集合型言語の確定節文法 DCSG と応用†

田中 卓史††

初めに，語順を持たない言語（集合型言語）の確定節文法 DCSG について述べる．これは通常の確定節文法 DCG[9] が語の部分列を示すのに二つのリストの差を用いるところを二つの集合の差（補集合）で置き換えたものである．DCSG の性質を利用すると，論理プログラミングにおいて陥るある種のループの問題を一般化された構文解析の問題に帰着して容易に解決することができる．次に，DCSG の機能を拡張する方法について述べる．文法規則を集合の変換規則としてみなし，逆変換のためのオペレータを導入する．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を行うもので，部分的な上昇解析を可能にする．この機能を用いると，左回帰の文法規則により下降解析が陥るループの問題を避けることができる．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見ることができる．DCSG 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．

1. まえがき

確定節文法 DCG[6] は文脈自由文法を述語論理の確定節を用いて表したもので，文法自体が構文解析を行うことのできる論理プログラムとなっている．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 DCG の方法を拡張して，電子回路の構成規則を確定節を用いて表す方法が提案された[10]．これは通常の DCG が語の部分列を示すのに二つのリストの差を用いるところを，二つの集合の差(補集合)で置き換えたものである[8]．補集合に基づく確定節文法は語順を持たない集合型の言語の確定節文法 DCSG として見ることができるので，回路解析に限らず多くの問題に適用することができる．そこで，初めに DCSG の性質を明らかにする．この性質を利用すると，論理プログラミングにおいて陥るある種のループの問題を一般化された構文解析の問題に帰着して容易に解決することができる．次に，DCSG の機能を拡張する方法について述べる．文法規則を集合の変換規則としてみなして，逆変換のためのオペレータを導入する．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を行うもので，部分的な上昇解析を可能にする．この機能を用いると，左回帰の文法規則により下降解析が陥るループの問題を避けることができる．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見ることができる．DCSG 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．

† DCSG: Definite Clause Set Grammars for Free Word-Order Language and Their Applications by TAKUSHI TANAKA (The National Language Research Institute).

†† 国立国語研究所

2. 集合型言語の確定節文法

2.1 集合型言語

文脈自由文法 $G=\langle Vn, Vt, P, S\rangle$ の定義をモディファイして語順を持たない言語（集合型言語）を定義することができる．ここで Vn は非終端記号の有限集合，Vt は終端記号の有限集合，P は次の形の生成規則の有限集合である．

$$A \rightarrow B_1, B_2, \cdots, B_n. \qquad (n \geqq 1)$$

$$\begin{cases} A \in Vn \\ B_i \in Vn \cup Vt \end{cases} \qquad (i=1, \cdots, n)$$

生成規則の右辺は記号列ではなくて，記号の集合（同一記号を複数個含んでよいマルチセット[1]）を表すものとする．すなわち，この規則は記号 A を記号の集合 $\{B_1, B_2, \cdots, B_n\}$ で書き換えることを意味する．この言語の文は出発記号 S ($\in Vn$) から，これらの規則を有限回適用して得られる終端記号のマルチセットである．ここで生成規則は文の生成よりも解析に用いることが多いので，単に文法規則と呼ぶことにする．

2.2 確定節文法 DCSG

集合型言語における一つの文は終端記号を要素とするマルチセットである．一方，文の導出に用いられた非終端記号はそのマルチセットの部分集合に相当している．そこで集合の要素を言及する述語 member と部分集合を言及する述語 subset を用いて集合型言語の確定節文法 DCSG (Definite Clause Set Grammar) を定義することができる．例えば，非終端記号から別の非終端記号を生成する規則

$s \rightarrow np, vp.$ (1)

は次の確定節に変換しておく.

