記号処理 4 － 3
（1978. 5. 19）

ブール関数処理システム Syllogister

竹島 卓　（富士通国際情報社会科学研究所）

1. まえがき

筆者は、人工の手足、いわゆるロボットの研究をしていたときに、数式処理に出会った。それは、ロボットの機構学的形状・状態をあらわすのに、三角関数の整式を要素とする4行4列のマトリクスを20個程掛け合わせたり、その行列式をとったりすることが必要であったからである。筆者は、共同研究者である他の2名とこの計算に挑戦して、途中でマトリクスの1要素が500ページ程になることを知って、さすがにこの試みを放棄してしまった。この計算は、記憶量や計算時間の膨大さを恐れて、未だに数式処理システムでも実行されていない。今では普通の数式処理システムとして、東大後藤研究室より移植した HLISP/REDUCE を当社の M190 で利用している。

ところで、筆者はデータベースにおけるリレーショナルモデル[v]の研究に携わり、その理論の中で扱われる重要な概念である関数従属という概念が、ブール代数によって完全に記述できることを見出した。この方法は、一般に、依存関係や有向グラフにおける推移反射閉包にも応用できることがわかり、種々計算をしているうちに、これは数式処理に適した問題であると認識するに至った。なぜならば、この種のブール代数の応用においては、最終的に数値的な結果を得ることが目的ではなく、式のままでみて変数間にどのような関係があるかを知ることが重要な目的であることが多いからである。また、ブール代数における方程式の解は一般に任意定数を含むため、数値的に解くということが殆どナンセンスに近いということも、理由の一つとしてあげられる。

ブール代数の数式処理については、筆者が研究のヒントを得た、山田[ii]の Syllogister や、今井[iii]の論文があるが、前者は、計算機がまだあるかなきかの時代に作成された専用の機械（らしい）ということもあり、（詳細はわかっていない）大量・大規模の応用分野には向かなかったであろう。後者は、ブール計画法、擬似ブール計画法への応用を目指して、大規模かつ高速性を追求した、大型計算機バッチ処理向きのシステムであると見られ、数式処理というよりは、係数処理に近いものといえよう。この他に、岡田[iv]によるブール式の素項生成プログラム等がある。筆者は、（数値処理においても同じことが言えるが）数式処理においては、大量・大規模な計算で、計算機がやっても到底できそうもないし、結果を見ても、人間の理解を越えるような計算をやらせる（のも必要とあらば仕方がないが）よりも、人間がちょっと計算するには大変だが、機械ならすぐ答えが出せ、人間にもすぐ役に立つような電卓的使用法が大切なのではないかと考えている。実際 APL や MUMPS SPEAKEASY などの TSS 言語が流行しているのは、そういった手軽さ・使おうと思ったときすぐ役に立つといった機能が買われてのことだろうと思われる。したがって、筆者の Syllogister（山田教授にちなんで同名のシステムとした）は、将来は、TSS 的な使い方ができるように、言語設計、処理系の設計をしていこうと意図している。

本稿では、第2節において、ブール代数の諸定義と諸定理を紹介する。resultant と resultant 定理とは山田教授に負うものであるが、筆者の利用に

合わせて、拡張した一般形で述べてある。

また、方程式の解表示、不等式の解法等は筆者の工夫であるが、必ずしも最善のものとは言えないので、他の良い方法等については、読者諸氏の御教示を乞いたいと考えている。 また本稿では、ブール式一般について扱ったが、ブール関数のクラスを、その積和標準形の係数が0か1に限ったもの（仮に整関数と呼ぶ）に制限した場合は、さらに有用な定理が証明できる。

実はリレーショナルモデルにおいて役立つのは、この整関数のクラスであり、多くの問題は整関数のクラスでとり扱えることから、Syllogister に整関数特有の option を用意することも必要であろう。

第3節では、Syllogister の対象とする式とその処理法の原理について説明(釈明?)した。さらに、Syllogisterのうちの核になる式の処理プログラムについても簡単に説明した。 このプログラムはほぼ pure LISP で書かれており（実はHLISP）実際に動くプログラムを付録にしておいた。百行程度のプログラムであるから、誰でも手軽に利用できると考える。

第4節では、簡単なまとめと感想を述べて全体の終わりとする。

2. ブール代数

2.1 基本的事項

集合Bにおいて、2つの内部演算＋と・とが定義されており、次の(i)～(vi)の公理Aを満足するとき、Bと公理Aとを組にして$\langle B, A\rangle$をブール代数と呼ぶ。Bをブール代数の基礎集合と呼ぶ。ブール代数は、Bと演算＋，・および公理にいう特別な元1と0，そして＋と・とのある意味での逆演算に相当する$'$とが指定されれば定まるので、組$\langle B, 1, 0, +, \cdot, ' \rangle$によってブール代数をあらわすこともある。さて公理Aは、Bに属する任意の元a, b, cについて次のように書かれる。

