるとは，G の中のあるhyper-edge E と R の生成規則 P :- Q_1, ... ,Q_n および，E と P の統合化(unification)による代入 s が存在して，

$H = (G - \{E\})s \cup \{Q_1,\ldots,Q_n\}s$

が成立することである．ただし，P :- $Q_1,\ldots,Q_n$ 中の変数は G の中に現れない新しい変数に置き換えておくものとする．

DCGG D において，hypergraph G から H が導出されることを

$G \overset{}{\underset{D}{\Longrightarrow}} H$

と表す．また関係 $\overset{}{\underset{D}{\Longrightarrow}}$ の反射的推移的閉包を $\overset{*}{\underset{D}{\Longrightarrow}}$ と書く．

［定義５］　DCGG D = (R,I)に対して，$I \overset{*}{\underset{D}{\Longrightarrow}} G$ なるhypergraph G は (R,I) から導出されたという．D の言語(Definite Clause Graph Language,DCGL),L(D) はつぎのように定義される．

$L(D) = \{G \mid I \overset{*}{\underset{D}{\Longrightarrow}} G,\ C(G)$はすべて終端ラベル$\}$

また，ふたつのDCGG D, D'に対して，L(D)=L(D')が成立するならば，DとD'は等価であると呼ぶ．

［例３］　直並列グラフの文法は，(R,{i(A,B)})で表される．ここで，R はつぎの生成規則の集合である．

i(A,B) :- r(A,B)
i(A,B) :- i(A,B), i(A,B)　(並列接続)
i(A,C) :- i(A,B), i(B,C)　(直列接続)

この文法の導出するグラフの例を図３に示す．

つぎの補題は後の命題の証明に用いられる．

ある導出の途中のhypergraphからつぎに導出を適用するhyper-edge（目標）を選ぶ写像を計算規則と呼ぶ．このとき，つぎの補題は論理プログラムの計算の場合[Lloyd 84]と同様に証明できる．

［補題６］ DCGG D=(R,I)に対して，任意のcomputation rule Cによる導出 $I \overset{*}{\underset{D}{\Longrightarrow}} G \in L(D)$が存在するならば，他の任意のcomputation rule C'に対しても，C'による導出 $I \overset{*}{\underset{D}{\Longrightarrow}} G' \in L(D)$が存在して，GとG'は同型となる．

４．　ＤＣＧＧと形式文法

ここでは，DCGGと文字列のための文脈自由型文法の関係について述べる．

任意の記号列$a_1a_2\ldots a_n$は線形グラフ

$\{a_1(A_0,A_1),a_2(A_1,A_2),\ldots,a_n(A_{n-1},A_n)\}$,

で表すことができる．ここで，A_i は変数であり，$i \neq j$ ならば$A_i \neq A_j$ である．この線形グラフを図４に示す．同型な線形グラフをひとつのものと考えれば，記号列とこの線形グラフは１対１に対応する．

文脈自由型文法はつぎの方法によって等価なDCGGに変換できる．

(1)文脈自由型文法の生成規則

$a \dashrightarrow b_1\ b_2\ \ldots\ b_n \qquad n \geq 1$

はDCGGの生成規則

$a(A_0,A_n)$:- $b_1(A_0,A_1)$, ... ,$b_n(A_{n-1},A_n)$

に変換される．
(2)開始記号 s は hypergraph {s(A_0,A_1)}に変換する．
このとき，文脈自由型文法から導出される言語は記号列をhypergraphで表すことによってDCGGからも導出され，その逆も証明できる．したがって，任意の文脈自由型言語に対してこれを生成するDCGGが存在する．
この方法は，Kowalski[Kowalski 1979]が文脈自由言語の構文解析に用いたものと同じである．つぎに、この方法によってDCGGから導出可能な記号列の言語のクラスは文脈自由型ばかりでなく，文脈依存型の言語をも含むことを示す．

