設計自動化 43－2
（1988. 7. 12）

2分決定グラフを用いた論理照合アルゴリズムの評価と改良

藤田 昌宏　藤沢 久典　川戸 信明
富士通研究所

CMUのBryantにより提案された、変数順を固定した2分決定木による論理式の表現は、極めて強力であり、論理照合に応用した場合、従来手法よりかなり大規模な回路を扱える可能性がある。しかし、2分決定木の大きさは変数順に大きく依存するため、よい変数順の決定手法の開発が実用上不可欠となっている。我々は、回路図に描いた場合にネットの交差ができるだけ起きないような変数順を求めるアルゴリズムを開発し、ISCASのテスト生成用ベンチマーク回路に適用した。その結果、c6288を除いて全ての回路を従来より100倍以上高速に論理照合することができた。

Evaluation and Improvements of Boolean Comparison Method Based on Binary Decision Diagrams

Masahiro FUJITA, Hisanori FUJISAWA, Nobuaki KAWATO

FUJITSU LABORATORIES LTD.

1015 Kamikodanaka, Nakahara-ku, Kawasaki 211, Japan

Bryant of CMU proposed a method to handle logic expressions, which is based on binary decision diagrams with restriction: variable ordering is fixed throughout a diagram. Although it has much more efficiency than other methods proposed so far, the efficiency heavily depends on variable ordering. So, we developed a simple but powerful algorithm for variable ordering. The algorithm tries to find a variable ordering which is very natural, i.e., minimize the number of crosspoint of nets, when the circuit diagram is drawn. We applied this to ISCAS benchmark circuits for test pattern generation, and can verify all circuits except c6288 in one hundred times or more faster time than those reported in the literature.

1．はじめに

入力変数 X = (X1, X2, ..., Xn)から構成される２つの論理式 f1, f2 が与えられた時、全ての Xの値に対し、f1, f2が同じ出力値をとるか否かを調べる問題は、論理照合 (Boolean Comparison) と呼ばれる。これは、

・自動回路合成結果の検証
・人手修正された回路の検証
・論理回路の簡単化

等を行う際に必要であり、論理設計段階のＣＡＤシステム開発には不可欠な技術である。従来から、様々な論理照合アルゴリズムが研究・開発されてきているが、それらをまとめると以下のように分類できる。

(1) 全ての場合をシミュレーション［２３］
(2) 入力変数値で場合分けし、問題を分割して処理［８］
(3) ２つの回路の排他的論理和（またはその否定）をとりそれが１（または０）になる入力値をＤアルゴリズム（または、ＰＯＤＥＭ法）を用いて探索する［１９］
(4) ２つの論理式を積和形論理式に展開し、比較する
(5) ２つの論理式の同一性判定を数学の定理証明のように式を変形して、同じになるかで行う［１１，１８］
(6) 回路を特殊な決定木（グラフ）に変換し、変換結果を比較することで処理する［２］

いずれの手法を用いても最悪の場合には入力変数の数に対し、指数的に処理時間が増大するが、手法と対象回路の性質により、かなり大きな向き不向きが存在する。入力変数の数が少ない時には、ゲート数が大きい場合でも(1)や(2)の手法が速い。特に(1)はベクトル計算機により超高速に実行できるため、入力数が20～30程度なら最も速い。反面、入力数がそれ以上になると、現実的時間では終了しない。

手法の比較には、同じ例題の実行が必要となるが、テスト生成用のベンチマーク回路［１］がそのまま論理照合のベンチマーク例題として用いられている。この例題の回路の規模を表１に示す。

(3)の方法はかなり精力的な研究結果がある［１９］が、比較的規模の小さいc432やc880の論理照合に 3～ 4 MIPS の計算機で数時間かかっている。また、(3)の並列処理の報告［１２］もあり、ほぼ台数に比例して速度向上を達成しているが、やはり絶対的時間が長く、大きな回路は扱われていない。(2)もこの辺の事情は同じである［８、９］。

(4)は、積和形に展開する際、特に加算等の演算系の論理式は積項数が大きくなりすぎて実行できなくなることが多い。反面、ＰＬＡ等のように、もともと積和形論理式で与えられる場合には、処理時間は速い。