(i) 閉則　$a+b \in B$,　$a \cdot b \in B$　（それぞれ唯一の元が定まる）
(ii) 交換則　$a+b=b+a$,　$ab=ba$
(iii) 分配則　$a+bc=(a+b)(a+c)$,　$a(b+c)=ab+ac$
(iv) 吸収則　$a+ab=a$,　$a\cdot(a+b)=a$
(v) 恒等則　特別な元1と0とがBにあって　$a+0=a$,　$a\cdot 1=a$
(vi) 補元則　aに対してある元αがBにあって　$a+\alpha=1$,　$a\alpha=0$

以上の公理は、＋と・，1と0とについての交換によって変わらないため、ブール代数においては＋と・，1と0の入れ替えは双対となっており、このためにある定理が証明されれば、その双対も証明可能である。

以上のように定義されるブール代数$\langle B, A\rangle$について、次の諸定理が成り立つ。

(vii) 結合則　$(a+b)+c=a+(b+c)$,　$(ab)c=a(bc)$
(viii) 恒等元の唯一性　公理(v)の恒等元1と0とはそれぞれ唯一である。
(ix) 補元の唯一性　公理(vi)の補元αはaに対して唯一である。

このことから、単項演算（補元演算）$'$を$a'=\alpha$によって定義する。

(x) 巾等則　$a+a=a$,　$aa=a$
(xi) 復元則　$(a')'=a$
(xii) 帰無則　$a+1=1$,　$a\cdot 0=0$
(xiii) 簡化則　$a+a'b=a+b$,　$a(a'+b)=ab$

(xiv) ドモルガン則　　$(a+b)' = a' b'$,　$(ab)' = a' + b'$

(xv) (名称なし)　　$0' = 1$,　$1' = 0$

2.2 ブール関数，ブール式と諸定理

Bをブール代数の基礎集合とする。　B^N からBへの写像 $f : B^N \to B$ の中で、演算 $+$ と $\cdot$, $'$ の有限回の合成によってあらわされるものをブール関数と呼ぶのが慣例である。　これは通常の数の上での連続関数に相当する。

数式処理の立場からすれば、関数と式とは厳密に区別されるべきであるが、ここでは特に区別の必要があるとき以外は、通常の数学的記法 $f(x)$ や $g(x, y)$ などで関数と式との両方をあらわすことにする。

ブール代数において重要であり、数式処理においても重要な役割を果たす諸定義と諸定理を以下に示す

(定義　1)　基本積(あるいは基本和)

n 個の変数 $x_1, x_2, \cdots, x_n$ の基本積(和)とは、それらの変数あるいは補元演算の付いた変数 n コの積(和)であって $0(1)$ に等しくはないもののことである。 ‖

変数に補元演算のついたものを補元変数ともいう。　変数と補元変数とを合わせてリテラルと呼ぶこともある。

[定理　1]　展開定理

任意のブール関数は、基本積あるいは基本和によって次のようにそれぞれ一意にあらわされる。

$$f(x_1, x_2, \cdots, x_n) \equiv \sum_{\langle t_1, t_2, \cdots, t_m \rangle \in B_0^n} f(t_1, t_2, \cdots, t_m)\, \xi_1 \cdot \xi_2 \cdot \cdots \cdot \xi_n$$

$$\equiv \prod_{\langle t_1, t_2, \cdots, t_m \rangle \in B_0^n} \left(f(t_1', t_2', \cdots, t_m') + \xi_1 + \xi_2 + \cdots + \xi_m \right)$$

ここに　$B_0 = \{0, 1\}$,　$\xi_i = \begin{cases} x_i & t_i = 1 \text{ のとき} \\ x_i' & t_i = 0 \text{ のとき} \end{cases}$　($\forall i \in \{1, 2, \cdots, n\}$ に対して)

とする．　Σ の方を積和標準形，Π の方を和積標準形という。 ‖ (証明　略)

[定理　2]

(i) $a = b$, (ii) $ab' + a'b = 0$, (iii) $ab + a'b' = 1$　は互いに同値である。 ‖ (証明略)

[定理　3]

$a + b = 0$ と $a = 0$ かつ $b = 0$ とは同値、双対に $a \cdot b = 1$ と $a = 1$ かつ $b = 1$ とは同値。 ‖　(証明　略)

[定理　4]

ブール代数では任意個の連立する等式を右辺が0の単一の等式に等価変換できる。　そのような単一の等式を0型の等式という。　双対に、右辺が1の単一の等式にも等価変換ができる。　これを1型の等式という。 ‖　(略証)　定理2，定理3　によって明らか。 ‖

以下では主として0型の等式を利用する。　これを用いた方程式の取り扱いにおいて重要な役割を果たすブール関数の resultant を定義する。

(定義　2)

n 変数 ($n \geq 0$) のブール関数 $f(x_1, x_2, \cdots, x_n)$ の m 変数($n \geq m \geq 0$) resultant

$$r(x_{\sigma(1)}, x_{\sigma(2)}, \cdots, x_{\sigma(m)}) = f_{x_{\sigma(1)}, x_{\sigma(2)}, \cdots, x_{\sigma(m)}}(x_{\sigma(1)}, x_{\sigma(2)}, \cdots, x_{\sigma(m)} \mid x_{\sigma(m+1)}, x_{\sigma(m+2)}, \cdots, x_{\sigma(n)})$$

