システムＬＳＩ設計技術　100－2
（２００１．２．９）

大規模論理関数簡約化専用プロセッサの基本設計

金杉　昭徳
埼玉大学　工学部　電気電子システム工学科

〒338-8570　浦和市下大久保255
TEL/FAX 048-858-3473
E-mail: kanasugi@ees.saitama-u.ac.jp

あらまし　従来，論理関数の簡約化は論理機能を回路に実装する際に，回路規模を縮小することを目的に行われている．そのため対象とする論理関数の規模はあまり大きくない．それに対し本論文では，極めて大規模な論理関数の演算を高速に実行する必要性から，論理関数の簡約化を行うものである．目標は，論理変数２千，論理項百万程度の大規模な論理関数を実時間で近似的に簡約化することである．

本論文では，まず本プロセッサが必要とされる背景と仕様について概説し，さらに基本的な演算アルゴリズムを述べる．続いてアーキテクチャ，さらにアルゴリズムに関しての計算機実験結果を示す．

キーワード　論理関数の簡約化，専用プロセッサ，ラフ集合

A Basic Design of Processor for Large-Scale Logic Reduction

Akinori KANASUGI
Department of Electrical and Electronic Systems, Saitama University

255 Shimo-okubo, Urawa, 338-8570 Japan
TEL/FAX 048-858-3473
E-mail: kanasugi@ees.saitama-u.ac.jp

Abstract　The purpose of conventional logic reduction techniques is to reduce the size of logic gates. Therefore, the scale of the logic function is not large. This paper proposes a basic design of processor that reduces large-scale logic functions. In this paper, the basic architecture, the execution units, novel execution algorithms and simulation results are shown.par

key words　logic reduction, special processor, rough sets

1 まえがき

従来，論理関数の簡約化は論理機能を回路に実装する際に，回路規模を縮小することを目的に行われている．そのため対象とする論理関数の規模はあまり大きくない．それに対し本論文では，極めて大規模な論理関数の演算を高速に実行する必要性から，論理関数の簡約化を行うものである．目標は，論理変数2千，論理項百万程度の大規模な論理関数を実時間で近似的に簡約化することである．データマイニング，決定支援システム，機械学習への応用を考えている．

本論文では，まず本プロセッサが必要とされる背景と仕様について概説し，さらに基本的な演算アルゴリズムを述べる．続いてアーキテクチャ，さらにアルゴリズムに関しての計算機実験結果を示す．

2 背景と仕様

本プロセッサは極めて大規模な論理関数の簡約化を目的としており，具体的には論理変数の数 $N = 2 \times 10^3$，また和積表現における項数 $M = 10^6$ 程度を考えている．これは，ラフ集合プロセッサ[1][2][3]への応用を考えて設定している．ラフ集合理論は，曖昧・不完全なデータベースから規則を抽出するのに有力であり，データマイニング，決定支援システム，機械学習等，幅広い分野での応用が期待されている．ラフ集合の演算においては，決定表と呼ばれるデータベースから識別行列を求める．この行列からルールを表す論理関数を求め，さらに関数を簡約化することにより，最終的なアウトプットを得る．本格的な応用では，決定表における属性（論理変数に相当）は2千程度，また行数は百万行にも達する．百万行の決定表から導かれる識別行列は，その大きさが百万行×百万列で，かつ各要素には2千個の論理変数を含むことになる．しかしながら詳細は省略するが，実際には百万行×1列（実質的には2千列）の行列が取り扱えれば対処できる．そこで本プロセッサは，上記の仕様（$N = 2 \times 10^3$，$M = 10^6$）を採用する．なお，論理式の簡約化は1度行えば済むというものではない．論理関数は頻繁に更新され，その度に論理式の簡約化を行い，常に最新のルールを用意して判断を下す（論理式の値を評価する）ような系を対象にしている．したがって，論理式の簡約化のための処理時間は1秒程度を目標とする．ただし，精度はあまり要求されず，原関数で求めた値と簡約化後の近似関数で求めた値が90%程度の確率で一致することを目標とする．