(5)は、扱う回路に対応する論理式を「うまく」式変形して処理するのであるが、完全自動では効率的に実行しにくい。

(6)は、CMU のBrayant が提案したものであり、回路を２分決定木に変換して比較するものである。文献［２］はアルゴリズムの提示が主であり、性能評価はあまりなされておらず、改良も含めてこれからである。特に、決定木を作る際の変数の順序付けにより、性能が大きく変わり、また、全ての順序付けを試すことは入力数が極小さい場合以外不可能なので、よい順序付けのアルゴリズムが開発が望まれている。

回路名	入力	出力	段数	ｹﾞｰﾄ数	１出力当たりの ｹﾞｰﾄ数	１出力当たりの 入力数
SN181	14	8	9	84	55	14
C432	36	7	7	203	145	36
C499	41	32	11	275	102	41
C880	60	26	24	469	130	45
C1355	41	32	24	619	322	41
C1908	33	25	40	938	557	33
C2670	233	140	32	1566	828	122
C3540	50	22	47	1741	1433	50
C5315	178	123	49	2608	937	67
C6288	32	32	124	2480	2327	32
C7522	207	108	43	3827	1096	194

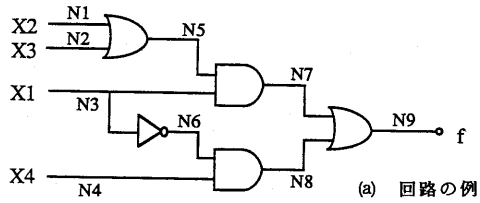
表１　ベンチマーク例題の回路規模

以上から分かるように、(6)以外では、用途を限る以外に実用的規模を扱えるものはほとんど存在しない。そこでここでは、(6)をISCAS のテスト生成用の回路に適用し、工夫を加えながら評価を行った。さらにその結果に基づき、変数の順序付けアルゴリズムを開発し、評価した。最終的に得られた論理照合プログラムの性能は、従来最も大規模論理式を取り扱えた(3)等の方法でも、数時間以上かかるか実行自体を諦めていた回路が、特殊なものを除き、数分から数十分で処理できるようになった。これは、速度比では10～100 倍以上にあたり、論理照合を行える回路規模が大幅に拡大したと言える。

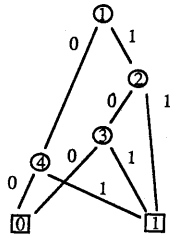
２章では、(6)のアルゴリズムの簡単な説明と、論理照合への応用法について述べる。３章では、変数順序の実行時間への影響について、ISCAS の回路の実行結果から考察する。４章では、３章の結果を踏まえた変数の順序付けアルゴリズムを示し、その実行結果を示す。５章では、変数の順序付け以外の改良について実行結果をもとに述べる。

2．グラフを用いた論理式の表現法と論理照合への応用

ここでいうグラフとは入力変数の順 (ordering) を固定した特殊な二分決定木 (Binary Decision Tree) を指す。回路図例とそれに対応するグラフを図１に示す。グラフの点 (vertex) は二種類存在し、一つは整数 index(v) で表



(a) 回路の例



(b) (a)に対応する2分決定グラフ

図1 2分決定グラフの例

される指標と2つのノード low(v) と high(v)を子としてもつようなノードvであり（図1中、○で示される）、もう一つは0または1の値 value(v) を持つノードvである（図1中、□で示される）。以下前者を節、後者を葉と呼ぶことにする。ここでvalue(v)は入力変数に対応し、ある点vの指標 index(v) は、その子の指標 index(low(v)), index(high(v)) よりも必ず小さくなくてはいけない（index(v)と入力変数の対応づけを変数順という。図1では、○の中の数字がindex を表している。また、グラフのedgeについた0、1は、それぞれ、low 、highを示す）。