は次のように定義される。ここに σ は順列 $\{1, 2, \cdots, n\}$ の置換演算である。

$$r(x_{\sigma(1)}, x_{\sigma(2)}, \cdots, x_{\sigma(m)}) \triangleq \prod_{\langle t_{m+1}, t_{m+2}, \cdots, t_n \rangle \in B_0^{n-m}} f(x_1, x_2, \cdots, x_n) \Big| \begin{array}{l} \langle x_{\sigma(m+1)}, x_{\sigma(m+2)}, \cdots, x_{\sigma(n)} \rangle \\ = \langle t_{m+1}, t_{m+2}, \cdots, t_n \rangle \end{array} \quad \blacksquare$$

［定理　5］　resultant 定理

n 変数（$n \geq 0$）ブール関数を $f(x_1, x_2, \cdots, x_n)$ とし、その任意の resultant の1つを $r(x_{\sigma(1)}, x_{\sigma(2)}, \cdots, x_{\sigma(m)})$　$(n \geq m \geq 0)$ とするとき、次の命題が成り立つ。

$$\exists x_1, x_2, \cdots, x_n \; f(x_1, x_2, \cdots, x_n) = 0$$
$$\Leftrightarrow \quad \exists x_{\sigma(1)}, x_{\sigma(2)}, \cdots, x_{\sigma(m)} \; r(x_{\sigma(1)}, x_{\sigma(2)}, \cdots, x_{\sigma(m)}) = 0 \quad \blacksquare$$

（略証）　$n = 0$ の場合は自明。（意味はとりにくいが）　$n = 1$ の場合は2つの場合に分ける。　(i) $m = 1$ のとき、これは自明。　(ii) $m = 0$ のとき、証明すべき命題は x_1 を x とかいて　$\exists x \, f(x) = 0 \Leftrightarrow f(1) f(0) = 0$ である。　先ず $\Rightarrow$ を示す。$\exists x f(x) = 0$ より $f(x) = 0$ をみたす x を a とすると、$f(a) = f(1) a + f(0) a' = 0$, 両辺に $f(1) f(0)$ を掛けて整理すると、$f(1) f(0) = 0$ となる。　$\Leftarrow$ は $f(0) f(1) = 0$ のもとで $x = f(0)$ が $f(x) = 0$ をみたすことを示せばよい。　$n \geq 2$ のときは、n に関する数学的帰納法による。 $\blacksquare$

［系　5・1］　方程式の解の存在

n 変数ブール関数 $f(x_1, x_2, \cdots, x_n)$ についての方程式　$f(x_1, x_2, \cdots, x_n) = 0$ が解をもつ（有解）必要かつ十分な条件は、0変数 resultant $f(\ | x_1, x_2, \cdots, x_n)$ が0となることである。 $\blacksquare$　（証明　略）

［定理　6］　解の公式

n 変数ブール関数 $f(x_1, x_2, \cdots, x_n)$ に関するブール方程式　$f(x_1, x_2, \cdots, x_n) = 0$ が解をもつとき、その解は次の式で与えられる。

$$x_1 = x_1^* \triangleq f_{x_1, x_2, \cdots, x_n}(0, x_2^*, \cdots, x_n^*) + u_1 f'_{x_1, x_2, \cdots, x_n}(1, x_2^*, x_3^*, \cdots, x_n^*)$$

$$x_2 = x_2^* \triangleq f_{x_2, x_3, \cdots, x_n}(0, x_3^*, x_4^*, \cdots, x_n^* \mid x_1) + u_2 f'_{x_2, x_3, \cdots, x_n}(1, x_3^*, x_4^*, \cdots, x_n^* \mid x_1)$$

$$\vdots$$

$$x_n = x_n^* \triangleq f_{x_n}(0 \mid x_1, x_2, \cdots, x_{n-1}) + u_n \cdot f'_{x_n}(1 \mid x_1, x_2, \cdots, x_{n-1})$$

ここに $u_1, u_2, \cdots, u_n$ は任意定数である。 $\blacksquare$

（略証）　$n = 0$ の場合は自明　（意味はわかりにくいが）　$n = 1$ の場合、証明すべき命題は、x_1 を x, u_1 を u と書いて「$f(x) = 0$ が有解のとき、解は $x = x^* = f(0) + u \cdot f'(1)$ である。」となる。$f(x^*)$ を計算すると　$f(x^*) = f(1) x^* + f(0)(x^*)' = f(1) f(0) = 0$　したがって $x = x^*$ は確かに解である。　x^* で解が尽くされていることを示すには、$a \in B$ を $f(a) = 0$ の任意解の1つとしたとき、適当な u があって $a = f(0) + u f'(1)$ となることを示せばよい。等式の同値変換定理により

$$(f(0) + u f'(1))' a + (f(0) + u f'(1)) a' = 0$$

この左辺を u の関数 $g(u)$ とみて resultant $g(|u)$ を求めると　$g(|u) = f(1) f'(0) a + f(0) a'$ となる。ここで a は $f(a) = f(1) a + f(0) a' = 0$ をみたすから $f(1) a = 0$, $f(0) a' = 0$ である。　これを $g(|u)$ の右辺に入れて、$g(|u) = 0$.　したがって resultant 定理によって　$\exists u \, g(u) = 0$ が成り立ち、解は x^* で尽くされていることがわかる。$n \geq 2$ の場合は n に関する数学的帰納法で証明できる。 $\blacksquare$