［命題７］文脈依存型言語 {$a^nb^nc^n$| n ≥ 1}を生成するDCGGが存在する．
［証明］D1 を DCGG (R,i(A,B))とする．ここで，R はつぎの生成規則である．

```
i(A,D) :- s(A,B,B,C,C,D)
s(A,A1,B,B1,C,C1) :-
        a(A,A1), b(B,B1), c(C,C1)
s(A,A2,B,B2,C,C2)  :-
        a(A,A1), b(B,B1), c(C,C1),
        s(A1,A2,B1,B2,C1,C2)
```

i と s は非終端ラベル，a，b，cは終端ラベルである．この文法は，線形グラフの形式で記号列 $a^nb^nc^n$ (n = 1,2,3,...)に対応するhypergraphを導出する（図５はこの記号列を表す線形グラフの構文を図示したものである）．{$a^nb^nc^n$|n≥1}とL(D)が等価であることは数学的帰納法より，容易に示すことができる．

５．　ＤＣＧＧの能力

DCGGの生成規則は形式文法のChomsky標準形に対応する形式に書き換え可能である．
［命題８］任意 DCGG はすべての生成規則がつぎの形式をした等価なDCGGに変換可能である．

P :- Q_1, Q_2　　または　　P :- B

ここで，P,Q_1,Q_2は非終端ラベルをもった述語項，Bは終端ラベルをもつ述語項である．
［証明］文法中に規則 P:-Q_1 があり，Q_1 が非終端記号をもつ文法は，この形式を含んでいない文法に変換することができるので，このような規則が出現しないと仮定する．つぎに，DCGG Dの生成規則

P :- Q_1, Q_2, ... ,Q_n　　n ≥ 3

とするとき，この規則をふたつの生成規則

P :- Q_1, R　　(5-1)
R :- Q_2, ... , Q_n　　(5-2)

に置き換えた文法を D' とする．ただし，Rは文法中に現れない非終端ラベルをもち，Q_2,...,Q_nに出現するすべての変数を引数にする述語項である．文法D'の導出において生成規則(5-1)を適用した後には必ず(5-2)を適用する計算規則を用いるならば，命題６より，L(D)=L(D')となる．同様の操作を生成規則の右辺の述語記号がふたつ以下になるまで繰り返す．つぎに，変換した生成規則の右辺に述語項がふたつ存在し，この中に終端ラベルをもつ述語項があるならば，その述語項について上で述べた方法をn=2の場合にも適用する．

DCGGの能力に関してつぎの基本的な命題が成立する．

［命題９］(Pumping Lemma) D を任意のDCGGとし，L(D)をその言語とする．D に対してある定数pが存在して，hypergraph G を導出して，|G|>pを満たすならば，Gはふたつのhypergraph A およびB_0に分割でき，各整数k≥0に対して，hypergraph　A U B_0 U B_1 U ... U B_kもまたL(D)に含まれる．ここで，すべての B_j(0≤j≤k)は互いに同型である．

この命題の証明は文脈自由型文法に対するuvwxy定理と類似した方法で証明される．証明を付録Ａに示す．この命題を用いて，あるグラフのクラスがDCGLでないことを証明できる．

［命題９の系］Hypergraphのクラス{G_i,i=0,1,...}に対して，$|G_i| = m^i$,i=0,1,2,...(m≥2)を満足するクラスは，DCGGによって導出できない．

従って，つぎのようなDCGGで導出できないグラフが存在する．
［例４］図６で示すようなグラフ，縦と横の辺の数が同じすべてのグラフのクラスはDCGGによって導出できない．

６．　論理プログラミングによる構文解析

ここでは，DCGGの生成規則を論理プログラム，または，Prologのプログラムに変換して，この計算によってDCGLのtopdown方式の構文解析を行う方法について述べる．われわれは論理プログラミングを用いた構文解析の問題点をつぎの命題に示す．
［命題１０］　D=(R,{l_1,l_2,...,l_n})をDCGGとし，G はhypergraphが変数の含まない単位節で表した集合とする．プログラム R U G と質問 ?- l_1,l_2,...,l_n.の計算が成功するための必要十分条件は D からの導出によって得られるhypergraphの例(instance)がGのある部分集合であることである．
［証明］論理プログラミングの性質と［命題６］から容易に示すことができる．