vを根（root）とするようなグラフGが示す関数 f_v は次のように対応づけることができる。

1）vが葉であるような場合

a）value(v)＝1の時、$f_v = 1$

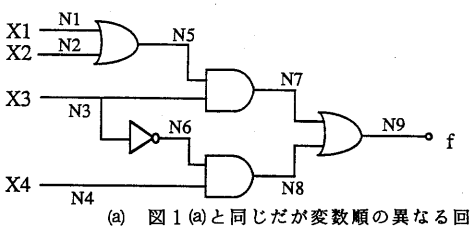
b）value(v)＝0の時、$f_v = 0$

2）vが節であり、その index(v) をiとすると

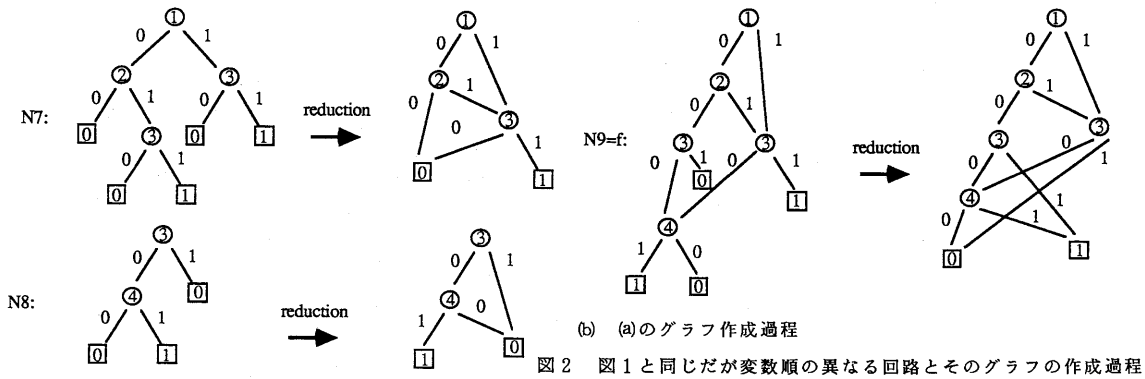
$$f_v(X_1, \cdots X_N) = \overline{X_i} \cdot f_{low(v)}(X1, \cdots, X_N) + X_i \cdot f_{high(v)}(X1, \cdots, X_N)$$

グラフでは low(v) ＝ high(v)であるような冗長な点は除去し、また2つのvとwを根とするグラフGv とGw が図形的に同型（isomorphic）な場合は1つにまとめるといった手続き（reduction)を行うことにより、同一論理を持つ回路は、同じ形のグラフに変換される。そのため2つの回路の論理的一致は、2つの回路を表すグラフの形が同型かどうかを調べることにより、検証できる。あるゲートgを出力とする回路のグラフへの変換は、そのゲートの入力側につながっているゲートを出力とした回路を表現するグラフに対して、gに対応する論理演算を施すことにより行う。その手間はグラフの大きさ（グラフの点の数の積）に比例するため、処理時間はゲートの数や種類及び入力変数の順と深く関わっている。しかし、変数順をうまく決めれば、入力変数の数にはそれほど影響を受けない為、シミュレーションや場合分けを用いた方法で扱うことが困難であるような比較的入力数の多いような回路もかなりの程度まで扱うことができる。

ここで、図2を例として、実際に入力から出力に向かってグラフを作ってみる。図2の回路は、図1の回路と同じであるが、入力変数の順序が異なる。従って、変換されたグラフは、図1のものとは異なる。グラフは入力から出力へ順に回路に沿って作っていく。まず、外部入力端子 X1, X2, X3, X4 に対するグラフを作る（ネットN1～N4）。これは、図に示すように、2つの葉をもつ3つのノードから



(a) 図1(a)と同じだが変数順の異なる回路



(b) (a)のグラフ作成過程

図2 図1と同じだが変数順の異なる回路とそのグラフの作成過程

なるグラフとなる。次にネットN5とN6を各ゲートの入力ネットに対応するグラフから、そのゲートの演算（and, or, not等）を適用することで計算する（このような操作をapply と呼ぶ）。その結果は、グラフ中に同じ形のサブグラフが存在するなど冗長なため（図２中では、⇨の左側）、reduction と呼ばれる操作で冗長性を除去する（⇨の右側）。このようなapply, reductionの処理を入力端子から順に繰り返し適用し、外部出力まで実行することで、回路全体に対応するグラフを得ることができる。図２の場合には、最終的にノードが７個のグラフになった。これは、図１のグラフのノード数６より１つ多い。このように、入力変数順によってグラフの大きさ（ノード数）が変化する。一般には、apply やreduction の処理の手間は入力となるグラフのノード数に比例するため（２つのグラフのapply の場合には、２つのグラフのノード数の積に比例）、できるだけグラフは小さくしたい。回路が大きくなると、変数順により、グラフの大きさは100 倍以上変化する。しかし、最適変数順は、現在のところ、変数の数の指数オーダの手間をかけないと判断できないので、最適性は放棄し、実用上十分よい変数順を見つけることが重要となる。次章では、ISCAS のベンチマーク例題を実行し、実際に変数順がどの程度影響を与えるかみてみよう。