3 論理関数の簡約化

3.1 表記

対象とする論理関数は和積形式とし，

$$f = \bigwedge_{i=0}^{M-1} \left\{ \bigvee_{j=0}^{N-1} \left(a_{ij} \bigwedge x_j \right) \right\} \tag{1}$$

とする．ここで記述を簡単にするために，便宜的に，

$$f = A \cdot \boldsymbol{x} \tag{2}$$

$$\boldsymbol{x} = [x_0, x_1, \cdots, x_{N-1}]^T \tag{3}$$

$$A = \begin{bmatrix} a_{00} & a_{01} & \cdots & a_{0,N-1} \\ a_{10} & a_{11} & \cdots & a_{1,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{M-1,0} & a_{M-1,1} & \cdots & a_{M-1,N-1} \end{bmatrix} \tag{4}$$

と表記する．通常の行列表記とは意味が異なることに注意されたい．例えば，

$$f = (x_0 + x_1 + x_3) \cdot (x_0 + x_2 + x_3) \cdot (x_1 + x_3) \tag{5}$$

なる論理関数を

$$f = A \cdot \boldsymbol{x} \tag{6}$$

$$\boldsymbol{x} = [x_0, x_1, x_2, x_3]^T \tag{7}$$

$$A = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \tag{8}$$

と表す．

このように論理関数 f は，係数行列 A によって決定される．ただし $a_{ij} = \{0, 1\}$．

3.2 前処理

本論文で扱う論理関数は極めて大規模なために，前処理により簡約化を行う．原理は簡単であり，x，y を論理変数とするとき，

$$\begin{aligned} x(x + y) &= x + xy \\ &= x \end{aligned} \tag{9}$$

が成り立つことを用いる．

具体例を示す．式(5)の第3項を，

$$x = x_1 + x_3 \tag{10}$$

とおけば，第1項は，

$$x_0 + x_1 + x_3 = x + x_0 \tag{11}$$

となる．したがって，式(9)の左辺と同じ形になるので，式(5)の第1項は省略できる．このような考え方により関数 f を簡約化する．

実際の手順としては，まず論理変数の少ない項を見つける．すなわち係数行列 $\boldsymbol{A}$ の各行

$$\boldsymbol{a_i} = [a_{i0}, a_{i1}, \cdots, a_{i,N-1}] \tag{12}$$

に含まれる“1”の数が少ないものを見つける．これを K 個（16〜64程度）探索し，選ばれた行ベクトルを $\boldsymbol{c_k}$ $(k = 0 \sim K-1)$ とする．ここで $\boldsymbol{c_k}$ は互いに含意関係にならないようにする．このベクトルを用いて，$\boldsymbol{A}$ の各行が消去できるか調べる．すなわち，

$$\boldsymbol{c_k} \subset \boldsymbol{a_i} \tag{13}$$

が成り立てば，$\boldsymbol{a_i}$ を消去できる．ただし，$\boldsymbol{c_k}$ をすべてチェックしなくてもよい場合もある．例えば，

$$\boldsymbol{c_2} \subset \boldsymbol{a_i} \tag{14}$$

が成立すれば，$\boldsymbol{c_3} \sim \boldsymbol{c_{K-1}}$ は調べなくてよい．

上式の判定は，全ての j $(= 0 \sim N-1)$ について

$$c_{kj} \subset a_{ij} \tag{15}$$

が成立することが必要である．しかし逆に言えば，$j = p$ において上式が不成立となったら，$j = p+1$ 以降はチェックしなくてよい．また含意演算は，

$$\overline{c_{kj}} + a_{ij} \tag{16}$$

で表される．そこで $\boldsymbol{c_k}$ の全ビットをあらかじめ反転させておき，これを $\boldsymbol{d_k} = \overline{\boldsymbol{c_k}}$ とおけば，

$$d_{kj} + a_{ij} \tag{17}$$

というOR演算で結果が求まる．

3.3 主処理

本論文では近似を許すことにより，極めて大規模な論理関数を高速に簡約化することを目的とする．その主処理の原理を以下に述べる．

論理式 f を積和形式に展開すると，

$$f = \bigvee_i b_i x_i + \bigvee_{ij} b_{ij} x_i x_j + \bigvee_{ijk} b_{ijk} x_i x_j x_k + \cdots \tag{18}$$