次に不等式の解を求めるための基礎となる定理をあげておく。

［定理 7］

(i) $ab=0$ かつ $b\neq 0$ ならば、$a\neq 1$

(ii) $\exists b\,(ab=0 \wedge b\neq 0) \Leftrightarrow a\neq 1$ ∎ （証明 略）

定理7を用いるとブール不等方程式 $f(x_1, x_2, \cdots, x_n)\neq 0$ を slac 変数 y を用いた連立方程式 $yf'(x_1, x_2, \cdots, x_n)=0$, $y\neq 0$ に等価変換ができる。

最後に、グラフや dependency diagram 等の推移反射閉包を求めたり、その上での推理を行なう上での基礎となる定理をあげる。

［定理 8］ resultant の最大性定理

n 変数 $(n\geq 0)$ ブール関数 $f(x_1, x_2, \cdots, x_n)$ と m 変数 $(n\geq m\geq 0)$ ブール関数 $g(x_{\sigma(1)}, x_{\sigma(2)}, \cdots, x_{\sigma(m)})$, および g と丁度同じ変数をもつ f の resultant $r(x_{\sigma(1)}, x_{\sigma(2)}, \cdots, x_{\sigma(m)})$ とについて次の命題が成り立つ。

$\forall x_1, x_2, \cdots, x_n\,[f=0 \Longrightarrow g=0] \Longrightarrow \forall x_{\sigma(1)}, x_{\sigma(2)}, \cdots, x_{\sigma(m)}\,[r=0 \Longrightarrow g=0]$ ∎

（略証） 変数の適当な番号の付け替えと並べかえを行なうことによって、σ を恒等置換と考えても一般性を失わない。 よって、σ は恒等置換であるとする。 $f=0$ が解をもたないときは resultant 定理によって $r=0$ も解をもたないからこの場合定理は自明である。 よって今 $f=0$ が有解であるとする。 解の公式によると、$f=0$ の一般解は、その任意の resultant を0とおいた方程式の一般解を用いてあらわされている。今の場合、特に $r=0$ の任意解 $x_1=x_1^*$, $x_2=x_2^*$, $\cdots$ $x_m=x_m^*$ によって、$f=0$ の任意解は $x_1=x_1^*$, $x_2=x_2^*$, $\cdots$, $x_m=x_m^*$, $x_{m+1}=x_{m+1}^*$, $\cdots$, $x_n=x_n^*$ とあらわされる。 したがって $f=0$ をみたす任意の x_1, x_2, $\cdots$, x_m について $g=0$ となるならば $r=0$ をみたす任意の x_1, x_2, $\cdots$, x_m について $g=0$ となる。 ∎

3 Syllogister

3.1 Syllogister の対象

Syllogister の対象は、ブール代数における数式である。 したがって、それに必要な概念は、ブール定数、ブール変数、これらから作られるブール式，ブール式を関数化して得られる関数，関数変数，ブール式を値とするブール式変数などであろう。 さらに何々変数と名付けられるものには、独立変数・従属変数という種類分けも必要であろう。

独立変数とは、その値が、他の変数への値の assignment によって変わらないものであり、従属変数とは、それが従属すると指定された変数への値の assignment によって、その値が定まるものである。

現在のところ、Syllogister が扱えるのは、変数と定数1，0のみから成るブール式であり、正式の言語仕様も定まっていないが、将来はここに述べたような概念まで取り扱えるようにする予定である。

3.2 問題点

ブール式の数式処理に特徴的な問題点、留意点を列挙して、その解決方法を考える。

(3.2.1) 標準形の問題

ブール式の標準形として適当なものは、積和標準形（あるいは双対の和積標準形）の正記形であろう。これは基本積の順序を定めて並べると約束すれば、表現

が一意な点で望ましいが、関与する変数をnとすると、2^n個の基本積項の各々に対して、その係数（必ず1も0，1とは限らない）を指定しなければならず、表現のためのspaceの点からは得策ではない。Bが{0,1}のときにはがロア標準形も表現の一意性があり、しかも必要spaceが少くてすむ可能性があるけれども、一般のブール束については、少し工夫しなければ表現の一意性が成立しない。

筆者は、任意の2つの式が与えられたとき、それらが等価か否かを決定するアルゴリズムがあることと、通常慣れ親しんでいることとの2点から、積和形を式のstyleとすることにした。（積和形にすることは、双対性からoptionalに簡単にできる）2つの式が等価か否かを決定するには、基本的には積和標準形の係数の一致を調べればよいのだが、どの式も積和形にするという条件がみたされるならば、2つの式$\xi = \eta$とについての対称差$\xi \oplus \eta \leq \xi\eta' + \xi'\eta$が0になれば$\xi = \eta$さもなくば$\xi \neq \eta$とすればよい。式を積和形にするための変形ルールは次のものである。