３．変数順序の実行時間への影響

変数順のグラフの大きさに対する影響をみるために、変数順の決定法として、

①元の回路データの順

②人手で試行錯誤して決めた順

③乱数

の計３種類について実際にベンチマーク回路を実行して調べた。結果を表２に示す（④は４章で述べるアルゴリズムに基づくものであり、後述）。①は、ベンチマーク回路はネットリストの形で提供されているが、そのネットリストに現れる変数順をそのまま使用したものである。②は、具体的には、ネットリストを元に人が回路図を描き、その回路図中に現れた変数順を使用している。回路図を描いた人は論理回路のことを知らないので、回路の論理を追って描いたものではなく、単純に見やすいように、あるいは、より簡単になるように（つまり、ネットの交差が少ないように）描いている。③はプログラムにより乱数を発生して変数順を決めている。

表２から、乱数順では実用にならないことや、②の回路図に現れた順はかなりよいことが分かる。実際、②による結果を他の論理照合（１章の(2)や(3)の手法）の結果［８、１９］と比較すると、特に回路規模が大きい場合 100倍あるいはそれ以上高速になっていることが分かる。従って、実験結果から分かることとして、回路図を描いて、その時現れた順がよいということになるが、それはなぜであろうか。実は、以下に述べる簡単な考察により、ある程度説明することはできる。ただし、反例を作ることは容易であり、完全な説明とはなっていない。実際、回路図に現れた順がいつもよいとは限らない。ただ、実用回路で実験を行ってみると、特に他の論理照合手法と比べかなり結果がよいことだけは言える。

回路名	① original 順序		② 人手順序		③ 乱数順序		④ 図４による順序	
	処理時間 全出力	ノード数 （最大）	処理時間 全出力	ノード数 （最大）	処理時間 全出力	ノード数 （最大）	処理時間 全出力	ノード数 （最大）
SN181	8	339	7	197	8	578	6	213
C432	109	1146	53	442	unable	>100000	1423	6196
C499	928	9020	673	4661	unable	>100000	673	4661
C880	unable	>100000	85	3421	unable	>100000	55	3359
C1355	1675	9020	———	———	unable	>100000	1009	4661
C1908	965	2912	1800	4505	> 50000		778	3076
C2670	unable	>100000	———	———	unable	>100000	338	14763
C3540	unable	>100000	———	———	unable	>100000	12620	53460
C5315	2140	11807	———	———	unable	>100000	498	3441
C6288	unable	>100000	———	———	unable	>100000	unable	>100000
C7552	unable	>100000	———	———	unable	>100000	383	2096