という形になる．この関数において低次な項ほど関数 f における重要な項と考えられる．そこでこのような項を探索し，それらの論理和をとることにより，元の論理式を近似する．その具体的な方法は次の通りである．

論理関数 f（すなわち $\boldsymbol{A}$）の中で最も頻繁に出現する論理変数を見つけ，それを仮に x_{m_1} とする．そして f を2つの論理関数の積に分解する．

$$f = f_{m_1} \cdot f_1 \tag{19}$$

ここで，

f_{m_1} ： x_{m_1} を含む項の論理積

f_1 ： x_{m_1} を含まない項の論理積

同様にして f_1 の中で最も頻繁に現れる論理変数を見つけて x_{m_2} とし，f_1 の中で x_{m_2} を含む項の積を f_{m_2}，x_{m_2} を含まない項の積を f_2 とすると，

$$f = f_{m_1} \cdot f_{m_2} \cdot f_2 \tag{20}$$

同様の操作を繰り返し，関数 f_L が全て変数 x_{m_L} を含むとすれば，

$$\begin{aligned} f &= f_{m_1} \cdot f_{m_2} \cdots f_{m_L} \\ &= \bigwedge_{l=1}^{L} f_{m_l} \end{aligned} \tag{21}$$

と書ける．このとき f は論理項

$$x_{m_1} \cdot x_{m_2} \cdots x_{m_L} \tag{22}$$

を必ず含む．したがって，なるべく L の小さな組み合わせを見つければ，f における重要な項になっている．

また最初の論理変数 x_{m_1} を変更すれば，以降の論理変数の組み合わせも異なってくる．したがって初期段階での変数 x_{m_1}，x_{m_2}，x_{m_3} を何種類か組み合わせることにより，有力な論理項を見つけることが可能になる．これらをツリー表現したものを図1に示す．

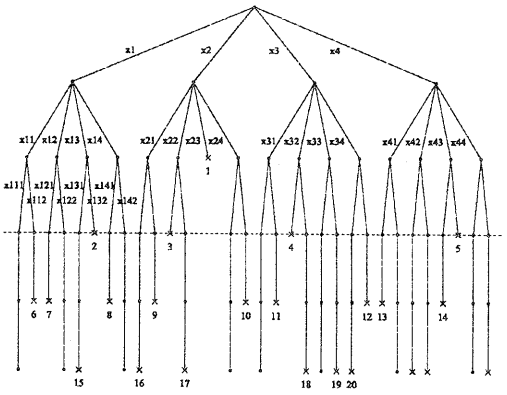

図 1: 論理関数の近似積和展開

次に，以上の操作を係数行列 A を用いて行う手順を示す．まず A の各列に含まれる “1” の個数をカウントする．次に最も “1” を多く含む列を見つける．その変数を x_{m_1} とし，$a_{i,m_1} = 1$ を満たす行を A から削除する．さらに m_1 列も A から削除する．このようにして得られた行列を A_1 とする．同様にして，A_1 の中で最も多く “1” を含む変数を見つけ，それを x_{m_2} とし，$a_{i,m_2} = 1$ を満たす行を A_1 から削除する．さらに m_2 列も A_1 から削除する．このようにして得られた行列を A_2 とする．以上の操作を A が空になるまで繰り返す．

ハードウェア化に際しては，最も多く “1” を含む列を探す方法がポイントになる．高速でしかもハードウェア量が少ないことが望ましい．単純に考えれば，各列の “1” の個数をカウンタで数え，その結果をコンパレータで比較すればよい．しかしながら本プロセッサは極めて大規模な論理関数を扱うので処理速度の面で問題がある．