(i) 分配則　$x(y+z) \to xy + xz$　(ii) 帰無則　$0 \cdot x \to 0$, $1+x \to 1$
(iii) 補元則　$x \cdot x' \to 0$, $x + x' \to 1$　(iv) 恒等則　$0+x \to x$, $1 \cdot x \to x$
(v) 復元則　$(x')' \to x$　(vi) ドモルガン則　$(x+y)' \to x'y'$, $(xy)' \to x'+y'$
(vii) および交換・結合則については、適当な順位をつけて式の要素を並べかえること。

これら7個のルールを適用すれば、求める積和形が得られることはほぼ自明であろう。このプログラムは後に示す。

(3.2.2) 簡約形の問題

標準形の問題は、無数にある等価な式から1つを選定する問題であったが、簡約形の問題は、式の表現をある言語の語とみたとき、その長さを最小にする問題として定式化できる。通常は、積和形の項の数を最小とする問題をその代表的なものとしている。この最小化問題の解は、交換・結合則によって等価なものは同一視したとしても、一般に唯一ではなく、本質的に異なる解が存在する。例えば、$xy' + xz' + x'y$ と $xy' + x'y + yz$ とは本質的に異なる等価な式であって、共に等価な式の中では最簡形である。

唯一性が保証できるような最簡形があれば、それを標準形として採用することもでき、大いに有用ではあるが、最簡形を得るための手間が果たして、妥当な出費かどうか、はなはだ疑問である。したがって最簡形を追求するよりは、必要に応じて、それが得られるようにoptionalな機能としておくことが実際的であろうと考えられる。この事に関連した簡化則の問題は(3.2.5)で述べる。

(3.2.3) 方程式とresultant

ブール方程式の解は一般に任意定数を含むもので、一般解として数式的に求めるものになる。解表示は定理6によって、resultantを用いて表わされるので、一般に式や関数のresultantをとる演算がブール式処理システムに用意されていることが要求される。

(3.2.4) 補元変数の取り扱い

これは難問である。補元変数を変数と認めると、結局、変数と補元変数は常に一組にして取り扱わねばならなくなり、これをうまく実現することは工夫を要する。しかも我々人間が入力する時は、x'と書いて変数のつもりか、補元をとったものか、意図が不明であり、翻訳が困難である。当面、Syllogisterでは、

補元変数という変数は扱わない。

(3.2.5) ブール代数に特有の演算規則の問題

通常の数の四則のものと大きく異なるものは、ド・モルガン則，吸収則，簡化則であろう。このうち、ド・モルガン則は簡明で、より外側の ' を内側に入れるものとして処理すればよい。しかしながら、吸収則はさほど簡単にはいかない。素朴に考えれば、ある式が他の式の一部になっているか否かをパターンマッチングで判定せねばならない。しかも交換・結合則が成立するから、たとえ何かの順序で整列されていたとしても、かなり複雑なアルゴリズムをとらねばならなくなる。この難点は、等価性の判定のアルゴリズムと同じ条件下で、次の定理を利用することによって可能である。

[定理 9] 吸収則の定理 (i) $a+b=a \iff$ (ii) $a'b=0$ ∎ (証明 略)

(i)は直接吸収則の形をしていないが、(ii)が成り立つならば、$\exists u\ b=au$ となることが resultant 定理によって簡単にわかるので、(i)が $a+au=a$ という吸収則をあらわしているといえる。

簡化則はさらに複雑である。これも一般にパターンマッチングによらねばならないが、次の、釈迦の掌定理（仮称、適切な名を御存知の方、お教え下さい）と吸収則のアルゴリズムを用いて実現可能である。

[定理 10] 釈迦の掌定理

$f(x_1, x_2, \cdots, x_n)$ を n 変数ブール関数，$r(x_{\sigma(1)}, x_{\sigma(2)}, \cdots, x_{\sigma(m)})$ をその m 変数 resultant のひとつとすれば、$f \equiv f+r$ ∎ (証明 略)

$f(x,y) = x + x'y$ のとき、$f_y(y|x) = y$, そこで $f(x,y) = f(x,y) + f_y(y|x) = x + x'y + y = x + y$ $x'y + y$ に吸収則を適用した。

この方法を徹底的にやれば、いわゆるブール関数の素項展開ができることになる。

ここまでやれば当初望んでいた最簡形にかなり近いところまで、式の表現が改善されたことになろう。しかし現在の Syllogister では吸収則の適用までで、resultant はまだとれず、簡化則も未装備である。

3.3. LISP による式の表現と処理の概要

変数と定数 0，1 を含み、+と・，' とで構成されるブール式を考える。LISP で扱うために、このブール式は前置記法のリスト表現に変換される。このとき、変数はそのまま文字アトムに、定数 1 は数値アトム 1 に 0 はリスト (NOT 1) に、+，・，' はそれぞれアトム OR, AND, NOT に変換される。例えば $x' + (y \cdot z + 0 + u)'$ は、S式の (OR (NOT X) (NOT (OR (AND Y Z) (NOT 1) U))) に変換される。ブール式の処理の核になるものは、この式をしかるべき整列を施した積和形にする LISP 関数 bcompute である。この関数は引数がアトムなら何もしない。そうでなく演算子が NOT ならば、その被演算項 (opr) を関数 bnot にわたす。

