自然言語処理 57－1
（1986. 9. 19）

ＤＣＳＧ：差集合に基づくＤＣＧについて

田　中　卓　史
国立国語研究所

語順を持たない集合型言語のためのＤＣＧ "ＤＣＳＧ" について述べる．これは通常のＤＣＧが語の部分列を示すのに差分リストを用いるところを差集合で置き換えたものである．ＤＣＳＧの性質を利用すると，Prolog のプログラミングにおいて陥り易いある種の無限ループの問題を，一般化された構文解析の問題に帰着させて解決することができる．

次に，この性質に関連してＤＣＳＧの機能を拡張する．文法規則は集合の変換規則としてみなされ，逆変換のためのオペレータが導入される．このオペレータは下降解析の過程に処理対象の書き換え操作を加えるもので，部分的に上昇解析を行うことに相当する．この機能を用いると，下降解析で陥る別種の無限ループの問題を解決することができる．拡張されたＤＣＳＧによる構文解析はルールの適用が下降型に制御されたプロダクションの過程としてみることができる．

DCSG : Definite Clause Grammars Based on Difference Set

Takushi TANAKA

The National Language Research Institute
3-9-14 Nishigaoka, Kita-ku, Tokyo, 115 Japan

We present DCSG, Definite Clause Set Grammars, a DCG-like formalism for free word order languages. In contrast with the usual DCG formalism, DCSG is based on difference sets instead of difference lists. By dealing with DCSG as a generalized parsing problem, we avoid a certain type of Prolog infinite loop. Next we extend DCSG by introducing an operator for bottom up analysis. Using this operator solves the problem of infinite loops in top-down parsing. Parsing with DCSG with this operator can be viewed as the top-down controlled firing of production rules.

1．はじめに

語順を持たない言語を考えると，文は終単端記号の集合となる．非終端記号は部分集合に与えた名前に相当する．集合型の言語を解析するＤＣＧについて述べる．

ＤＣＧは文脈自由文法を述語論理式（確定節）で表したものである[2]．ＤＣＧの方法は一般に規則の形をとるものが適当な論理式に変換できるならば，その規則の適用を論理式の証明の過程で置き換えられることを示唆している．そこで回路の構成規則から論理式への変換を行う方法が提案された[5]．これはＤＣＧの文法・論理式変換メカニズムを語順を持たない集合型の言語に拡張したもので，通常のＤＣＧが語の部分列を示すのに差分リストを用いるところを，差集合で置き換えたものである．差集合に基づくＤＣＧは単に回路の解析にとどまらず，多くの問題に適用することができる．

初めに差集合に基づくＤＣＧの性質を明らかにする．この性質を利用すると，Prologのプログラミングにおいて陥り易いある種の無限ループの問題を容易に解決することができる．次に，この性質に関連してＤＣＧの機能を拡張する方法について述べる．文法規則は集合の変換規則としてみなされ，逆変換のためのオペレータが導入される．このオペレータは下降解析の過程に処理対象の変形操作を加えるもので，部分的に上昇解析を行うことに相当する．この機能を用いると下降解析が陥る別種の無限ループの問題を解決することができる．拡張されたＤＣＧによる構文解析はルールの適用が下降型に制御されたプロダクションの過程としてみることができる．この文法を用いて書かれたプロダクション・システムはバックトラックにより別解を得ることができる．

2．集合型言語のＤＣＧ

2．1　集合型言語

文脈自由文法の生成規則

```
A --> B1,B2,...,Bn.        (n ≧ 1)
```

（Ａは非終端記号，Bi (i=1,...,n)は終端記号あるいは非終端記号を表す．

において，右辺を記号列ではなく，記号の集合を表すものとして解釈すると，文脈自由の集合型言語を定義できる．すなわち，この規則は記号Ａから記号の集合 {B1,B2,...,Bn} を生成する．この集合型言語は次の差集合に基づくＤＣＧにより解析できる．

2．2　差集合に基づくＤＣＧ[3]

差集合に基づくＤＣＧにおいて非終端記号から終端記号を生成する規則

```
noun --> [book].
```

は次の Prolog の節に変換される．終端記号には述語memrest (member and the rest) が用いられ，非終端記号には述語 subset が用いられる．

```
subset(noun,S0,S1) :-
        memrest(book,S0,S1).          (1)
```

S0, S1 はリストにより表された集合である．ここで述語memrest は (2)と (3)で定義され，

```
memrest(M,[M|R],R).                   (2)
memrest(M,[A|X],[A|Y]) :-
        memrest(M,X,Y).               (3)
```