いま，1 つのデータをカウントするのに要する時間を t_c とすれば，全データのカウントには，

$$T_c = t_c \times M \times N \qquad (23)$$

だけの時間を要する．仮に，$t_c = 10$ [ns] としても，

$$T_c = 10 \times 10^{-9} \times 10^6 \times 2 \times 10^3 = 20[s] \qquad (24)$$

を要する．図 1 から明らかなように，f の展開には，この操作を数十〜数百回行う必要がある．したがって，上式の値を 3〜4 桁下げる必要がある．そのために処理の並列化とデータの間引きを行う．詳細は次章で述べる．

4 ハードウェアの基本設計

提案するプロセッサのブロック・ダイアグラムを図 2 に示す．プロセッサ内部で特徴的なユニットは，

(a) コア・セレクタ
(b) カバリング・ユニット
(c) 再構成ユニット

である．

実装形態としては，ホスト・コンピュータに付加するサブ・システムとし，将来的には複数のプロセッサをパラレルで動作させ，高速化を図る．

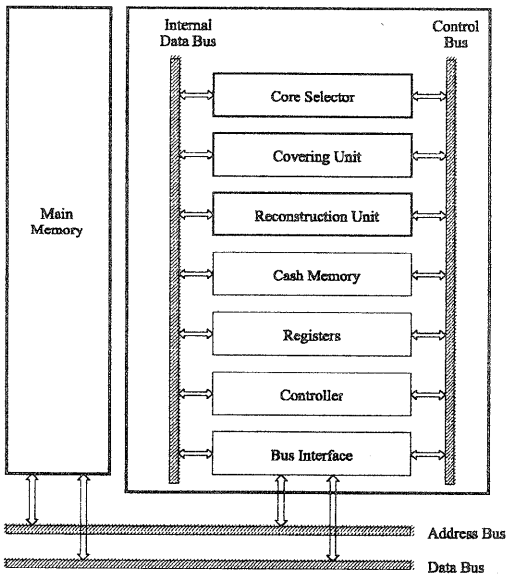

図 2: ブロック・ダイアグラム

本章では，各ユニットの内部構成，データ形式，演算アルゴリズムについて述べる．

4.1 入力データ形式

入力データ形式を図3に示す．各フィールドの意味は以下の通りである．

(a) Data：各論理変数の有無（各 1 bit，計 2,032 bit）
(b) Sum：Data フィールド内の "1" の総和（11 bit）
(c) Flag：他のデータによって含意されるか否か（1 bit，詳細は後述）

256B で係数行列の 1 行を表現するので，100 万項で 256MB を要する．またこの形式から，扱える論理変数の正確な数は，$N = 2{,}032$ とする．