subset(s, $S0$, $S2$) : - subset(np, $S0$, $S1$),
subset(vp, $S1$, $S2$). (1)′

ここで変数 $S0, S1, S2$ はリストを用いて表された終端記号のマルチセットが代入される. 述語 subset の第一引数は部分集合に与えた名前, 第二引数は対象となる集合, 第三引数はその部分集合を除いた残りの集合 (補集合) となるように用いる. (1)′ を解析に用いる場合, $S0$ に対象となる集合が代入される. (1)′ は手続き的に見ると, 集合 $S0$ 中に部分集合 s を同定するために, 初めに $S0$ 中に部分集合 np を同定し, 次にその補集合 $S1$ 中に部分集合 vp を同定し, その補集合を $S2$ とせよと読むことができる.

一方, 非終端記号から終端記号を生成する規則

$np \rightarrow$ [mary]. (2)

は次の確定節に変換する.

subset(np, $S0$, $S1$) : - member(mary, $S0$, $S1$). (2)′

(2)′ は集合 $S0$ 中に部分集合 np を同定するには, 集合 $S0$ の要素として mary が存在することを示せと読むことができる. ここで, 述語 member は次のように定義しておく.

member(M, [M|X], X).
member(M, [A|X], [A|Y]) : -
member(M, X, Y). (3)

述語 member は述語 subset と類似の引数をとるが, 第一引数が第二引数に代入された集合の要素となる点が異なる. 第三引数からは第一引数の要素を除いた補集合が得られる. 述語 member は(3)により直接的に定義されるので, 述語自体が集合の要素を同定する能力を持っている. 一方, 述語 subset は (1)′, (2)′ のような規則により相対的に定義され, 述語自体に部分集合を同定するような能力はない.

3. DCSG の性質

3.1 ループの問題

DCSG の性質を明らかにするために, 次の問題を解く論理プログラムを考えよう. ある電気回路で図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 を用いて次の命題で表す.

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

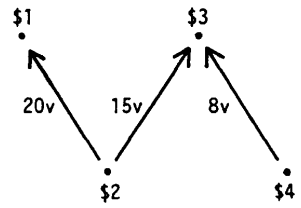

図 1 ある回路の電圧
Fig. 1 Voltages on a circuit.

"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. これらの命題を公理のデータベースに加えておく. 節点の順序に関係なく電圧のデータを求めるには, 次のように定義される述語 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 を次のように定義する.

v(A, C, V) : - volt(A, C, V). (6)

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

(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. (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 間に電圧の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.

3.2 構文解析に帰着した問題

電圧を求める問題を構文解析の問題として扱うために, "voltage" を述語記号ではなく関数記号として用いて, (4)に相当する情報を複合項のリストとして次の命題(8)で表す.

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

このリストをある回路の電圧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. 通常の

言語と異なり語順に意味はなく，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．ここで語に構造を仮定しているので，構造の変数の値が異なった語が無数に存在し，終端記号は有限集合にならない．しかし，文法規則中に変数を導入するとこれら無限個の終端記号を言及することができる．(5)に対応して次の文法規則を与える．

$$\left.\begin{array}{l}\mathrm{volt}(A, B, V) \rightarrow [\mathrm{voltage}(A, B, V)].\\ \mathrm{volt}(A, B, -V) \rightarrow [\mathrm{voltage}(B, A, V)].\end{array}\right\} \quad (9)$$

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は存在しない非終端記号である．文法規則中に導入した変数 A, B, V は規則が適用される際に，対応する記号が代入されるものとする．DCSG の文法・確定節変換プロシージャにより文法規則(9)は確定節(9)′となる．

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

電圧記述文(8)の中に非終端記号 volt(A, B, X) を同定するには，次のゴールを実行すればよい．

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

変数 A, B, X は(5)の対応する変数と同じように振る舞う．異なる点は，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，

$$\mathrm{v}(A, C, V) \rightarrow \mathrm{volt}(A, C, V). \quad (10)$$

$$\mathrm{v}(A, C, V+W) \rightarrow \mathrm{volt}(A, B, V), \mathrm{v}(B, C, W). \quad (11)$$