演算子が AND (OR) のときは、その被演算項のリスト (oprl) の各要素を bcompute した結果のリストを関数 band (bor) にわたす。さもなくばエラーである。演算子が NOT の場合にその opr を先に bcompute しないのは、$(x')' \to x$ となって処理が早くなる可能性を利用したかったからである。

bnot は引数がアトム α のときは、(NOT α) を返す。引数が (NOT α) のときは、α を bcompute した結果を返す。演算子が AND や OR のときは、

ド・モルガン則に対応する演算をする。関数 bor は、和の結合変換則、恒等則、補元則、帰無則、巾等則等によって式を簡単化し、しかるべき順序によって項を整列する。吸収則、簡化則は適用しない。また分配則も積和形を得るのが目的であるから、当然適用しない。関数 band は bor と同様の規則の適用によって式を簡単化する他、積の分配則を適用する。(補助関数 band2 による) 項の整列順序は、正確には付録の関数 compare1 を見ていただけばよいが、おおむね次の順序である。

1 < 1' < リテラル < 非リテラル。2つの変数 x, y の順位は線形全順序なら何でもよい。その順位をもとにして、x < y のとき、x < x' < y < y' と定める。非リテラル式については、' の式 < 積の式 < 和の式 とする。ただし、' の式を ξ' とするとき、ξ と η とを比較して等しいときは η < ξ' とする。各「' の式」や「積の式」,「和の式」の中では、今までの定義の辞書式帰納的適用によって定まる順位とする。

吸収則を適用する関数 barbsorp は 3.2.5 に述べたアルゴリズムによっている。

また、resultant の補助関数としても用いるための、置換関数 bsubstitute も付録に収めてある。bsubstitute [e; a] においてeは置換の対象となる式、aは置換対のリストで、置換対は点対 (<変数アトム>. <ブール式>) で、おきかえた(同時置換)ブール式を値とする関数である。処理自身は LISP の eval [e; a] と似たようなものである。

4 あとがき

何だかんだと resultant の有難さを吹聴して回ったあげくに、そんなに役に立つというのなら、ひとつ実際にその resultant を作って実証してみろという人の口車に乗って、ブール式の処理をやり始めたが、核になる部分だけを作るのは、意外に簡単であった。これは LISP の記述力に大いに依拠している。しかしアルゴリズムはわかり易い簡明直截版?であるから、スピードの点は全く保証しない。(スピードアップより先ず正しいプログラムを!)

しかし数式処理システムというからには、記述言語を明確にして、自由にプログラミングできるようにしなくてはならない。したがって言語処理系を作成する必要がある。これは大変だし、あまり考えなしに言語を決めると、あとで動きがとれなくなる可能性があるので、当面は処理機能の拡充を図ることが先決と考えている。

謝辞 日頃御指導いただく北川敏男所長に感謝いたします。また、熱心に討論して下さった、国藤進、古村光夫、田中啓夫、安達統衛の各氏に御礼申し上げます。

文献

(i) 竹島卓: 関係データベースにおける関数従属関係のブール方程式を用いる方法について, DBMS 6-2 (1978). (ii) 山田欽一: カッコいい代数, 数セミ (1966頃). (iii) 今井正治他: ハッシング技法を用いたアルゴリズムとその擬似ブール計画法への応用, 情報処理 V.18, no.7 (iv) 岡田康治他: Clause Selection 法による論理関数の素項生成, 信学技法 EC-74-37 (v) Codd, E.F.: Relational Completeness of Data Base Sublanguages, Data Base Systems (1971).