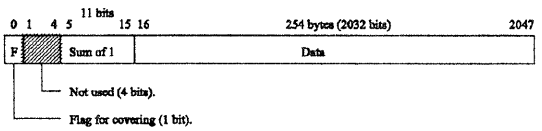

図 3: 入力データ形式

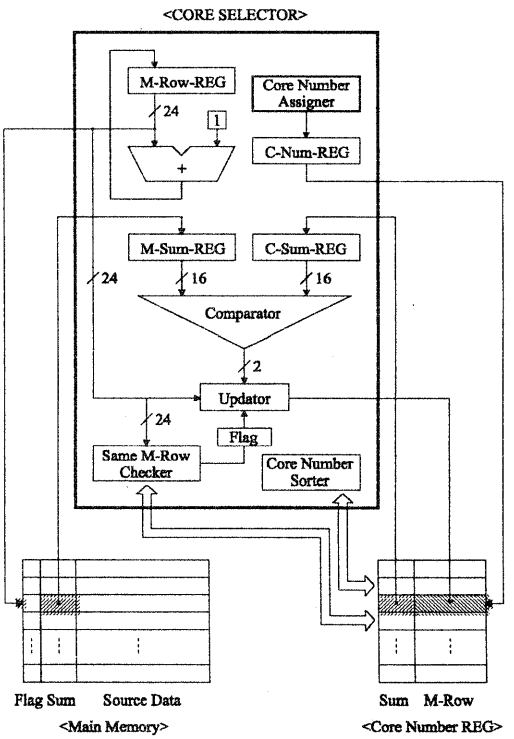

図 4: コア・セレクタ

4.2 コア・セレクタ

本ユニットは主記憶のデータの中から，Sum の値（"1" の総和）が小さいものを選び（これをコア・データと呼ぶことにする），その番号を専用のレジスタ（Core Number REG）に格納する．なお，Sum 値はデータ格納時にチェックし，予め設定しておく（本ユニット内では計算しない）．なお，Core Number REG は 32 本程度用意する．この本数が少ないと次節で述べるカバリングが不十分になり，不要なデータを十分に取り除けない．一方，この本数が多すぎると含意判定に時間を要し，回路規模も増大する．したがってトレード・オフをシミュレーション等により十分に検討する必要がある．図4にコア・セレクタのブロック・ダイアグラムを示す．

4.3 カバリング・ユニット

コア・データを用いて，入力データを簡約化する．すなわち含意演算により入力データを間引く．結果は各入力データの先頭ビットのフラグ（F）に格納する．残しておく場合には "1"，消去する場合は "0" とする．

処理は，256 ビット長単位で行い（64 ビット長の含意判定ユニットを 4 つ用意する），1 データ（2032 ビット長）について 8 回繰り返す[1]．途中で含意条件が満たされなかった場合は，残りの判定処理をスキップすることにより，処理を高速化する．この処理を Core Number REG のデータ全てに対して行うが，含意条件が満たされた時点で他はスキップして高速化する（図5）．なお，Core Number REG のデータはコア・キャッシュ（容量は 256B が 32 本で，8KB）に予めストアしておく．この際にビットを反転しておく．図6にカバリング・ユニットのブロック・ダイアグラムを示す．

なお，本プロセスの終了後に，フラグ・ビットが "1" のデータをメイン・メモリの先頭に集め，次段の処理を簡約化すると共に，作業領域を確保する．

[1] ただし，先頭の 2 バイトはフラグと Sum が格納されているので，処理の対象外とする．

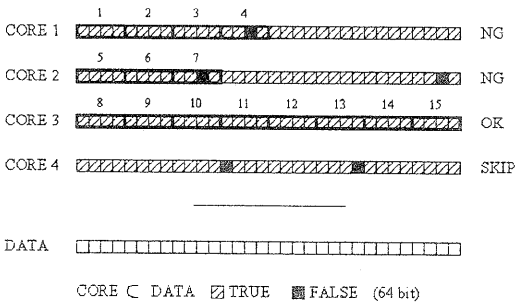

図 5: 含意条件判定処理の高速化

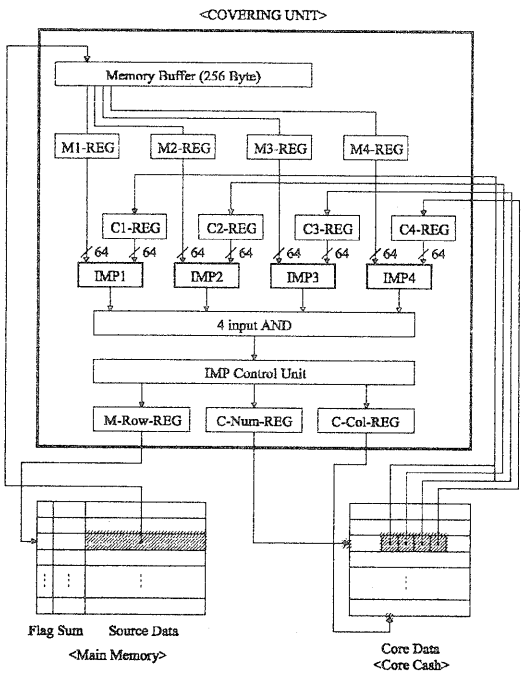

図 6: カバリング・ユニット

4.4 再構成ユニット

本ユニットは，第 1 に，コア・データによってカバーされずに残ったデータを対象にして，最も頻繁に現れる論理変数を求める．第 2 に，その論理変数を元に，入力データを再構成する．