表２　グラフに変換する方法の実行結果（SUN3/260, 24MByte,　処理時間単位秒）

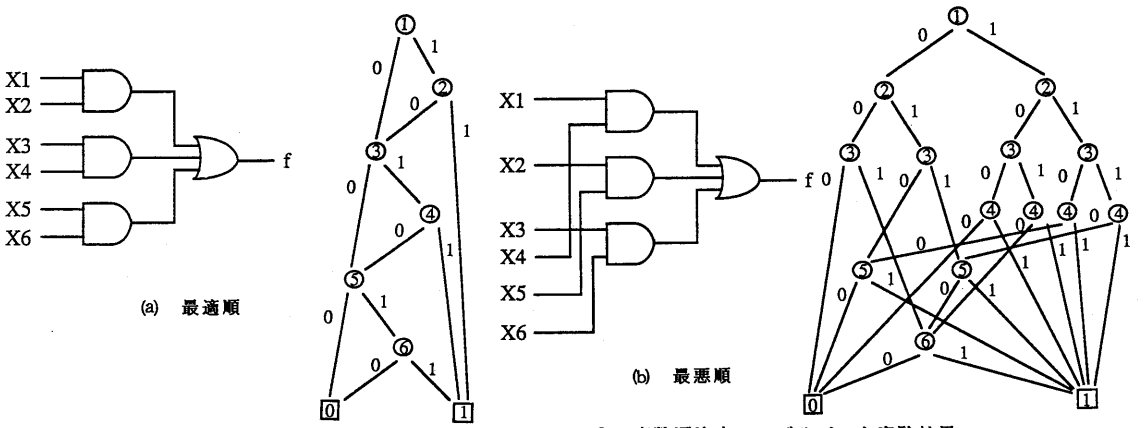


図３　変数順のグラフの大きさへの影響

回路図を描いた際に現れる変数順がよい理由を調べるのに際し、まず、図３の場合を考えてみよう。図では、X1～X6の順に変数順を決定するとする。まず、(a)のように通常回路図を描く順でグラフを作るとノード数は 8となる。これに対し、回路図にするとかなりネットが交差する順である(b)に変数順を決めグラフを作るとノード数は16となる。実は、(a)の変数順が最適であり、(b)の変数順が最悪となっている。このように、回路図をできるだけネットの交差のないように描いた時に現れる変数順はかなりよい結果を与えることが分かる（少なくともよい結果を与える場合があることが分かる）。このことは、グラフの意味を考えると容易に分かる。

ただし、単純にネットの交差が少ないような回路図中の順が最適とはいえない例も存在する。例えば、図１と図２を比べると、図２の方が回路図中の順になっているが、図１よりもグラフに変換した結果のノード数が 1つ多い。この原因を考えると、図１と図２の違いは、ネットN3の順を先にするか後にするかの差であり、N3は他のN1, N2, N4と異なり、２つのゲートの入力端子に接続されている（以降、各ネットに対し、接続されているゲートの入力端子の数をそのネットのファンアウトと呼ぶ。例えば、N1, N2, N4はファンアウトが 1であり、N3は 2である）。直観的に言って、ファンアウト数の多いネットは先に値は決まった方がより多くのネットの値が決まり、グラフが小さくなる（もちろん、反例は存在する）。

以上から、原則として、回路図を描いた時に現れるであろう順とし、ファンアウトが 1のものが 2以上のものの後にくるように変数順を決定するのがよいと言える。次章では、これをアルゴリズム化し、 ISCASベンチマーク回路に適用した結果について述べる。

３．変数順決定アルゴリズムと実験結果

前章の変数順決定の考察から図４のような変数順決定アルゴリズムを開発した。論理照合を行う際には、扱う回路規模をできるだけ小さくしたいので、通常、対象回路から各出力ごとに回路を切り出して比較する方法が取られる。従って、図４も１出力の回路を対象としている。図４はネット単位に回路を辿りながら変数順を決定していく。まず、出力端子に接続されているネットから始め、回路を出力から入力へ深さ優先で遡っていく（従って、出力端子に接続されているネットをoutnetとすると、makeOrder(outnet) として図４を実行する）。原則として、入力変数に到達するとそれを次の順の変数と決める。この単純な探索により、図３の場合は(a)の変数順が得られる。しかし、前章で述べたように、入力に接続されているネットのファンアウトも考慮する必要があるので、図４はその分、少し複雑になっている。

変数 FANOUT2UPは最も最近、回路を辿っている時に発見したファンアウト数が 2以上の入力変数に接続されているネットを記憶し、 FANOUT1LISTは変数順がまだ決まっていない変数の内、ファンアウトが 1のネットに繋がっていてかつ、回路を辿ってきた時に発見しているものが発見した順にリストの形で入る。これは、 1つのゲートの処理が終了し、 1つ出力側のゲートに戻る時（深さ優先の探索としている）に、 FANOUT2UPで示される変数の後に挿入される。但し、挿入する位置は、 FANOUT2UPで示される変数の直後ではなく、その変数の後にファンアウトが 1の変数が続いている時にはそれらの後に挿入する。

