テスト生成手法を用いた論理回路設計に関する研究

応用物理学専攻
市原 英行

1999 年 9 月

目 次

第 1 章　序論 **1**

第 2 章　論理回路のテストと回路簡単化 **5**

2.1　まえがき 5

2.2　論理回路のテスト 6

2.2.1　論理テストと IDDQ テスト 6

2.2.2　故障モデル 6

2.2.3　テスト生成の流れ 9

2.2.4　経路活性化法 10

2.3　テスト生成手法を用いた回路簡単化 13

2.3.1　検出不能な縮退故障と冗長部分回路除去 13

2.3.2　冗長部分回路の付加と除去による回路簡単化 16

2.4　あとがき 18

第 3 章　含意操作の拡張 **19**

3.1　まえがき 19

3.2　含意操作の拡張 20

3.2.1　静的学習 20

3.2.2　再帰学習 22

3.3　学習順序と獲得含意数 25

3.4　提案学習順序 27

3.5　実験結果 28

3.6　あとがき 32

第 4 章　含意関係の不変性を考慮した回路簡単化 **33**

4.1　まえがき 33

4.2　部分回路変換と含意関係 34

4.3　部分回路変換における含意関係の不変性 35

4.3.1 学習関与直接含意 35
4.3.2 論理ゲートの入力の除去と含意関係 37
4.3.3 分岐の枝の除去と含意関係 38
4.3.4 無効となる可能性をもつ間接含意 39
4.4 アルゴリズム 40
4.5 実験結果 41
4.6 あとがき 43

第 5 章 含意操作による冗長指摘を用いた冗長付加と除去による回路簡単化 45

5.1 まえがき 45
5.2 含意関係に基づく冗長付加 46
5.3 含意操作を用いた冗長指摘 47
5.3.1 基本的なアイデア 47
5.3.2 不当割当対 48
5.3.3 含意操作を用いた冗長指摘 54
5.4 冗長部分回路の同時除去性 55
5.5 実験結果 56
5.6 あとがき 58

第 6 章 テストベクトル数が制限された条件下でのテスト生成 61

6.1 まえがき 61
6.2 測定ベクトル数の制限下でのテスト生成問題 62
6.3 故障検出率の計算方法 63
6.3.1 全故障に対する故障検出率の計算の重要性 63
6.3.2 重み付き故障リスト 65
6.4 測定ベクトル選択手法 65
6.4.1 遷移ベクトルの評価値 65
6.4.2 選択アルゴリズム 67
6.5 実験結果 68
6.6 あとがき 71

第 7 章　結論　73

第1章　序論

近年の半導体技術の発展は，大規模で複雑な集積回路の製造を可能にしたため，論理回路の設計は非常に複雑で困難なものとなった．論理回路設計の複雑さを緩和するためには，コンピューターによる設計支援 (CAD: Computer Aided Design) が必要不可欠であり，現在も半導体技術の開発と共に盛んに研究が行われている．

一方，論理回路の信頼性を向上するために，製造された集積回路が正しく動作するかどうかを調べるためのテスト技術も年々その重要性を増している．論理回路のテストは，テストベクトルと呼ばれる検査入力を回路に印加し，回路の応答が正常な回路と同じかどうかを調べることで行われる．このとき用いるテストベクトルの優劣で，テストの信頼性とコストが決まるため，効果的なテストベクトルを作ることが望まれる．しかし，論理回路の大規模かつ複雑化は，テストベクトルの生成をも困難にしており，テスト生成問題もコンピューターの支援が必要不可欠となっている．

テスト生成では, (1) テスト生成時間を最小にする，(2) 故障検出率が最大であるテストベクトルを生成する， (3) 生成されるテストベクトル数を最小にする，という 3 つの目標がある [1, 2].

(1) のテスト生成時間を最小にする目標に対して，テスト生成をコンピュータ上で行うためのテスト生成アルゴリズムに関する研究は古くから行われており，いくつかの高速なアルゴリズムが知られている [1]-[7]．しかしながら現在では回路の大規模化により，より高速なテスト生成アルゴリズムが必要とされている，テスト生成アルゴリズムは回路の内部に故障を仮定し，その故障が検出されるように内部の信号値を割り当てていくことでテストベクトルを生成する．その信号値割当の操作の 1 つである含意操作は，与えられた信号値割当から一意に決まる信号値を決めるための操作であるが，より多くの信号値を決めることができるように含意操作を拡張することで，テスト生成アルゴリズムを高速化することができる [5, 6, 7]．本論文では，含意操作の拡張するために提案された静的学習 [5] を扱う. 静的学習で行われる処理の順序を考察することで，静的学習の効率を向上させる手法を提案する．