次のような形で用いられる．

memrest(集合の要素，集合，残り)．

一方，述語 subset はリストの要素の部分集合を言及するもので，第一項は第二項の集合の部分集合に与えた名前となる．第三項はその部分集合を除去した残りとなる．すなわち，述語memrest も subset も第一項は第二項と第三項の集合の差に相当する．

subset(部分集合の名前，集合，残り)．

3．差集合に基づくＤＣＧの性質

3．1　ループの問題

差集合に基づくＤＣＧの性質を明らかにするため次の問題を考えよう．ある回路で図１

のように電圧のデータが与えられていたとする．これを述語voltage を用いて次の論理式で表そう．

```
voltage($1,$2,20).
voltage($3,$2,15).        (4)
voltage($3,$4,8).
```

"voltage($1,$2,20)" は節点 $1 と節点 $2 の間の電圧が 20 voltであることを述べている．ただし第一項を電圧の正の方向とする．これらの論理式が Prolog のデータベースに加えられているとき，節点の順序に関係なく電圧のデータを検索するには，次の述語 volt を用いればよい．

```
volt(A,B,V) :- voltage(A,B,V).
volt(A,B,-V) :- voltage(B,A,V).   (5)
```

さらに，任意の二点 A，C 間の電圧を求める述語 vを定義するため，次の節を与えたとする．

```
v(A,C,V) :- volt(A,C,V).                (6)
v(A,C,V+W) :- volt(A,B,V),
              v(B,C,W).                 (7)
```

節(6) は節点 A，C 間に電圧のデータが与えられている場合に適用される．節(7) は電圧のデータがない場合にデータのある節点 A と中点B の間の電圧 Vを求め，さらに中点 B と節点C の間の電圧W を加えればよいことを定義している．しかし，この定義はうまく働かない．節点 $1，$4 間の電圧を求めようとして次のゴールを実行すると，

```
?- v($1,$4,X).
```

節点 $1，$4 間の電圧を与えるデータが存在しないので，(7) が適用される．(7) の最初のサブゴールは volt($1,$2,20)で成功し，中点B は節点 $2 が仮定される．次のサブゴール v($2,$4,W) も(6) が適用できず，(7) が適用されサブゴールに展開される．第一のサブゴールは volt($2,$1,-20)で成功し，第二のサブゴールは最初のゴールと同じになってループに陥る．このような問題を避けるには，同じデータが二度使われないように使ったデータを削除すればよい．節(5) を(5)'で置き換えればよいが，この方法はバックトラックにより別解を求めるようなことができない．

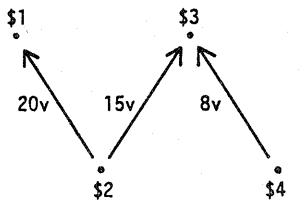

図 1　ある回路の電圧

Fig.1　Voltages on a certain circuit

```
volt(A,B,V) :-
        voltage(A,B,V),
        retract(voltage(A,B,V)).
volt(A,B,-V) :-
        voltage(B,A,V),            (5)'
        retract(voltage(B,A,V)).
```

別の方法でよく用いられるのは，使ったデータの軌跡を残す方法である．次の規則はこの方法で中点B の選択条件を与えている．節点 $1，$4 の間の電圧を求めるには，ゴール v($1,$4,X,[]) を証明すればよい．

```
v(A,C,V,_) :- volt(A,C,V).                (6)'
v(A,C,V+W,T) :- volt(A,B,V),
                not member(B,T),
                v(B,C,W,[A|T]).           (7)'
```

3．2　構文解析としてとらえた問題

電圧データの記述に用いた"voltage" を述語記号ではなく関数記号として用いると，(4) に相当する情報は複合項のリスト(8) で表すことができる．

```
[voltage($1,$2,20),voltage($3,$2,15),
 voltage($3,$4,8)]                        (8)
```

関数としての"voltage($1,$2,20)" は項 $1,$2,20 の間の関係を述べるのではなく，単に項 $1,$2,20 からなる複合項を構成している．このリストをある回路の電圧の状態を

記述した文であると考え，各々の複合項を単語であると考えよう．普通の文と異なり語順は重要ではなく，文は単なる語の集合となっている．実質的な情報は語の構造の中に含まれている．任意の節点間の電圧を求める問題は適当な文法規則を定めることで構文解析の問題に帰着することができる．節(5)に対応して次の文法規則を与える．

```
volt(A,B,V) --> [voltage(A,B,V)].
volt(A,B,-V) --> [voltage(B,A,V)].        (9)
```