次の確定節に変換される．

```
subset(v(A, C, V), S0, S1) : -
    subset(volt(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 を求めるには電圧記述文の中に非終端記号 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，ループに陥るようなことは起こらない．図1の節点 $1, $4 間の電圧は次のようにして求められる．

```
?- vData(S0),
   subset(v($1, $4, X), S0, _).
```

$X=20+(-15+8)$

論理プログラムの実行を後向き推論による定理証明の過程として見たときに，3.1 節のループの問題は同一の公理が多重に用いられることに起因していた．一方，3.2 節ではファクト型の公理を終端記号として扱い，ルール型の公理を文法規則として扱うことで，電圧を求める問題を非終端記号の同定の問題に帰着させた．文脈自由型の言語では一つの文を構成するどの終端記号も非終端記号への還元に一度しか寄与できないので，必然的に同じデータを多重に用いるループの問題は解消されることになった．

集合型言語の文法規則を後向き推論規則として見ると，推論に一度使ったデータを何度も使わないという性質は我々の暗黙の仮定ともよくなじむ．同じ方法を用いて，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できる．

4. DCSG の拡張

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 member(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．したがって，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終端記号の組合せとして定義されるので，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．終端記号による特

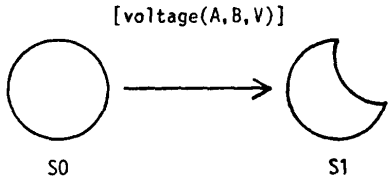

図 2 終端記号による集合の変換
Fig. 2 Set conversion by terminal symbol.

定の要素の除去変換の逆はその要素を集合に加えることである．非終端記号による部分集合の除去変換の逆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，一般に変数が入出力に関して双方向的であることを利用して，述語 member や subset の入出力の関係を取り換えればよい．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．しかし要素除去の逆変換を述語 member を用いて次のように行うと，

?- member(m, OUT, [a, b, c]).

集合の表現にリストを用いたので，OUT=[m, a, b, c]; [a, m, b, c]; [a, b, m, c]; [a, b, c, m] の場合が生じ，単に要素追加の目的で述語 member を用いるのは機能が大き過ぎることになる．一方，非終端記号による部分集合の除去変換の逆も述語 subset の入出力の関係を取り換えたのでは具合の悪い問題が生じる．解析に使うことを想定して定義した文法規則をそのまま生成に使うと，(11)′の変数 B のように値が定まらず，存在的に解釈せねばならない変数が残ることになる．

4.2 add オペレータ

集合の変換を考える上で，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のみを考えることにする．節点 $5 と $6 の間の電圧が 10 volt であることを示す終端記号を集合の要素として追加するには文法規則中で逆変換のオペレータ “add” を用いて

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

のように表して，次のような節に変換する．

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

すなわち，文法規則において通常の記号は対象となる集合の中に同定されると対応する要素が除去され，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追加される．文法規則に逆変換のオペレータを導入すると，定義される非終端記号は必ずしも部分集合には相当しなくなる．そこで，非終端記号を確定節に変換する際の述語記号として部分集合を意味する subset を用いることがふさわしくないので，拡張 DCSG では代わりに，単に集合の変換を意味する述語記号 convert を用いることにする．

5. 下降解析と上昇解析の融合

5.1 下降解析の問題点

下降解析が陥りやすいループの問題として回路の構造解析の例を考える[7),9)]．図 3 の抵抗回路 ca40 は

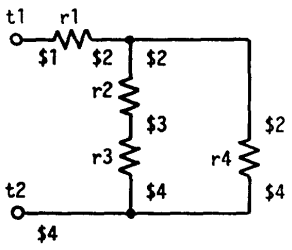

図 3 回路 ca40
Fig. 3 Circuit ca40.

ca40 を述語記号とする命題(12)で表す．ここで複合項 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)

二端子素子の直列接続と並列接続とを繰り返して得られる回路は直並列回路と呼ばれている．回路 ca40 が抵抗器の直並列回路かどうかを決定するために次の規則を与える．

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)

(13)は節点の順序に無関係に抵抗器を言及できる非終端記号 res(X, A, B) を定義している[9)]．(14)は節点 A に接続される任意の素子 X を言及する非終端記号 anyElm(X, A) を定義している．直並列回路を言及する非終端記号 spCircuit(X, A, B) は X が回路に与えた名前，A, B が接続される節点を表す．(15)は 1 個の抵抗器を直並列回路として定義している．(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 には他の素子が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．(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．この定義を用いてゴール(18)により，回路 ca40 の節点 $1, $4 間を直並列回路として同定しようとするとループに陥る．