変数順決定の具体例として、図１の回路の変数順を図４に従って決定した時の実行トレースを図５に示す。最初は、出力に接続されているネットN9から始め、回路をN7, N5, N1と遡っていく（最初にN8ではなくてN7を選ぶ理由はなく、順不同であり、単に元の回路データ上で現れた順で処理する。従って、最初にN8, N4と選ぶことも考えられ、その場合には変数順は、C, D, A, Bと決定される。この場合にはグラフのノード数は 7となる）。各ネットを処理している

```
{ Initially all nets must be marked off }
procedure makeOrder(N);
begin
  foreach I ∈ Set of all input nets of the gate to which N is connected do
  begin
    if I is marked then continue;
    if I is directly connected to a primal input then
      if I is connected more than one gate then
      begin
        FANOUT2UP := I;
        if I is not in ORDER then ORDER := append(ORDER,I);
      end else FANOUT1LIST := append(FANOUT1LIST, I);
    else makeOrder(I);
  end;
  if FANOUT2UP <> undef then
  begin
    Insert FANOUT1LIST into ORDER after FANOUT2UP;
    FANOUT1LIST := NIL;
  end;
  Mark N;
  return;
end;
```

図4　変数順決定アルゴリズム

実行中のネット	FANOUT2UP	FANOUT1LIST	変数順
N 9	undef	NIL	NIL
N 7	undef	NIL	NIL
N 5	undef	NIL	NIL
N 1	undef	A	NIL
N 2	undef	A, B	C
N 3	C	NIL	C, A, B
N 8	C	NIL	C, A, B
N 6	C	NIL	C, A, B
N 4	C	D	C, A, B
終了時	C	NIL	C, A, B, D

図5　図1に対する図4のアルゴリズムの適用結果

時点での FANOUT2UPと FANOUT1LISTの値は図4の通りとなり、結果的に変数順 C, A, B, D を得る。

このアルゴリズムを ISCASのベンチマーク回路に適用した結果を表2④に示す。表から分かるように、C432を除いて全て図4のアルゴリズムによる結果が最もよい。実行時間については、文献［8、19］等に与えられている従来のものと比較すると、100倍かそれ以上の高速化が達成されている。表3の結果からみる限り、1000ゲート以上の回路の論理照合を実用時間で処理できる手法が開発できたと言える。また、実際の回路として、大型計算機の制御論理にも適用したが、ほぼ同様の結果が得られている。

回路名	①	②
SN181	3	7
C432	512	732
C499	312	—
C880	34	46
C1355	—	627
C1908	441	736
C2670	336	280
C3540	7613	×
C5315	253	460
C6288	×	×
C7522	×	365

表3　多出力処理①と中間変数導入②の効果
SUN3/260, 24MB
処理時間単位秒

さらに、5章で述べるように、論理の比較を高速に行えることから、論理照合以外にも、多段論理式簡単化（品質の高い自動回路合成）、スウィッチレベルの回路の論理検証、テスト生成、順序回路の自動検証等応用範囲は広く、本結果の重要性は極めて高い。

4．変数順以外の高速化手法とその実行結果

変数順以外の高速化手法として、ここでは、

① 多出力を同時に照合する

② グラフを入力から順に計算していく際、途中のネットでグラフのノード数が大きくなりすぎた場合には、そこで回路を切断して外部入出力とし、それらを用いて出力と切断して外部出力となったネットに対するグラフ（もとの出力に対するグラフには、この時点では切断して外部入力となった外部入力変数と含んでいる）を計算してから後、compose演算［2］によって 切断しなかった場合の出力にたいするグラフを求める

①については、以下のような理由で計算の重複を避けることができ処理時間が短縮できる可能性がある。まず、1出力ずつ処理すると、一般的には、図6のように異なる出力に対しては、回路のある部分を重複して切り出すことになり、従ってこの部分のグラフの計算は複数回同じことを計算してしまう。そこで、①では、各出力ごとにグラフを計算していくが、現在処理中の出力に関係する回路の内、

他の出力にも関係する部分のグラフの計算結果は残しておくようにする。このようにすると、1出力ごとに処理するのに比べ、必要なメモリ量が増加するが、同じ回路部分の計算は1回しか行わないため、処理時間の減少が期待できる。あるネットのグラフを残しておくべきか否かの判断は、そのネットが何回使われているか、つまりそのネットのファンアウト値を予め計算しておくことにより、そのファンアウト値の回数だけ使用するまで残しておくようにすればよい。