そのためにまず，各ビットごとに “1” の出現頻度をカウントする．しかしながら，目的は正確な出現回数ではない．そこで高速化と回路規模の縮小を目的とした工夫を行う．すなわち，

(a) カウント対象データの間引き
(b) カウンタビット長の短縮

である．

(a) は，全てのデータを対象とするのではなく，20%程度のデータをランダムに選んでカウントするものである．これにより 5 倍の高速化が達成される．

(b) は，本来 20 ビット長のカウンタを必要とするところを，8 ビット程度の短いビット長のカウンタで代用するものである．このためオーバーフローが生じる度に，全てのカウンタを 1 ビット右にシフトする．これにより，回路規模の縮小と同時に高速化が図れる．

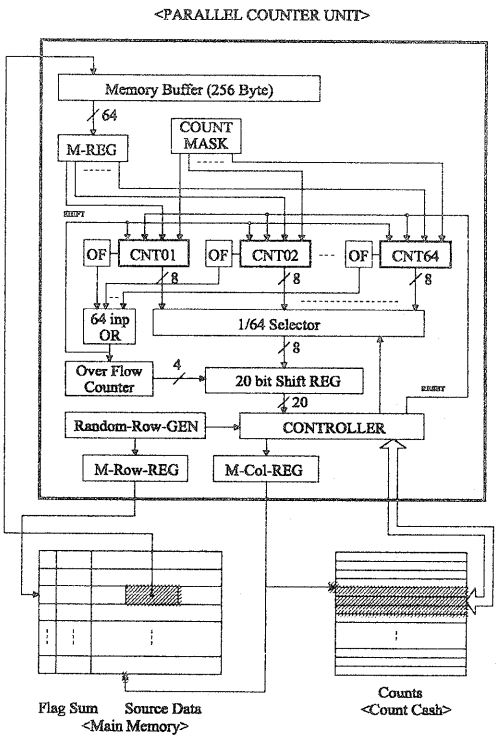

図 7: カウンタ・ユニット

図 7 にカウンタ・ユニットのブロック・ダイアグラムを示す．内部処理は 64 ビット（論理変数）毎に行う．カウント中に生じたオーバーフロー回数は，

Over Flow Counter に記憶しておき，各論理変数の出現回数を Count Cash に書き込む際に，このカウンタ値だけ，左にシフトする．Count Cash は，20bit × 2,032 本で約 6KB である．

各論理変数の出現度をカウントした後，どの論理変数が支配的なのかを決定する．この際，必ずしも正確に最大値を見つける必要はない．そこで簡易的な方法として，Count Cash 内の最上位ビットをアドレスの上位から下位にスキャンしてゆき，最初に "1" が見つかった時点で，そのデータを最大値と見なす技法を用いる（図 8）．もし，全ての最上位ビットが "0" であれば，1 つ下のビットを同様にスキャンしてゆく（Max A）．もし，精度を上げたければ，最初は，"11" という 2 ビットをチェックしてゆき（Max B），それが見つからなければ，"10" を探すという風に拡張することも可能である．

このような技法の有効性を確認するために計算機実験を行った結果を図 9 に示す．同図の横軸は論理変数で 32 分割してあり，縦軸は "1" の出現確率である．各論理変数毎に "1" の出現確率を直線的に変えてあり，データ件数は 100 万件である．(a) は 10bit，(b) は 8bit，(c) は 6bit の短縮カウンタを用いており，直線に近いほど精度が高いと見なせる．また最大値の検索には，最上位 1bit だけをチェックした．これらの結果から，6bit のカウンタでかつ簡易的な最大値の検索法を用いても実用になると考えられる．