```
?- ca40(CT),
   subset(spCircuit(X, $1, $4), CT, REST).
                                        (18)
```

(16)，(17)は文法規則の右辺の左端に左辺と同じ形の非終端記号が現れる左回帰の文法規則と呼ばれるもので，DCSG の下降解析のメカニズムでは，一般にループに陥る危険性を持っている．例えば(16)に基づく確定節の第一のサブゴールがたまたま(15)により成功すると，次のサブゴールへと進むことができるのだが，いったん失敗すると第一のサブゴールは同じ規則(16)でサブゴールに展開されるので，同じ失敗を繰り返し，無限にサブゴールを展開してゆく．

ゴール(18)では節点 $1, $4 間の直並列回路の同定に(15)が失敗し，(16)が適用されて第一のサブゴールは $X=r1$ で成功する．第二のサブゴールも $Y=r4$ で成功するが，第三の条件 "not anyElm(_,$2)" で失敗しバックトラックする．次に，第二のサブゴールは(16)で展開され，$Y=$sr($r2, r3$) で成功するが再び第三の条件で失敗しバックトラックする．ここで第二のサブゴールが(17)で展開されると成功につながるのだが，そうはならずに，最後に行った選択の別の選択として第二の第二のサブゴールを(16)で展開し，うまく行かず，さらに(16)で無限にサブゴールを展開する．

下降解析を成功させるには左回帰の文法規則を除去せねばならない．一般の文脈自由文法では補助的な非終端記号を導入することで，左回帰の規則を右回帰に書き換える方法が知られている（グライバッハの標準形[5]）．文法規則が変数を含むので機械的にこの方法を適用することはできないが，非終端記号として三端子回路を導入すると下降型に直並列回路を同定できる文法規則が得られる（付録）．この規則に基づく下降解析は対象回路中の素子の増加とともに急激に解析の速度が落ち，後述の上昇解析の方法と比較して非常に能率が悪い．左回帰の文法規則を扱えるように，上昇解析を行う論理プログラムに変換する研究も行われているが[2]，直ちに応用することはできない．

5.2 下降型に制御された上昇解析

文法規則に add オペレータを用いることのできる拡張 DCSG では，このループの問題を異なった方法で効率よく解決することができる．規則(16)と(17)の代わりに2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る．

```
rSeries(sr(X, Y), A, C) →
                    res(X, A, B),
                    res(Y, B, C),
                    not anyElm(_, B).    (16)'
rParallel(pr(X, Y), A, B) → res(X, A, B),
                            res(Y, A, B).  (17)'
```

直列回路を表す非終端記号 rSeries(sr(X, Y), A, C) は節点 A, C 間に存在する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である．今，この直列回路を除去して，新たに節点 A, C 間に同名の抵抗器 resistor(sr(X, Y), A, C) を加えることを考える．この集合の変換は(19)で表され（**図 4**），