②については、以下のような理由で計算途中でのグラフのノード数の増加を押さえ計算時間が短縮できる可能性がある。回路の入力から順にグラフを計算していくと、途中例えば、図7(a)のネット tに対するグラフのノード数が大きくなってしまうと、その後に続くネットに対するグラフの計算にはそのノード数の大きいグラフから計算しなければならないことになる。ゲートの入力に対するグラフから出力に対するグラフの計算手間は入力に対するグラフの大きさに比例するので、一旦大きなグラフができてしまうとそれ以降の計算時間が大きくなってしまう。特に、回路の入力に近い（出力までまだ多くのゲートが存在する）ところでグラフが大きくなる場合に計算時間の増大は大きい。そこで、(b)のようにネット tで回路を切断し、外部入出力端子とすることで、入力端子 tに対するグラフは1変数のみのグラフとし（ノード数3）、 t以降の回路に対するグラフを小さく保つ。このようにして、出力 yに対するグラフが求まるが、それは入力として xx以外に tも含んでいる。また、切断したネット tから出た出力端子 tに対するグラフも求まるが、それは入力としては xのみ含んでいる。
従って、各々のグラフをＧｙ，Ｇｔとすると、以下が成立する。

$$y = Gy(x, t), \quad t = Gt(x)$$

一方、(a)に対しては、グラフをＧとすると、

$$y = G(x)$$

が成り立つ。以上から、ＧをＧｙ，Ｇｔで表現すると、

$$G(x) = Gy(x, Gt(x))$$

となり、ＧｙとＧｔの compose演算を行うことによりＧを求めることができることが分かる。グラフに関する compose演算も apply演算と同様に計算できる。

次に①、②の ISCASのベンチマーク回路への適用結果を表4に示す。表中、－になっているところは実験していないことを示す。表から以下のことが言える。ただし、①と②はそれぞれ別個に独立に試している。

まず、①については、実行できたものは、処理時間が2倍近く速くねっているものも多く、効果が大きいことが分かる。しかし、 c6288や c7552は実行できなかった。その理由は、①を適用するには全ての出力に対し、同じ変数順としなければならないが、回路が大きい場合には、全ての出力に対しうまくいく変数順を発見することは困難になる

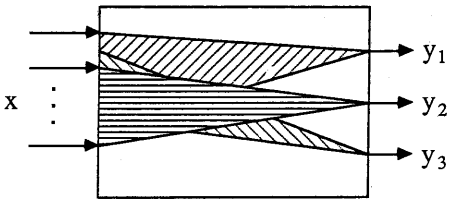


図6　多出力同時処理の効果

からである。つまり、回路が大きくなると、ある変数順はある出力に対してはうまくいき、グラフは小さくなるが、他の変数に対してはうまくいかず、グラフが大きくなりすぎて実行できなくなることが起こりやすい。実際、 c6288と c7552ではよい変数順をみつけることができなかった（他の回路に対する変数順の決定は、図4のアルゴリズムをある出力に対し適用し、残った入力があれば他の出力に対しても図4のアルゴリズムを適用することで順序を決めていった）。従って、多出力に対するよい変数順を決める問題は今後の課題となる。

次に、②の適用結果をみると、これも実行できた場合には、2倍程度処理速度が速くなっている。回路の切断は途中のネットに対するグラフのノード数がある値（実験的に決めている）を越えた場合に行うようにしている。通常、数箇所以下である。実行不可能となったのは、回路を切断したために、却って出力に対するグラフのノード数が取り扱えないくらい増大してしまうことが起こったためである。つまり、回路を切断して、グラフを作った後、 compose演算を実行中にノードが大きく成り過ぎてしまう場合があった。これは、切断点が悪いからかもしれないし、変数順がよくないからかもしれなし、他の理由からかもしれず、今後の検討課題となっている。

以上、うまくいかない例題もあるが、多くの場合には①や②を適用することで、数十%の処理速度向上が期待でき、その効果は大きいと言える。

5．おわりに

グラフ表現に基づく論理照合手法に関して、変数順の決定法とその評価結果について述べた。従来より 100倍以上の高速化が達成されており、論理照合できる回路規模が大幅に拡大したと言える。

ここで示した変数順決定アルゴリズムは単純なものであり、今後の改良により、さらに大きな性能改善が期待できる。また、論理照合以外の分野、例えば論理式簡単化等への応用も活発化すると考えられる。

参考文献

[1] F. Brglez and H. Fujiwara. A nertral netlist of 10 combinational benchmark circuits and a target translator in fortran. In *IEEE International Symposium on Circuits and Systems*, June 1985.