正しく構文解析が行われるためには，プログラムR U G と質問 ?- l_1,...,l_n. による計算が導出されたhypergraphと同型であるときのみ成功する必要がある．従って，DCGG の実際の構文解析においてはつぎのようなふたつの問題を解決する必要がある．
(1) Gの部分hypergraphが文法の言語に含まれるならば G もこの計算によっ成功してしまう．
(2) G が文法から導出されるhypergraphの縮退である場合にも計算が成功する．
このほかの問題として，上の計算が成功する場合でも，実際のPrologの計算では無限ループに入ってしまうことがある．これについては，後の６．３節で述べる．
［例５］DCGG がただひとつの節からなる論理プログラム{q(A,C) :- p(A,B), p(B,C)}と初期hypergraph {q(A,C)}からなるとすると，hypergraph {p(a,b), p(b,c)}のほかに縮退したhypergraph {p(a,b), p(b,a)}，や{p(a,b), p(b,b)}などが計算によって成

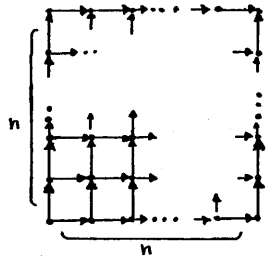


図６　DCGGで導出できないグラフ

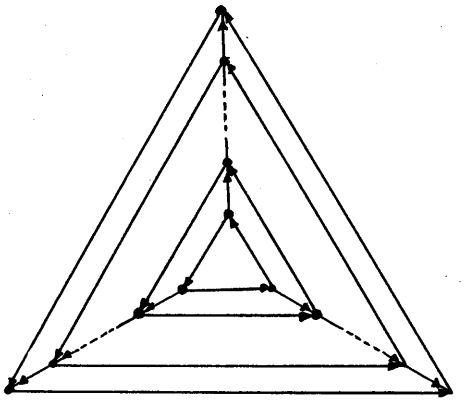


図７(a)　例６の文法の導出するグラフ

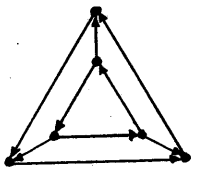


図７(b)　解析するグラフ

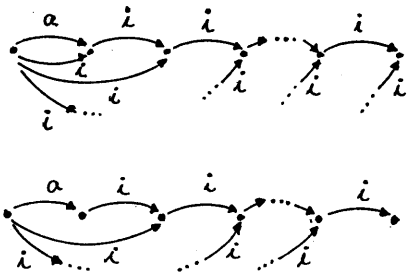


図８　直並列グラフのパタン

功する（図１）．

６．１　論理プログラミングによる構文解析

上の(1)の問題点を解決するためひとつの方法について述べる．まず，hyper-edgeの集合をリストで表し，DCGGの生成規則

$p(A_1,\ldots,A_m)$:-$q_1(B_{11}, \ldots ,B_{1n_k}),\ldots,$
$q_k(B_{k1}, \ldots ,B_{kn_k})$　m, n, $k \geq 1$　　(6-1)

に対して，つぎのようにふたつの変数の引数を生成規則の各辺の述語項に追加し，頂点をすべて変数とした節に書き換える．

$p(P_0,P_k,A_1, \ldots ,A_m)$:-
$q_1(P_0,P_1,B_{11}, \ldots ,B_{1n_k}), \ldots ,$
$q_k(P_{k-1},P_k,B_{k1}, \ldots ,B_{kn_k})$.　　(6-2)

ここで，p, q_i は述語，P_i, A_i, B_{ij}は変数である．ただし，終端ラベルを持ったhyper-edge $e(A_1, \ldots ,A_j)$が生成規則の中に存在するならば，これをつぎのような述語項に変換する．

$sub(P_{i-1},e(A_1, \ldots ,A_j),P_i)$

左辺の$P_0,P_1,\ldots,P_n$にはhyper-edgeの集合を表すリストが代入される．述語項 $p(P_i,P_j,A_1,\ldots,A_n)$は$P_i$の表すリストと$P_j$の表すリストの差が入力hypergraphのhyper-edge $p(A_1,\ldots,A_n)$であることを表している．