```
srEqivct → rSeries(X, A, C),
           add [resistor(X, A, C)].      (19)
```

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は存在せず，直列回路が書き換えられたものであるから，文法的には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．非終端記号となるべきものを終端記号に書き換えることで，(16)'，(17)' の規則を実質的に直並列回路に対しても使えるようになる．もちろん処理上の終端記号に別の名前を与え，実際の終端記号と分けて扱うこともできる[9]．

並列接続された回路を除去して，等価な素子を加える変換は同様に(20)で表される．

```
prEqivct → rParallel(X, A, B),
           add [resistor(X, A, B)].      (20)
```

これらの規則による集合の変換は，終端記号をより上位の非終端記号に書き換えていく上昇解析の操作として見ることができる．この書き換え操作を制御するため次の規則を与えると，等価変換が行えなくなるまで繰り返すことができる．

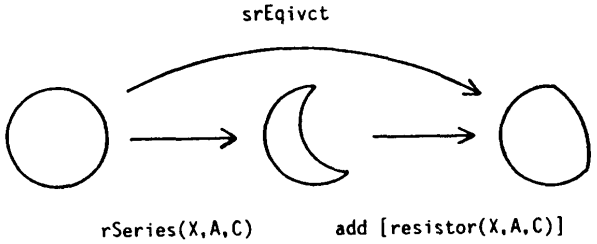

図 4 "srEqivct" による集合の変換
Fig. 4 Set conversion by "srEqivct".

```
spEqivct → srEqivct ;
           prEqivct.                    (21)
mspEqivct → spEqivct,
            mspEqivct.                  (22)
mspEqivct → not spEqivct.               (23)
```

規則(21)は直列変換と並列変換の両方を示す直並列変換を定義している．(22)は直並列変換を繰り返す変換を定義している．(23)は(22)が適用できなくなるときに働く．(23)は次の確定節に変換され，直並列変換が失敗したときに入力された回路をそのまま出力する．

```
convert(mspEqivct, S0, S0) : -
    not convert(spEqivct, S0, _).       (23)'
```

ここで直並列回路の定義として規則(15)の代わりに，与えられた回路全体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1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 5**）．

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

6. 文法規則とプロダクション・システム

回路の等価変換を定義する規則(19)，(20)は次の形をしたプロダクション・ルールとして見ることができる．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の包含関係として見ることができる．

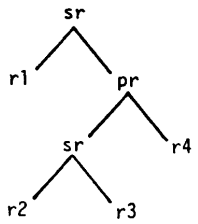

図 5 回路 ca40 の構造解析木
Fig. 5 Parse tree for circuit ca40.

```
condition → condition,
            condition.
```

一方，回路の等価変換の繰り返しを定義する規則(21)，(22)，(23)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.
```