ここで，"[" と"]" とで囲まれた voltage(A,B,V) は二点間の電圧を記述する終端記号である．一方，"volt(A,B,V)" は電圧記述文の中に存在しない非終端記号である．通常の文法規則と異なり変数を含んでいる．差集合に基づくＤＣＧの変換メカニズムにより文法規則(9)は Prolog の節(9)'となる．

```
subset(volt(A,B,V),S0,S1) :-
  memrest(voltage(A,B,V),S0,S1).
subset(volt(A,B,-V),S0,S1) :-              (9)'
  memrest(voltage(B,A,V),S0,S1).
```

電圧記述文(8)の中に非終端記号 volt(A,B,X)を同定することは，変数 S0 に(8)を代入してゴール

```
?- subset(volt(A,B,X),S0,S1).
```

を実行することである．変数A,B,X は節(5)の対応する変数と同じように振る舞う．異なるのは，同定に用いられた終端記号が除去されて，電圧記述文の残りの部分が変数 S1 から得られることである．

任意の二点間の電圧を求めるため(6),(7)に対応して次の文法規則を与えると，

```
v(A,C,V) --> volt(A,C,V).                  (10)
v(A,C,V+W) --> volt(A,B,V),
                 v(B,C,W).                 (11)
```

次の Prolog の節に変換される．

```
subset(v(A,C,V),S0,S1) :-
  subset(voltage(A,C,V),S0,S1).            (10)'
subset(v(A,C,V+W),S0,S2) :-
  subset(volt(A,B,V),S0,S1),
  subset(v(B,C,W),S1,S2).                  (11)'
```

任意の二点 A，C 間の電圧X を求めるには文(8)の中に非終端記号 v(A,C,X) を同定すればよい．同定に用いられた終端記号は順次電圧記述文から削除されるので，同じデータが何度も用いられ，無限ループに陥るようなことは起こらない．すなわち，Prologのプログラミングにおいて，同一のデータを多重に用いることで陥る無限ループの問題を，語順を持たない文としてデータを扱い，構文解析の問題として":-"記号を"-->" 記号に書き替えることで容易に解決できたのである．この方法は(5)'の方法と似ているが削除されたデータがバックトラックの際に復元される点が異なっている．従って，(6)'，(7)'の方法のように定義を複雑にすることなしに別解を枚挙することができる．なお，プログラムを文法規則へ書き替える際に終端記号となるものは "[" と "]" とで囲むことを注意せねばならない．

差集合に基づくＤＣＧを後向き推論規則として見たとき，推論に一度使ったデータを二度，使わないという性質は我々の暗黙の仮定とよくなじむ．Prologの ":-" 記号の代わりにこの文法の "-->" 記号を用いることで，一筆書きの問題，国電で赤羽から品川まで行く道順を枚挙する問題などを容易に解決することができる．

4．文法規則の別の側面

4．1　集合の変換

文法規則において [voltage(A,B,V)] で表される終端記号は述語 memrest(voltage(A,B,V),S0,S1) に変換された．処理対象となる集合は変数 S0 に代入され，その集合から要素 voltage(A,B,V) が除去されて変数 S1 から出力された．従って，集合の方に着目すると終端記号は与えられた集合から特定の要素を除去する変換を表している（図２）．一方，非終端記号は最終的に

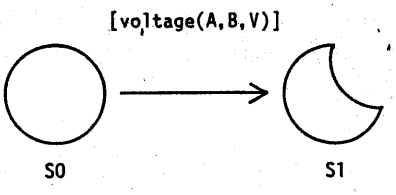

図２　終端記号による集合の変換

Fig.2　Set conversion by terminal symbol

終端記号を組み合わせて定義されるので，与えられた集合から非終端記号に相当する部分集合を除去する変換を表している．すなわち文法規則は終端記号や非終端記号で表される集合の変換の関係を定義するものとして見ることができる．

ここで逆変換の問題を考えよう．終端記号で表される特定の要素を除去する変換の逆変換はその要素を集合に加えることである．非終端記号で表される部分集合の除去変換の逆変換はその部分集合を加えることである．逆変換を実現するには，変数が入出力の関係に関して双方向的であることを利用して，述語 memrest や subset における入出力の関係を取り替えればよい．すなわち，第三項を集合の入力に，第二項を出力にとればよい．しかし，要素の除去変換の逆変換を述語 memrest を用いて次のように行うと，

```
memrest(m,OUT,[a,b,c]).
```