述語subはつぎの節から定義される．

```
sub([X|A],X,A).
sub([A|B],X,[A|Y]) :- sub(B,X,Y).
```

述語sub(A,B,C)は「リストによって表された集合 A から要素 B を取り除いた集合を表すリストが C である」ことを意味する．

以上の変換によって，hypergraphをhyper-edgeのリスト$[e_1,\ldots,e_k]$によって表し，$a_1,\ldots,a_n$を初期hypergraphの頂点を定数としたとき，質問

?- $i([e_1,\ldots,e_k],[],a_1,\ldots,a_n)$.

によって，hypergraphが文法から導出されるhypergraphの縮退のとき，成功する．

［例６］図７(a)のようなグラフの集合を表す文法はつぎのようなHorn節からなる生成規則の集合

```
q(A,B,C) :- r(A,B), r(B,C), r(C,A)
p(A,B,C) :- q(A,B,C)
p(A,B,C) :- q(A,B,C), r(A,D),
            r(B,E), r(C,F), p(D,E,F)
```

と初期hypergraph

{p(A,B,C)}

によってに表される．この文法の生成規則はつぎのように変換される．

```
q(P0,P3,A,B,C) :- sub(P0,r(A,B),P1),
      sub(P1,r(B,C),P2), b(P2,r(C,A),P3).
p(P0,P1,A,B,C) :- q(P0,P1,A,B,C).
p(P0,P5,A,B,C) :- q(P0,P1,A,B,C),
      sub(P1,r(A,D),P2), sub(P2,r(B,E),P3),
      sub(P3,r(C,F),P4), p(P4,P5,D,E,F).
```

図７(b)のグラフがこのグラフ文法を満足していることをつぎの質問によって確かめることができる．

```
?- p([r(a,b),r(b,c),r(b,c)],a,b,c).
```

この方法によって問題点(1)は解決するが，問題点(2)は解決されず，縮退したhypergraphも受理されてしまうので，プログラムの節に縮退したhypergraphを受理しないような条件を加えることが必要となる．

６．２　縮退したhypergraphを認識しない構文解析法

［第１の方法］６．１節で述べた生成規則からHorn節への変換に加えて，各節につぎの操作を施す．

(1)非終端ラベルをもつ各々の辺 $p(A_1,\ldots,A_n)$ に対応する述語項に，この辺が含む $A_1,\ldots,A_n$ 以外の頂点（副頂点と呼ぶ）の集合を表すリストのための引数を追加する．

(2)ふたつ以上の目標を含む各節に対して，これらの目標に対応する辺の副頂点の集合が互いに素であり，それらの和集合が節の左辺の副頂点のためのリストであることを意味する目標 disj(L1,L2) および union(L1,L2,L3) を追加する．

この方法で例５の文法を変換したプログラムをつぎに示す．

```
q(P0,P3,[],A,B,C) :- sub(P0,r(A,B),P1),
  sub(P1,r(B,C),P2), sub(P2,r(C,A),P3).
p(P0,P1,L0,A,B,C) :- q(P0,P1,L1,A,B,C).
p(P0,P2,L0,A,B,C) :- q(P0,P1,L1,A,B,C),
  sub(P1,r(A,D),P2), sub(P2,r(B,E),P3),
  sub(P3,r(C,F),P4), p(P4,P5,L2,D,E,F),
  union(L1,L2,L0), disj(L1,L2).
```

述語union(A,B,C)はリストA，Bで表される集合の和のリストがCであることを，述語disj(A,B)はリストAとBの中に共通の要素がないことを意味している．

構文解析のための質問をつぎに示す．

```
?- p([r(a,C),r(C,b),r(C,b)],[],_,a,b,c).
```

［第２の方法］第２の方法では各生成規則に定数がなく，左辺に同じ変数が２回以上現れないと仮定する．各生成規則をつぎのような手順で変換する．

(1)上と同様に生成規則(6-1)を上の(6-2)の形式の節に変換する．

(2)生成規則の左辺に現れず，右辺にのみ現れる頂点（変数）X_1, X_2, ..., X_nが含まれる場合には，特別が述語を用いた目標 generate(X_1, X_2, ..., X_n) を変換した節の右辺の先頭に追加する．目標 generate(X_1,X_2,...,X_n) は述語が呼び出されるたびに各変数 X_iにそれぞれ一意的な定数を発生して代入を行う．