付録　Syllogisterの核関数

```
(BCOMPUTE (LAMBDA (X)
 (COND ((ATOM X) X)
       ((EQ (OPE X) (QUOTE NOT)) (BNOT (OPR X)))
       ((EQ (OPE X) (QUOTE AND))
        (BAND (MAPCAR (OPRL X) (FUNCTION BCOMPUTE))) )
       ((EQ (OPE X) (QUOTE OR))
        (BOR (MAPCAR (OPRL X) (FUNCTION BCOMPUTE))) )
       (T (BERROR (LIST (QUOTE ***BCOMPUTE? *) X))) ) ))
(BNOT (LAMBDA (X)
 (COND ((ATOM X) (LIST (QUOTE NOT) X))
       ((EQ (OPE X) (QUOTE NOT)) (BCOMPUTE (OPR X)))
       ((EQ (OPE X) (QUOTE AND)) (BOR (MAPCAR (OPRL X) (FUNCTION
          (LAMBDA (X) (BCOMPUTE (LIST (QUOTE NOT) X))) ))))
       ((EQ (OPE X) (QUOTE OR)) (BAND (MAPCAR (OPRL X) (FUNCTION
          (LAMBDA (X) (BCOMPUTE (LIST (QUOTE NOT) X))) ))))
       (T (BERROR (LIST (QUOTE ***BNOT? *) X))) ) ))
(BOR (LAMBDA (X)
 (COND ((NULL (CDR X)) (CAR X)) (T (BOR2 (CAR X) (BOR (CDR X))))) ))
(BOR2 (LAMBDA (X Y)
 (COND ((EQ (OPE X) (QUOTE OR))
        (COND ((EQ (OPE Y) (QUOTE OR))
               (BORO (ORRESOLVE (OPRL X) (OPRL Y))) )
              (T (BORO (ORRESOLVE (OPRL X) (LIST Y)))) ) )
       ((EQ (OPE Y) (QUOTE OR))
               (BORO (ORRESOLVE (LIST X) (OPRL Y))) )
       (T      (BORO (ORRESOLVE (LIST X) (LIST Y)))) ) ))
(BORO (LAMBDA (X)
 (COND ((NULL (CDR X)) (CAR X)) (T (CONS (QUOTE OR) X))) ))
(ORRESOLVE (LAMBDA (X Y)
 (COND ((NULL X)
        (COND ((NULL Y) (QUOTE ((NOT 1)))) (T Y)) )
       ((NULL Y) X)
       (T ((LAMBDA (Z) (COND
   ((EQ Z (QUOTE BEQL)) (ORRESOLVE2 (CAR X) (ORRESOLVE
                        (CDR X) (CDR Y)) ))
   ((EQ Z (QUOTE BCMP)) (QUOTE (1)))
   ((EQ Z (QUOTE BLES)) (ORRESOLVE2 (CAR X) (ORRESOLVE
                        (CDR X) Y) ))
   (T (ORRESOLVE2 (CAR Y) (ORRESOLVE X (CDR Y)))) ))
           (COMPARE2 (CAR X) (CAR Y)) )) ) ))
(ORRESOLVE2 (LAMBDA (X Y)
 (COND ((ONEP X) (QUOTE (1)))
       ((EQUAL X (QUOTE (NOT 1))) Y)
       ((ONEP (CAR Y)) (QUOTE (1)))
       ((EQUAL (CAR Y) (QUOTE (NOT 1))) (LIST X))
       (T (CONS X Y)) ) ))
```

```
(BAND (LAMBDA (X)
 (COND ((NULL (CDR X)) (CAR X)) (T (BAND2 (CAR X) (BAND (CDR X))))) ))
(BAND2 (LAMBDA (X Y)
 (COND ((EQ (OPE X) (QUOTE OR))
        (COND ((EQ (OPE Y) (QUOTE OR))
               (BOR (MAPCAR2 (OPRL X) (OPRL Y)
 (FUNCTION (LAMBDA (U V) (BANDO (ANDRESOLVE (BANDOPR U)
                                            (BANDOPR V)))) )) )
              (T (BOR (MAPCAR (OPRL X)
 (FUNCTION (LAMBDA (U) (BANDO (ANDRESOLVE (BANDOPR U)
                                          (BANDOPR Y)))) ))) ) )
       ((EQ (OPE Y) (QUOTE OR)) (BOR (MAPCAR (OPRL Y)
 (FUNCTION (LAMBDA (V) (BANDO (ANDRESOLVE (BANDOPR X)
                                          (BANDOPR V)))) )))
       (T (BANDO (ANDRESOLVE (BANDOPR X) (BANDOPR Y)))) ) ))
 (BANDO (LAMBDA (X)
 (COND ((NULL (CDR X)) (CAR X)) (T (CONS (QUOTE AND) X))) ))
(BANDOPR (LAMBDA (X)
 (COND ((EQ (OPE X) (QUOTE AND)) (OPRL X)) (T (LIST X))) ))
(ANDRESOLVE (LAMBDA (X Y)
 (COND ((NULL X) (COND ((NULL Y) (QUOTE (1))) (T Y)))
       ((NULL Y) X)
       (T ((LAMBDA (Z)
   (COND ((EQ Z (QUOTE BEQL))
          (ANDRESOLVE2 (CAR X) (ANDRESOLVE (CDR X) (CDR Y))) )
         ((EQ Z (QUOTE BCMP)) (QUOTE ((NOT 1))))
         ((EQ Z (QUOTE BLES))
          (ANDRESOLVE2 (CAR X) (ANDRESOLVE (CDR X) Y)) )
         (T (ANDRESOLVE2 (CAR Y) (ANDRESOLVE X (CDR Y)))) ) )
           (COMPARE2 (CAR X) (CAR Y)) ) ) ))
(ANDRESOLVE2 (LAMBDA (X Y)
 (COND ((ONEP X) Y)
       ((EQUAL X (QUOTE (NOT 1))) (QUOTE ((NOT 1))))
       ((ONEP (CAR Y)) (LIST X))
       ((EQUAL (CAR Y) (QUOTE (NOT 1))) (QUOTE ((NOT 1))))
       (T (CONS X Y)) ) ))
```

```
(COMPARE1 (LAMBDA (X Y)
 (COND ((AND (ATOM X) (ATOM Y)) (COMPATOM1 X Y))
       ((EQ (OPE X) (OPE Y)) (COMPOPRL1 (OPRL X) (OPRL Y)))
       ((EQ (OPE X) (QUOTE NOT))
        ((LAMBDA (Z) (COND ((EQ Z (QUOTE BEQL)) (QUOTE BGTR))
                           (T Z) ))
          (COMPARE1 (OPR X) Y) ) )
       ((EQ (OPE Y) (QUOTE NOT))
        ((LAMBDA (Z) (COND ((EQ Z (QUOTE BEQL)) (QUOTE BLES))
                           (T Z) ))
          (COMPARE1 X (OPR Y)) ) )
       ((ATOM X) (QUOTE BLES))  ((ATOM Y) (QUOTE BGTR))
       ((EQ (OPE X) (QUOTE AND)) (QUOTE BLES))
       ((EQ (OPE Y) (QUOTE AND)) (QUOTE BGTR))
       (T (BERROR (LIST (QUOTE ***COMPARE1? *) X Y))) ) ))
(COMPARE2 (LAMBDA (X Y)
 (COND ((AND (ATOM X) (ATOM Y)) (COMPATOM1 X Y))
       ((EQ (OPE X) (OPE Y)) (COMPOPRL1 (OPRL X) (OPRL Y)))
       ((EQ (OPE X) (QUOTE NOT))
        ((LAMBDA (Z) (COND ((EQ Z (QUOTE BEQL)) (QUOTE BCMP))
                           (T Z) ))
          (COMPARE1 (OPR X) Y) ) )
       ((EQ (OPE Y) (QUOTE NOT))
        ((LAMBDA (Z) (COND ((EQ Z (QUOTE BEQL)) (QUOTE BCMP))
                           (T Z)))
          (COMPARE1 X (OPR Y)) ) )
       ((ATOM X) (QUOTE BLES))  ((ATOM Y) (QUOTE BGTR))
       ((EQ (OPE X) (QUOTE AND)) (QUOTE BLES))
       ((EQ (OPE Y) (QUOTE AND)) (QUOTE BGTR))
       (T (BERROR (QUOTE ***COMPARE2? *) X Y))) ) ))
(COMPATOM1 (LAMBDA (X Y)
 (COND ((EQ X Y) (QUOTE BEQL))
       ((ONEP X) (QUOTE BLES))
       ((ONEP Y) (QUOTE BGTR))
       ((ORDERP X Y) (QUOTE BLES))
       (T (QUOTE BGTR)) ) ))
(COMPOPRL1 (LAMBDA (X Y)
 (COND ((NULL X) (COND ((NULL Y) (QUOTE BEQL)) (T (QUOTE BLES))))
       ((NULL Y) (QUOTE BGTR))
       (T ((LAMBDA (Z) (COND ((EQ Z (QUOTE BEQL))
                              (COMPOPRL1 (CDR X) (CDR Y)) )
                             (T Z) ))
           (COMPARE1 (CAR X) (CAR Y))) ) ) ))
```