(2) 故障検出率が最大であるテストベクトルを生成すること，および，(3) 生成されるテストベクトル数を最小にすることは，どちらも生成されるテストベクトルに対する要求である．故障検出率は，回路内に定義したすべての故障の中で検出できる故障の割合であり，高いことが望ましい．また，テストベクトル数は論理回路のテスト時間に影響を与えるため，少ないことが望ましい．この 2 つの目的は一般的にトレードオフの関係にあるため，できるだけ多くの故障を検出できるテストベ

クトル集合を得た後に，テストベクトル数を減らすことが行なわれている [26]-[35]．特に IDDQ テストと呼ばれる静的電源電流 I_{ddq}を測定するテスト方法では，テスト時間が長くなる傾向があるため，テストベクトル数を減らすことがより重要となっている．このため，IDDQ テスト用のテストベクトルを減らすための手法がいくつか提案されている [30]-[35]．これらの手法は，故障検出率を最大にするという制限下でテストベクトル数を最小としていたが，この制限下では十分に小さいテストベクトル集合が得られないの可能性がある．そこで，あらかじめテストベクトル数に制限を加え，テスト時間を許容できる範囲内に抑える必要がある．本論文は，テストベクトル数に上限がある場合のテスト生成問題を定義し，この問題に対して与えられたテストベクトルの系列から，与えた上限数のテストベクトルを選択する方法を提案する．ベクトルの選択では，故障の検出困難度から求まるベクトルの評価値を用いるが，これにより短時間で効率的なテスト選択ができることを示す．

テスト生成で用いられる様々な手法は，テスト生成問題のためだけではなく，回路簡単化 [8], [14]-[22]，論理検証 [23]，フォールスパス解析 [24] などの問題にも適用できる．本論文ではテスト生成手法を応用した回路簡単化手法について述べる．

回路簡単化 [8]-[22] は，与えられた仕様または機能から論理回路を作成する論理合成時に行なわれる処理である．自動合成ツールなどで合成された論理回路はいくつかの回路の論理に関係しない冗長な部分回路を含んでいる場合があるが，これらの冗長な部分回路は設計者が意図していない不要な部分であるため，除去することが望まれる．さらに，冗長な部分回路を持たない論理回路でも，回路があらわす論理関数が等価であるように回路を変形し，それまで非冗長であった部分回路を冗長に変えて除去することで回路を簡単化できる．

組合せ回路の簡単化はよく研究されていて，二段回路簡単化手法としては ESPRESSO[9]，また多段回路簡単化手法としては，論理関数を代数式と見て割算を行なう MIS[10] や許容関数を用いたトランスダクション法 [11] などが提案されている．

テスト生成手法を用いて，回路簡単化における冗長部分回路の除去 [14]-[19]，そして回路が表す論理関数が等価なままの回路変換である冗長部分回路の付加 [20]-[22] を行なうことができる．冗長部分回路の除去を行なう場合，縮退故障の検出可能性に基づく冗長信号線の判定が行なわれる．縮退故障とは，信号値が 0 または 1 に固定してしまう故障であり，縮退故障が検出不能である場合はその信号線は冗長信号線であることがわかる．一方，冗長部分回路を付加は，含意操作により得られる信号値間の含意関係を利用することで行なうことが可能である．

テスト生成手法を回路簡単化に用いる利点は，テストベクトルが生成可能な回路であれば，大規

模な回路であっても回路簡単化が行なえる点である．しかしながら，テスト生成自体は計算時間がかかる手法であるため，テスト生成を行なうことで冗長な部分回路を見つけて除去する手法は処理時間の点で問題となる．

本論文ではテスト生成手法を利用した 2 つの高速な回路簡単化手法を提案する．1 つは，回路変換に対して不変な含意関係を利用することで回路簡単化に必要な処理時間を短縮する手法である．冗長部分回路を除去すると回路内の信号値間の含意関係の一部が無効となるために，含意関係を調べ直す必要がある．回路が大規模になると含意関係を調べ直す時間も膨大になるため，この再計算時間を減らすことが望まれる．そのため，回路の部分変換に対する含意関係の不変性について考察し，不変な含意関係を繰り返して用いることで，含意関係を抽出する操作の繰り返しを省く手法を提案する．もう 1 つは，冗長部分回路の発見をテスト生成手法の含意操作を利用して行う高速な回路簡単化手法の提案である．これまでの論理簡単化では冗長部分回路を付加したことによって生じる他の冗長部分回路を発見する操作に時間がかかっていたため，この操作を高速化すれば計算時間の大幅な削減が期待できる．

本論文は以下のように構成される．第２章では，論理テストと IDDQ テストについて述べ，論理テストに対するテスト生成手法について簡単に説明する．また，テスト生成手法を用いた回路簡単化について概説する．

第３章では，テスト生成手法における含意操作を拡張するための２つの手法，静的学習と再帰学習について述べる．さらに，静的学習における内部処理の順番が静的学習の能力に影響を与える点を指摘し，計算機実験を用いて有効な静的学習の処理順序を考察する [47].