[2] R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computer*, C-35(8):667–691, August 1986.

[3] S. Devadas and A.R. Newton. *On the Verification of Sequential Machines at Differing Levels of Abstraction.* Technical Report UCB/ERL M86/93, Electronics Research Laboratory, University of California, Berkeley, December 1986.

[4] M. Fujita, S. Kono, H. Tanaka, and T. Moto-oka. Tokio: logic programming language based on temporal logic and its complication to prolog. In *3rd ICLP*, London, 1986.

[5] M. Fujita, H. Tanaka S. Kono, and T. Moto-oka. Assistance in hierarchical and structured logic design using temporal logic and prolog. In *Proceedings. Pt. E*, IEE, September 1986.

[6] M. Fujita, H. Tanaka, and T. Moto-oka. Logic design assistance with temporal logic. In *IFIP 7th Computer Hardware Description Languages and their Application*, IFIP, August 1985.

[7] M.J.C. Gordon and J. Herbert. Formal hardware verification methodology and its application to a network interface chip. In *Proceedings. Pt. E*, pages 255–270, IEE, September 1986.

[8] G.D. Hachtel and R.M. Jacoby. Algorithms for multilevel tautology and equivalence. In *IEEE International Symposium on Circuits and Systems*, June 1985.

[9] G.D. Hachtel and P.H. Moceyunas. Parallel algorithms for boolean tautology checking. In *International Conference on Computer Aided Design*, November 1987.

[10] F.K. Hanna and N. Daeche. Specification and verification of digital systems using higher-order predicate. In *Proceedings. Pt. E*, pages 242–254, IEE, September 1986.

[11] W.A. Hunt Jr. The mechanical verification of a microprocessor design. In *WG10.2 International Working conference on From HDL Descriptions to Guaranteed Correct Circuit Designs*, Septmeber 1986.

[12] H.T. Ma, S. Devadas, and A.L. Sangiovanni-Vincentelli. Logic verification algorithms and their parallel implementation. In *24th Design Automation Conference*, June 1987.

[13] Z. Manna and A. Pnueli. *Verification of Concurrent Programs, Part 1: The Temporal Frame work.* Technical Report STAN-CS-81-836, Dept. of Computer Science, Stanford University, June 1981.

[14] B.C. Moszkowski. *Reasoning about Digital Circuits.* Technical Report STAN-CS-83-970, Stanford University, June 1983.

[15] H. Nakamura, M. Fujita, S. Kono, and H. Tanaka. Temporal logic based fast verfication system using cover expressions. In *VLSI '87*, IFIP, August 1987.

[16] P. Roth. Hardware verification. *IEEE Trans. Computer*, C-26, 1977.

[17] K. Supowit and S.J. Friedman. A new method for verifying sequential circuits. In *23rd Desgin Automation Conference*, June 1986.

[18] T.J. Wagner. *Hardware Verification.* Technical Report STAN-CA-77-632, Dept. of Computer Science, Stanford University, September 1977.

[19] R. Wei and A.L. Sangiovanni-Vincentelli. Proteus: a logic verification system for combinational circuits. In *International Test Conference*, 1986.

[20] P. Wolper. Temporal logic can be more expressive. In *22nd Annual Symposium on Foundation of Computer Science*, October 1981.

[21] 藤沢，藤田，川戸：グラフに基づく論理照合アルゴリズムの評価と改良，情報処理学会第37回全国大会発表予定．

[22] 藤田，藤沢：各種論理照合アルゴリズムの比較と統合，第18ＦＴＣ研究会

[23] 石浦、安浦、矢島：High-speed logic simulation on vector processors, IEEE Trans. CAD, Vol. CAD-6, No. 3, pp. 305-321, May 1987.

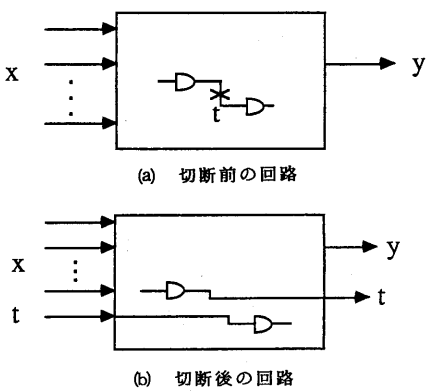


図７　回路切断による処理の高速化