(3)構文解析されるhyper-edgeのリストによって表したhypergraphのすべての頂点は定数でなく変数に置き換える．質問中に現れる頂点はすべて異なった定数を代入するか，または述語generateによって発生する．

述語 generate はつぎのように定義される．

```
generate(N):-uqname(X),
  retract(uqname(_)), N is X+1,
  assert(uqname(N)).
uqname(0).
```

［例７］例６の文法に対する縮退したグラフを認識しないためのプログラムはつぎのようになる．

```
q(P0,P3,A,B,C) :- sub(P0,r(A,B),P1),
   sub(P1,r(B,C),P2),b(P2,r(C,A),P3).
p(P0,P1,A,B,C) :- q(P0,P1,A,B,C).
p(P0,P5,A,B,C) :- generate(D),
  generate(E), generate(F),
  q(P0,P1,A,B,C), sub(P1,r(A,D),P2),
  sub(P2,r(B,E),P3), sub(P3,r(C,F),P4),
  p(P4,P5,D,E,F).
```

図７(b)のグラフの構文解析はつぎの質問によって実行される．

```
?- p([r(a,C),r(C,b),r(C,b)],[],a,b,c).
```

ここで， a, b, c は定数，Cは変数である．

６．３　Prologにおける構文解析におけるループの回避

Prologシステムを用いたtopdownの構文解析においては，一般の論理プログラムの計算が成功する場合でも無限ループに入ってしまうという問題が生ずる．生成規則の右辺に非終端ラベルをもつhyper-edgeが必ず存在する（形式文法のGreibach標準型に相当）文法への変換が可能ならば，この問題は解決する．例３で述べた直並列グラフに対しては，このような変換ができる．

直並列グラフの文法の規則はつぎのものからなる．

```
i(A,B) :- a(A,B)                (6-3)
i(A,B) :- i(A,B), i(A,B)        (6-4)
i(A,C) :- i(A,B), i(B,C)        (6-5)
```

(6-3)を(6-4),(6-5)に代入して得られる生成規則をグループ (A)

```
i(A,B) :- a(A,B)
i(A,B) :- a(A,B), i(A,B)
i(A,C) :- a(A,B), i(B,C)
```

と，それに加えて，グループ (B)

```
i(A,B) :- i(A,B), i(A,B)
i(A,C) :- i(A,B), i(B,C)
```

からなる生成規則を合わせた文法としても，ふたつの文法は等価な言語を生成する．このようにグループ(A)の生成規則をグループ(B)の規則の右辺の左に代入し，得られた規則をグループ(A)に加える．このようにして得られる(A)の規則の一部は(B)の規則を逆に適用することによって，簡単化できる．この操作をつづけると，図８に示したパタンに関する規則のみが，簡単化できない規則として残る．これは別の規則によって表すことができ，直並列グラグの文法はつぎのようになる．

```
i(X,Y) :- a(X,Y)
i(X,Y) :- a(X,Y), i(X,Y)
i(X,Z) :- a(X,Y), i(Y,Z)
i(X,Z) :- a(X,Y), z1(X,Y,Z)
i(X,A) :- a(X,Y), z1(X,Y,Z),i(Z,A)
i(X,Z) :- a(X,Y), z2(X,Y,Z)
i(X,Z) :- a(X,Y), z2(X,Y,Z),i(X,Z)
z1(X,Y,Z) :- i(X,Z), i(Y,Z)
z1(X,Y,Z) :- i(X,A), i(Y,A), z1(X,A,Z)
z2(X,Y,Z) :- i(X,Y), i(Y,Z)
z2(X,Y,Z) :- i(X,Y), i(Y,A), z2(X,A,Z)
```