[m,a,b,c], [a,m,b,c], [a,b,m,c], [a,b,c,m]の場合が生じる．従って，集合に要素を追加する目的で述語memrest を用いるのは機能が大きすぎることになる．一方，非終端記号による変換の逆変換も単に述語 subset の入出力の関係を取り替えたのでは具合の悪い問題が生ずる．文法規則は解析に使うことを想定して定義されているので，入出力の関係を逆転して生成に使うと，規則 (11)'のＢのように値が定まらずに変数のまま残ることがある．この変数は存在的に解釈すれば良いが，全称的に解釈してしまうと節点Ｂは任意の節点と接続されていることになる．

４．２　ａｄｄオペレータ

集合の変換を考えるうえで，解析用に定義した文法規則をそのまま生成に用いることには無理がある．そこで，逆変換としては要素の追加変換のみを考えることにする．節点 $5 と $6 の間の電圧が 10volt であるという終端記号を集合の要素として追加する変換は文法規則の中でオペレータ "add"を用いて表され，

```
add [voltage($5,$6,10)]
```

次のような節に変換される．

```
S3 = [voltage($5,$6,10)|S2]
```

すなわち，文法規則における通常の記号は対象となる集合の中に同定されるとその集合から対応する要素が"retract" され，add オペレータの付加された記号は集合を表すリストの先頭へ"asserta" される．文法規則に逆変換のオペレータ"add" を導入すると，非終端記号は必ずしも与えられた集合の部分集合に対応しなくなる．そこで， Prolog の節に変換する際に述語 subset を用いることがふさわしくないので，代わりに述語 convert を用いよう．

５．下降解析と上昇解析の融合

５．１　下降解析の問題点

下降解析が陥り易い無限ループの問題を考えよう．図３に示す抵抗回路 ca40 は ca40 を述語記号とし，回路表現を項にとる論理式 (12) で表す．複合項terminal(t1,$1) は節点 $1 に接続された外部端子 t1 を示し，resistor(r1,$1,$2)は節点 $1 と $2 に接続された抵抗器 r1 を示している．

```
ca40([terminal(t1,$1),terminal(t2,$4),
      resistor(r1,$1,$2),
      resistor(r2,$2,$3),
      resistor(r3,$3,$4),
      resistor(r4,$2,$4)]).          (12)
```

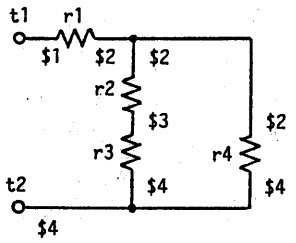

図３　回路 ca40

Fig.3　Circuit ca40

直並列回路は直列接続と並列接続を繰り返して得られる回路である．回路 ca40 が直並列回路かどうかを決定するために，次の文法規則を考えよう．

```
res(X,A,B) --> [resistor(X,A,B)];
               [resistor(X,B,A)].   (13)
anyElm(X,A) --> [terminal(X,A)];
                res(X,A,_).         (14)
spCircuit(X,A,B) --> res(X,A,B).    (15)
spCircuit(sr(X,Y),A,C) -->
              spCircuit(X,A,B),
              spCircuit(Y,B,C),
              not anyElm(_,B).      (16)
spCircuit(pr(X,Y),A,B) -->
              spCircuit(X,A,B),
              spCircuit(Y,A,B).     (17)
```

規則 (13) は接続される節点の順序に関係なく抵抗器を言及するための非終端記号 res(X,A,B) を定義している．規則 (14) は節点A に接続される任意の素子X を表す非終端記号anyElm(X,A) を定義している．非終端記号 spCircuit(X,A,B) はX が直並列回路に与えた名前を表し，A，B が接続される節点を表す．規則 (15) は一個の抵抗器を直並列回路として定義している．規則 (16) は直並列回路が直列接続されたものを直並列回路として定義している．直列回路の中点B には他のいかなる素子も接続されてはならないという条件が付加されている．同定された回路にはスコーレム関数sr(X,Y) により名前が与えられる．規則 (17) は直並列回路を並列接続したものを直並列回路として定義している．この定義を用いて次のゴール

```
?- ca40(CT),
     strans(spCircuit(X,$1,$4),CT,REST).
```