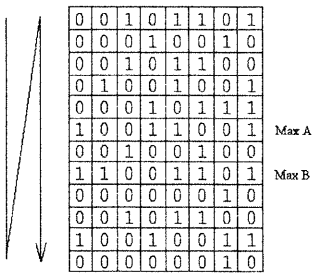

図 8: 最大値の簡易検索

続いて，入力データの再構成を行う．これは，3.3 で述べた行列の処理を行う．

4.5 ユニットの割り当て

本プロセッサでは，再構成ユニットを複数用意して，並列動作させることを考えている．そこで，各ユニットにタスクを適切に割り当てる必要がある．目的とする論理式の項数を 20 と設定しているので，まず図 1 の破線に示すように，32 項の論理式（未完成）を求める．初段は単一のプロセッサ，2，3 段は 4 つのプロセッサを並列に動作させる．各項は，$x_i \wedge x_{ij} \wedge x_{ijk}$ (i, j = 1〜4, k = 1,2) で与えられる．これを出発点として，ユニットを割当て論理式を求める．ユニットは図 10 に示すように，横方向に割り当てる．これにより，単純なアルゴリズムで，なるべく浅い論理項を見つけるようにする．図 1 において，"×" は終端している論理項，"·" は終端していない論理項である．1〜20 の順に論理項を拾ってゆく．

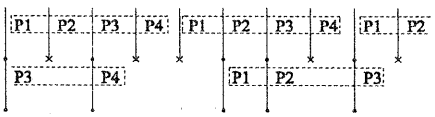

図 10: ユニットの割り当て

5 むすび

本論文では，大規模論理関数簡約化専用プロセッサの実現に向けての基本設計について述べた．目標とするプロセッサは，論理変数が 2 千，論理項が百万程度といった極めて大規模な論理関数を実時間で近似的に簡約化することを目標とする．今後はより詳細部の設計，論理合成，シミュレーション，実装と進める予定である．

謝辞

本研究を進めるにあたり，御指導・御討論頂いたワルシャワ大学教授 Andrzej Skowron 博士に心から感謝致します．

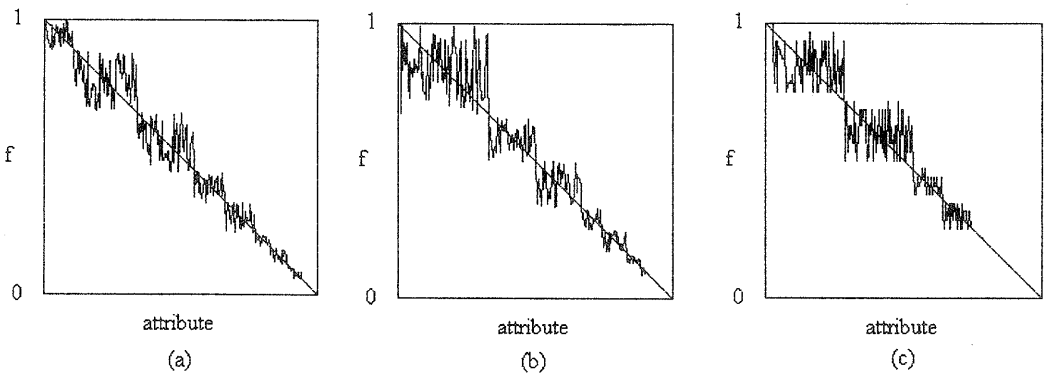

図 9: 計算機実験結果

参考文献

[1] S. K. Pal and A. Skowron : "Rough Fuzzy Hybridization – A New Trend in Decision - Making", Springer Verlag (1999).

[2] A. Kanasugi : "Rough Processor", Workshop on Rough sets – Foundations and Applications – (2000).

[3] 金杉 昭徳 : "ラフ集合専用プロセッサの設計", 人工知能学会研究会資料, SIF FAI A003 15 (2000).