第４章と第５章では，テスト生成手法を用いた回路簡単化について述べる．第４章では，含意関係の不変性を考慮した高速な回路簡単化手法を提案する [43]．初めに，部分回路変換と信号値間の含意関係の有効性の関連について述べ，回路の部分変換に対して不変な含意関係だけを見つける方法を提案する．それを，冗長部分回路除去手法に利用することで本手法の有効性を確かめる．

第５章では，テスト生成手法の含意操作を用いた冗長部分回路の付加と除去による回路簡単化手法を提案する [44, 47, 49]．最初に，冗長部分回路を付加する前は有効であったが，付加後は不当になる信号値割当対を考える．そして，不当割当対から含意操作を行うことで，不当割当対を検出のために必要とする縮退故障を特定し，そこから冗長部分回路付加によって新たに生じた冗長部分回路を発見する手法を提案する．

第６章では，テスト生成手法における，テストベクトル数が制限された条件下でのテスト生成問

題に対して，この問題におけるテスト選択手法を提案する [45]．提案手法では，故障の検出困難度から求まるベクトルの評価値を用いて，与えられたテストベクトル集合からよりよいテストベクトルを選ぶ．

第 7 章では，本論文のまとめと今後の課題について述べる．

第2章　論理回路のテストと回路簡単化

2.1　まえがき

集積回路設計の流れを図 2.1 に示す．システム設計は，論理回路の仕様から機能ブロックを構成し，その動作を決める．機能設計では，機能ブロック内の動作を，レジスタ間のデータや制御値の流れを記述したレジスタトランスファレベル (RTL) 回路を用いて記述する．論理設計では，RTL 回路からゲートレベル回路を合成する論理合成を行う．回路設計では，ゲートレベル回路をセルライブラリを用いてトランジスタレベル回路に変換する．そして，レイアウト設計では, マクロセルの配置・配線を行う．また, テスト生成では，製造された論理回路をテストするために必要な入力信号値（テストベクトル）を主にゲートレベル回路に対して生成する．本論文で対象としているのは，テスト生成と論理設計の論理合成中に行われる回路簡単化であり，対象回路は主にゲートレベル回路である．

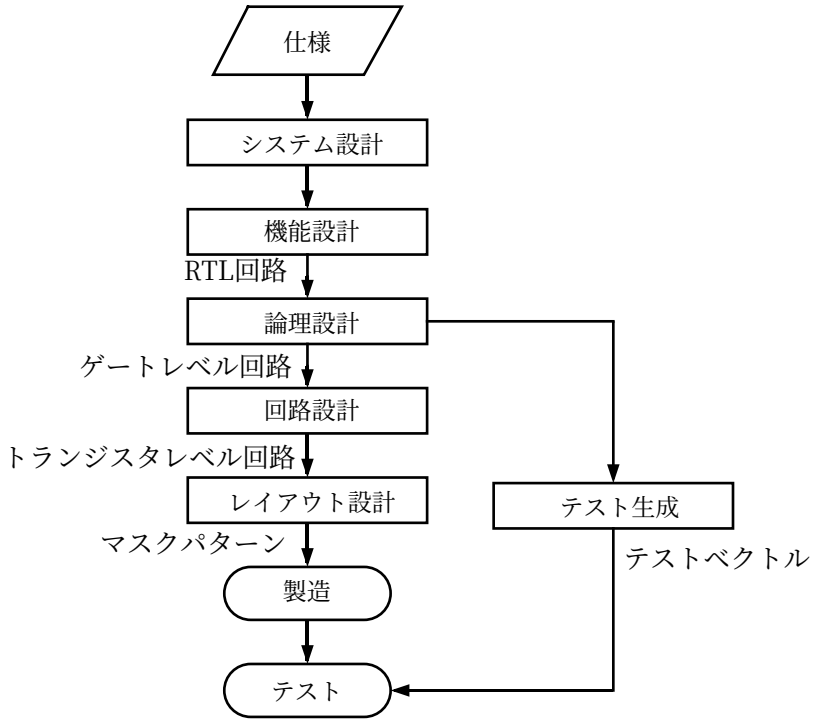


図 2.1: 集積回路設計の流れ

ゲートレベル回路は論理素子とそれらを結ぶ信号線からなり，論理素子に記憶素子（フリップフロップ）を含まないものを組み合せ回路，含むものを順序回路と呼ぶ．本論文は，第 7 章のテストベ