により，回路 ca40 の節点 $1 と $4 の間を直並列回路として同定しようとすると無限ループに陥る．

下降解析は出発記号を展開することで入力された終端記号と一致するものが生成され，初めて成功する．この定義では展開に (16) の直列接続が優先されるため，並列接続を含む回路 ca40 は解析されることがない．対象回路が直列接続だけで構成されておれば成功することもあるが，左回帰を含むので本質的に無限ループの問題を含んでいる．通常の文脈自由文法では中間的な非終端記号を仮定することで，左回帰を避けることができるが，ここでの文法規則は記号が変数を含み変数を共用する形で相互に関係しているので，うまくこのような非終端記号を定義することができない．

この問題を下降解析で解決するには両方の規則が公平に適用されるように，また各々の規則のサブゴールが順序によらず公平に展開されるように，素子の数の少ない順に直並列回路を枚挙するような方法を考えねばならない．文法規則に addオペレータを用いると，この問題を上昇型にとらえて解決することができる．

５．２　上昇解析を含む下降解析

規則 (16) と (17) の代わりに二個の抵抗器 X，Y の直列回路と並列回路を定義する規則を与えよう．

```
spCt(sr(X,Y),A,C) -->
              res(X,A,B),
              res(Y,B,C),
              not anyElm(_,B).      (16)'
spCt(pr(X,Y),A,B) -->
```

```
        res(X,A,B),
        res(Y,A,B).                    (17)'
```

抵抗器の直列回路からなる直並列回路は非終端記号 spCt(sr(X,Y),A,C)で表され，節点 A，C 間に直列接続された抵抗器 X，Y を素子集合から除去する変換を示している．同様に，抵抗器の並列回路からなる直並列回路は非終端記号 spCt(pr(X,Y),A,B)で表され，節点 A，B 間に並列に接続された抵抗器 X，Y を素子集合から除去する変換を表している．

ここで，これらの非終端記号を対象となる文（素子集合）の中に陽に加える操作を考える．これは規則 (18) で表され，

```
spEqivct --> spCt(X,A,C),
             add [resistor(X,A,C)].
                                       (18)
```

電気的には回路の直列等価変換に相当する．加えられた抵抗器は最初の素子集合の中に存在せず，直列回路が書き換えられたものという意味において非終端記号の性格を持っている．しかし，処理対象となる集合の要素として存在するという意味において終端記号である．すなわち，非終端記号となるべきものを終端記号に還元して処理対象となる集合の要素に加えたのである．これは処理対象自体を書き替える上昇解析の操作となっている．

ここでは簡略化のため，非終端記号に相当する等価素子も終端記号の抵抗器と同じ記号を用いたが，文法上では非終端記号で処理上では終端記号になるものと，文法上の終端記号とを分けることもできる．

この書き換え操作を制御するため次の規則を与える．

```
mspEqivct --> spEqivct,
              mspEqivct.               (19)
mspEqivct --> not spEqivct.            (20)
```

規則 (19) は直並列変換の繰り返しを定義している．規則 (20) は規則 (19) が適用できなくなるときに働く．(20)は次の Prolog の節に変換され，直並列変換が失敗したときに入力された素子集合をそのまま出力する．

```
convert(mspEqivct,S0,S0) :-
    not convert(spEqivct,S0,_).    (20)'
```

ここで直並列回路の定義として規則 (15) の代わりに与えられた回路が直並列回路であるかどうかを調べる定義 (15)'を用いよう．これは直並列等価変換を繰り返すことで二つの外部端子の間が一個の抵抗器に変換されることを要求している．

```
spCircuit(X,A,B) --> mspEqivct,
                     [terminal(_,A)],
                     [terminal(_,B)],
                     res(X,A,B).   (15)'
```

次のゴールにより回路 ca40 は直並列回路となることを証明できる．

```
?- ca40(CT),
   convert(spCircuit(X,A,B),CT,[]).
```

このゴールは成功し，次のように変数の値が得られる．得られた直並列回路の名前は成功したゴールの軌跡を保持しており，名前から回路の構造解析木が得られる（図４）．

```
X = sr(r1,pr(sr(r2,r3),r4))
A = $1
B = $4
```

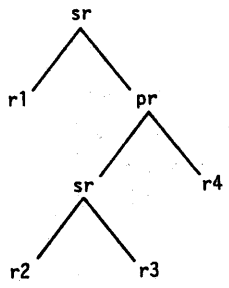

図４　回路 ca40 の構造解析木

Fig.4　Parse tree for circuit ca40

一方，外部端子を持たない回路において，与えられた二つの節点 A，B 間が直並列回路となるかどうかを調べるには次の定義を用い

ればよい．

```
spCircuit2(X,A,B) -->
        add [terminal(_,A)],
        add [terminal(_,B)],
        spCircuit(X,A,B).                (21)
```