これらの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ックスに統一されて，DCSG の文法・論理式変換プロシージャで論理プログラムに変えられている．ここで新しくオペレータ “test” を導入しよう．これが付加された記号

```
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．test は not と同じように，要素除去の副作用を伴わない条件テストが行える．test と add を組み合わせて用いると通常のプロダクション・ルールを書くことができる．この方法で得ら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．

文脈に依存する集合型言語（文脈自由とならない回路[9),10)]）を考えるときに，文法規則の適用は生成の場合にも解析の場合にも，記号の出現環境をテストすることが必要になる．test オペレータを用いると副作用を伴わずに環境のテストを行うことができる．

7. むすび

この研究で発展させた DCSG は論理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．特に，与

えられたデータ中に構造を見いだす種類の問題は，データを語順を持たない文（複合項の集合）として扱い，構文解析の問題に帰着させて効果的に解決できる．

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を書き換える上昇解析の操作が行え，下降解析で陥る無限ループの問題を解決することができた．しかし，add オペレータは DCSG 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・システムとして見ることができ，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ックスの下に定義された．add オペレータにより DCSG はバックトラックの行えるプロダクション・システムの機能を備えたのである．

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も応用することができる．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．非終端記号の列は事象の列を表し，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．

謝辞 日頃，有益な討論を行っている人工知能の勉強会 AIUEO の皆様と，論理プログラミングを教えていただいた多くの友人に感謝する．

参考文献

1) Dershowitz, N. and Manna, Z.: Proving Termination with Multiset Orderings, *CACM*, Vol. 22, pp. 465-476 (1979).
2) 伊藤，中川：文脈自由言語パーザへの Prolog プログラム変換の応用，AI 学会誌，Vol. 1, No. 2, pp. 250-253 (1986).
3) 松本，田中，平川，三吉，安川，向井，横井：Prolog に埋め込まれたボトムアップパーサ：BUP, *Proc. LPC '83*, ICOT (1983).
4) 中島秀之：Prolog, 産業図書，東京 (1983).
5) 西田富士夫：言語情報処理，コロナ社，東京 (1981).
6) Pereira, F. and Warren, D.: Definite Clause Grammars for Language Analysis, *Artif. Intell.*, Vol. 13, pp. 231-278 (1980).
7) Tanaka, T.: Representation and Analysis of Electrical Circuits in a Deductive System, *Proc. IJCAI-83*, pp. 263-267, Karlsruhe, West-Germany (1983).
8) Tanaka, T.: Parsing Circuit Topology in a Deductive System, *Proc. IJCAI-85*, pp. 407-410, Los Angeles, CA. (1985).
9) 田中卓史：論理プログラミングによる回路の表現と構造の解析，電子通信学会論文誌A，Vol. J 68 A, No. 12, pp. 1350-1356 (1985).
10) 田中卓史：確定節文法を用いた電子回路の構造解析，電子通信学会論文誌D，Vol. J 69 D, No. 3, pp. 443-450 (1986).

付録 下降型に直並列回路を同定できる文法規則

```
spCircuit(t(R, T), A, B) → res(R, B, C),
                           tt(T, A, B, C).   (a)
tt(nil, A, B, A).                            (b)
tt(s(X, T), A, B, C) → spCircuit(X, C, D),
                       tt(T, A, B, D),
                       not anyElm(-, C).     (c)
tt(p(X, T), A, B, C) → spCircuit(X, B, C),
                       tt(T, A, B, C).       (d)
```

節点 A, B 間に存在する直並列回路を節点 B につながる1個の抵抗器 R と残りの回路 T（三端子回路 tt）に分ける（a）．この三端子回路を（b），（c），（d）の場合で定義する（**図 6**）．

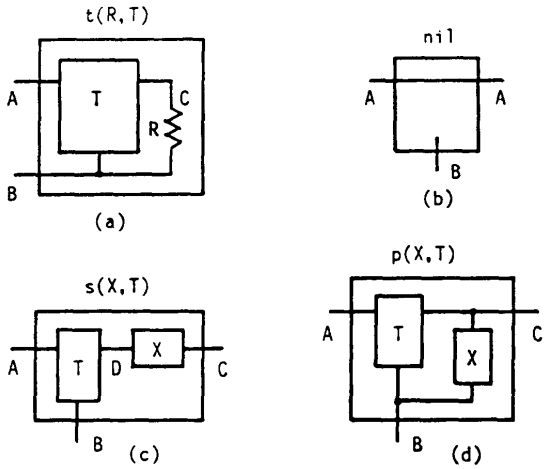

図 6 三端子回路を用いた直並列回路の定義
Fig. 6 Definition of series-parallel circuits using three terminal circuits.

（昭和61年4月15日受付）
（昭和62年9月 9 日採録）

田中　卓史（正会員）

昭和19年生．昭和42年九州大学工学部電子工学科卒業．昭和47年同大学院博士課程単位修得．同年九州大学工学部助手．昭和52年より国立国語研究所勤務．主任研究官．昭和57-58年 Yale 大学計算機科学科 Pd フェロー．人工知能の立場からの言語研究に従事．工学博士．電子情報通信学会，人工知能学会，認知科学会，計量国語学会，AIUEO, 米国 IEEE-CS, AAAI 各会員．