この文法をもとに，６．１節で述べた変換を適用して得られた直並列グラフ解析用プログラムを付録Bに示す．

７．　むすび

この報告では，論理プログラミングにもとづくグラフ文法を提案し，その能力と実際の構文解析について議論した．Hypergraphの導入によって，この文法は論理プログラムのもつ演繹の能力を生かしたものとなっていることが特長である．DCGGの能力については，つぎのような重要な問題が残されており，現在これらに

ついて検討を進めている．

(1)Multisetではなく，一般の集合で定義されるhypergraphに対する文法の能力との比較

(2)この文法の生成規則の左辺がひとつのhyperedgeでなく，部分hypergraphであるような文脈依存型の文法に拡張する方法

構文解析については，論理プログラミングによるtopdown方式による方法を述べた．ここでは，hyperedgeの集合を線形リストによって表したが，これを（リストまたは項による）二進探索木で表したり，述語subの代わりに完全に非決定的にassertを行う組み込み述語によって処理効率を向上することができる．また，実際のPrologの処理では無限ループの問題も重要となる．これを解決するひとつの方法としてbottomupによる方法について検討も加える必要がある．

われわれはこの文法を図形の認識および図形用の構文エディタに応用する予定である．実際のグラフの構文解析においては処理速度が大きな問題点となるが，図形処理に対する応用においては頂点の位置関係による探索の制御によって処理効率の向上を図ることが可能である．

［文献］

[Aho 72] Aho, A. and Ullman, J.(1972), The Theory of Parsing, Translation and Compiling I, Prentice-Hall.

[Berge 76] Berge(1976), C., Graphs and Hypergraphs, North-Holland.

[Claus 79] Claus, V., Ehrig, H. and Rozenberg, G. (ed.) (1979) Lecture Notes in Computer Science 73, Springer.

[Ehrig 83] Ehrig, H., Nagl, M. and Rozenberg, G.(ed.)(1983) Lecture Notes in Computer Science 153, Springer.

[Fu 82] Fu, K. S.(1982) Syntactic Pattern Recoginition and Applications, Prentice-hall.

[Kowalski 79] Kowalski, R.(1979) Logic for Problem Solving, Elsevier North Holland.

[Lloyd 84] Lloyd, J. W.(1984) Foundations of Logic Programming, Springer-Verlag.

[Pereira 80] Pereira, F. and Warren, H.(1980) Definite Clause Grammar for Language Analysis, Artificial Intelligence 13.

[Pereira 86] Pereira, F. C. N.(1986)Can Drawing Be Liberated from the von Neumann Style?, Logic Programming and its Applications(ed. M. V. Caneghem and D.H.D. Warren), Ablex Publishing co.

[Vigna 78] Vigna, P. and Ghezzi, C.(1978) Context-free Graph Grammar, Information and Control, Vol. 37.

[大川 80]大川，那須，西谷(1980) k-グラフ生成文法，信学会 Vol. 63-D, No. 9.

[松本 83]松本(1983)Prologに埋め込まれたボトムアップパーサ：BUP, Proc. of The Logic Programming Conf. '83,3.1

[塩谷 86]塩谷,中村(1986) Definite Clause Graph Grammar，情報処理学会第３２回全国大会．

-[塩谷 86]中村，塩谷(1986) DCGG:論理プログラミングのためのグラフ文法，情報処理第３３回全国大会．

[塩谷 86]塩谷，中村(1986) DCGG(Definite Clause Graph Grammar)の構文解析方式，情報処理３３回全国大会．

付録　A　　命題９の証明

この証明においては，ふたつのhypergraph G, H が同型であるとは，上で述べた同型の条件に加えて，同型写像 f に対して，任意の定数の頂点 $a \in V(G)$に対して，$f(a)=a$，任意の定数の頂点 $b \in V(H)$ に対して，$f^{-1}(b)=b$ 成立すると仮定する．生成規則はすべて命題８の形式に変換されていると仮定する．