6．プロダクション・ルールとしての文法

回路の等価変換を定義する規則（18）は次の形をしたプロダクション・ルールとしてみることができる．変換される集合がワーキング・メモリに相当する．通常の前向き推論と異なり，条件テストが行われるとテストに用いられた集合の要素が除去される．アクションとしては別の要素が集合に加えられる．

```
ruleName --> condition,
                action.
```

規則（13），（14），（16）'，（17）' などで定義される非終端記号は次の形をした条件間の包含関係の定義としてみることができる．

```
condition --> condition,
                condition.
```

一方，回路の等価変換の繰り返しを定義する規則
（19），（20）はプロダクション・ルールの適用順序を定める制御規則として見ることができる．

```
ruleName --> ruleName,
                ruleName.
```

さらに，規則（15）'は条件の定義にプロダクション自体を使う形をしている．

```
condition --> ruleName,
                condition.
```

これら性格の異なった規則が集合の変換という見方から一つのシンタクスに統一され，ＤＣＳＧの文法・論理式変換メカニズムにより Prolog のプログラムに変えられたのである．ここで一つのオペレータ"test"を導入しよう．このオペレータが付加された記号

```
test res(X,$1,$2)
```

は節点 $1 と $2 の間に抵抗器があるかどうかを調べるだけで，成功してもその抵抗器を除去するようなことはない．これは次のような節に変換され，

```
convert(res(X,$1,$2),S3,_)
```

not と同じように，要素除去の副作用のない条件テストが行える．この文法を用いて書かれたプロダクション・システムはバックトラックにより別解を得ることができる．

7．むすび

この研究で発展させた差集合に基づくＤＣＧは，Prologのプログラミングにおける一つのトゥールを提供している．とくに，与えられたデータの中に構造を見出す種類の問題は，データを素論理式の集合で表す代わりに語順を持たない文（複合項の集合）で表し，一般化された構文解析の問題に帰着させることにより，効果的に取り扱うことができる．

一方，素論理式で表される命題でもルール的なものをこの方法で扱うのは適当でない．例えば，「すべてのＸはＸの妻（夫）を愛する」"loves(X,spouse(X))"を関数とみなして対象となる集合の中に加える．文法規則の中の終端記号 [loves(john,Y)] がこの集合の中に同定されると，Y = spouse(john) が得られるが同時に集合の要素 loves(X,spouse(X)) も除去されてしまうので，ルールとしては一回しか使えないことになる．

文法規則を集合の変換規則としてとらえ，add オペレータを導入した．これにより下降解析の過程の中で対象の変形操作が行え，下降解析で陥る無限ループの問題を上昇解析を組合わせることで解決することができた．しかし，add オペレータはＤＣＧの性格を変えるため，非終端記号は部分集合に相当せず，単に集合の変換に与えた名前となった．集合の変換という見方は内容的に異なる種類の規則を統一的に扱うことを可能にしている．直並列回路の例は適用される規則が下降型に制御されたプロダクション・システムとしてみ

ることができ，プロダクションの条件もプロダクション・ルールもそのルールの制御も同一のシンタクスの下に定義される．

拡張されたＤＣＳＧは構造解析型の問題だけでなく，事象・状態変化型の問題に応用することもできよう．状態の記述に複合項の集合を用いる．事象は状態を変化させる非終端記号として定義する．事象の列は非終端記号の列で表され，ＤＣＳＧにより状態変化をシミュレートすることができる．もちろん，ある状態の構造を test オペレータにより副作用を起こさずに調べることもできる．

参考文献

1) Clocksin,W.F. and Mellish,C.S.: Programming in Prolog, Springer-Verlag, Berlin, Heidelberg, New York (1981)

2) Pereira,F. and Warren,D.: Definite Clause Grammars for Language Analysis, Artif.Intell.Vol.13,pp.231-278,(1980).

3) 田中穂積，松本裕治：自然言語処理における Prolog,情報処理,vol.25,no.12(1984).

4) T.Tanaka: "Parsing Circuit Topology in a Deductive System", Proc. IJCAI-85, pp.407-410, Los Angeles, CA. (1985).

5) 田中卓史：確定節文法を用いた電子回路の構造解析，信学論 J69D-3, pp.443-450, (1986).