BEQL, BGTR, BLES はそれぞれ 2つのブール式 t_1 と t_2 との順位が等い, $t_1 > t_2$, $t_1 < t_2$ の意味.

BCMP は t_1 と t_2 が互に他の補元をあらわす式のとき.

```
(BABSORP (LAMBDA (E)
  (COND ((EQ (OPE E) 'OR¥) ((LAMBDA (Z)
          (COND ((NULL (CDR Z)) (CAR Z))
                (T (CONS OR¥ Z)) ) ) (BABSORP2 NIL (CDR E)) ))
        (T E) ) ))

(BABSORP2 (LAMBDA (LIN RR)
   (COND (RR (BABSORP3 NIL (CAR RR) (CDR RR)))
         (T (REVERSE LIN)) ) ))
(BABSORP3 (LAMBDA (MIN A R)
   (COND (R (COND ((BZEROP (BCOMPUTE
                             (LIST AND¥ (LIST NOT¥ A) (CAR R)) ))
                   (BABSORP3 MIN A (CDR R)) )
                  ((BZEROP (BCOMPUTE
                             (LIST AND¥ A (LIST NOT¥ (CAR R))) ))
                   (BABSORP2 LIN (CDR RR)) )
                  (T (BABSORP3 (CONS (CAR R) MIN) A (CDR R))) ))
          (MIN (BABSORP2 (CONS A LIN) (REVERSE MIN)))
          (T (REVERSE (CONS A LIN))) ) ))
(BZEROP (LAMBDA (E) (EQUAL E (QUOTE (NOT 1))) ))


(BSUBSTITUTE (LAMBDA (E A)
   (COND ((OPE E) (CONS (OPE E) (MAPCAR (OPRL E)
                 (FUNCTION (LAMBDA (X) (BSUBSTITUTE X A))) )))
         (T (BVALUE E A)) ) ))
(BVALUE (LAMBDA (E A)
   ((LAMBDA (Z) (COND (Z (CDR Z)) (T E))) (ASSOC E A)) ))


(MAPCAR2  (LAMBDA (X Y F)
  (COND ((NULL X) NIL)
        (T (APPEND (MAPCAR  Y
    (FUNCTION (LAMBDA (YY) (F (CAR X) YY))) )
                   (MAPCAR2 (CDR X) Y F) )) ) ))


(OPE (LAMBDA (X) (COND ((ATOM X) NIL) (T (CAR X)))))
(OPR (LAMBDA (X) (CADR X)))
(OPRL (LAMBDA (X) (CDR X)))
```

NOT¥, AND¥, OR¥ は大域変数で, その値はそれぞれアトム NOT, AND, OR になっている.