最初に，生成規則と初期hypergraph中には定数が存在せず，生成規則の左辺の引数には，同じ変数があらわれない場合について示す．

文法の中のラベルの数が m，述語記号の引数の数の最大をnとし，$p=2^{m \cdot n!}+1$とする．規則は命題８の形式の規則をもつ文法に変換されていると仮定すると，経路の長さがj以下ならば，導出の長さは$2^{m \cdot n!}-1$以下である．したがって任意のG　L(D)に対して$|G|>p$ならば，Gのどの導出木も$2^{m \cdot n!}+1$以上の経路を含んでいる．このとき最長の経路は，ふたつの同型なhyper-edge q,q'を含む．

$$I \overset{*}{\Rightarrow}_D A_0 \cup \{q\} \overset{*}{\Rightarrow}_D A_0 \cup B_0 \cup \{q'\} \overset{*}{\Rightarrow}_D A_0 \cup B_0 \cup A_1$$

ここで，A_0は I から導出されたグラフからqを除いたもの，B_0はqから導出したグラフからq'を除いたもの，A1はq'から導出されたグラフである．qとq'は同型であり，規則中に定数がなく，規則の左辺に同じ変数がないので，q'に対してもqから$B_0 \cup \{q'\}$の導出を行った同じ規則の適用列を任意の回数おこなうことができる．したがって，

$$I \overset{*}{\Rightarrow}_D A_0 \cup B_0 \cup B_1 \cup \ldots \cup B_m \cup A_1'$$

であるような導出が存在する．ここで
$B_i, i=0,1,2,\ldots$は互いに同型となる．

つぎに一般の生成規則の場合について考える．各導出に対して，ある代入　$s_i (i=1,2,\ldots)$が存在して

$$I \overset{*}{\Rightarrow}_D G_1 s_1 \overset{*}{\Rightarrow}_D G_2 s_2 \overset{*}{\Rightarrow}_D \cdots \overset{*}{\Rightarrow}_D G_n s_n$$

が導出されたと仮定する．このような導出に対して，

$$I s_1 \ldots s_n \overset{*}{\Rightarrow}_D G_1 s_1 \ldots s_n \overset{*}{\Rightarrow}_D \cdots \overset{*}{\Rightarrow}_D G_n s_1 \ldots s_n$$

が成立する．

このとき，ある整数 i, j (i<j)とhyper-edge p, p'が存在して，つぎに示す３つのhyper-edgeが同型となる．

$$p\ s_1 \ldots s_i,\quad p\ s_1 \ldots s_j,\quad p'\ s_1 \ldots s_j$$

したがって，iからj番目までの導出をおこなうのに用いた規則を繰り返し適用すると同型な部分hypergraphが得られる．

付録　B　　直並列グラフ解析用Prologプログラム

```
i(P0,P1,X,Y) :- sub(P0,a(X,Y),P1).
i(P0,P2,X,Y) :- sub(P0,a(X,Y),P1), i(P1,P2,X,Y).
i(P0,P2,X,Z) :- sub(P0,a(X,Y),P1), i(P1,P2,Y,Z).
```

```
i(P0,P2,X,Z) :- sub(P0,a(X,Y),P1),
            z1(P1,P2,X,Y,Z).
i(P0,P3,X,A) :- sub(P0,a(X,Y),P1),
            z1(P1,P2,X,Y,Z),i(P2,P3,Z,A).
i(P0,P2,X,Z) :- sub(P0,a(X,Y),P1),
           z2(P1,P2,X,Y,Z).
i(P0,P3,X,Z) :- sub(P0,a(X,Y),P1),
       z2(P1,P2,X,Y,Z),i(P2,P3,X,Z).
z1(P0,P2,X,Y,Z) :- i(P0,P1,X,Z), i(P1,P2,Y,Z).
z1(P0,P3,X,Y,Z) :- i(P0,P1,X,A), i(P1,P2,Y,A),
             z1(P2,P3,X,A,Z).
z2(P0,P2,X,Y,Z) :- i(P0,P1,X,Y), i(P1,P2,Y,Z).
z2(P0,P3,X,Y,Z) :- i(P0,P1,X,Y), i(P1,P2,Y,A),
             z2(P2,P3,X,A,Z).
```