クトル数が制限された条件下でのテスト生成手法で順序回路を対象とする他は，すべて組合せ回路を対象とする．本論文で扱う論理ゲートは NOT, AND, NAND, OR, NOR, EXOR ゲートである．1 つのゲートの出力信号線が 2 つ以上のゲートにつながっている場合，出力信号線は分岐することになる．分岐前の信号線を分岐の幹，分岐後のそれぞれの信号線を分岐の枝と呼ぶことにする．

本章では，2.2 節では，テスト方法とテスト生成手法について述べる．2.3 節では回路簡単化手法について述べる

2.2　論理回路のテスト

2.2.1　論理テストと IDDQ テスト

本論文では，論理回路のテストとして，論理テストと IDDQ テストという 2 種類のテスト方法を対象とする．論理テストは，論理回路のテストにおいてもっとも広く利用されているテスト方法である．論理テストでは，テストベクトルに対応する信号値 (電圧) を入力信号線に印加し，出力信号線の信号値（電圧）をテスターで調べ，期待されている信号値と等しいかどうかで故障回路を見分ける．

一方，IDDQ テストは，CMOS 回路に対するテスト技術であり，CMOS 回路の静的電源電流 (I_{DDQ}) を測定することで回路のテストを行う．正常な CMOS 回路では，スイッチング時以外の静的な状態での I_{DDQ}は極めて小さいが，トランジスタの欠陥があったり，信号線間が短絡しているときには，I_{DDQ}は正常な回路と比べて非常に大きな値となる．IDDQ テストは，論理テストでは検出できない回路の欠陥を検出できる利点があるが，I_{DDQ}の測定に時間がかかるため，テスト時間が長くなる欠点がある．

2.2.2　故障モデル

テスト生成では，回路の物理的欠陥を模式化した故障モデルを用いてテストベクトルを生成する．本論文では，論理テストに対しては縮退故障モデルを，IDDQ テストに対してはゲート内短絡故障モデルを使用する．

縮退故障モデルは, 各信号線の信号値が論理値 0 または 1 に固定してしまう故障である．信号値が 0(1) に縮退する故障を 0(1) 縮退故障と呼ぶ．現実の物理的欠陥は，必ずしも信号線の論理値が固定される故障ではないものの縮退故障とみなして扱える場合も多く，論理テストのテスト生成の研究で

は有効な故障モデルとして長く利用されている．本論文では，信号線 a 上の $v(\in \{0,1\})$ 縮退故障を a^v と表すことにする．

一方，ゲート内短絡故障モデルは，トランジスタレベルで表現された論理回路上で定義される故障である．トランジスタを結ぶ信号線と論理ゲートの入力線と出力線，そして VDD と GND をこのゲートのノードと呼ぶ．図 2.2 の典型的なトランジスタレベルの NAND ゲートでは，6 つのノード a, b, c, d, VDD, GND がある．ゲート内短絡故障モデルは，ゲート内の 2 つのノードが短絡（ショート）する故障モデルであり，ノード p と q の短絡故障は，$\langle pq \rangle$ と表記する．図 2.2 には，ゲート内短絡故障 $\langle cd \rangle$ を示している．標準的な n 入力の CMOS ゲートでは，$2n+2$ のノードが存在するため，${}_{(2n+2)}C_2 = (n+1)(2n+1)$ 個のゲート内短絡故障が存在することになる．IDDQ テストによってゲート内短絡故障 $\langle pq \rangle$ が検出される条件は，この故障の影響で VDD から GND までの電流パスが生じることであり，論理値で考えると $(p,q) = (0,1)$ または $(1,0)$ となることを意味する．図 2.2 の NAND ゲートの入力 $(a,b) = (1,0)$ は，ノード c,d の論理値を $(c,d) = (0,1)$ とするため，ゲート内短絡故障 $\langle bc \rangle$ のテストベクトルであり，VDD-p2-d-c-n2-GND という電流パスが生じることになる．

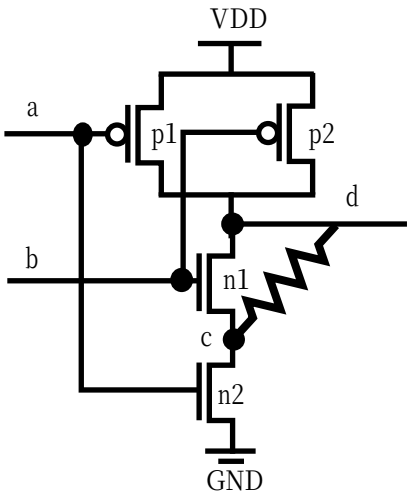


図 2.2: トランジスタレベルの 2 入力 NAND

複数の故障が常に同じテストベクトルで検出できる場合，それらの故障は等価故障と呼ばれる．故障の等価性は次のよう定義できる．

定義 2.1（故障の等価性） 2 つの故障 f_1, f_2 を考える．もし, f_1 を検出することができるすべてのテストベクトルの集合 T_1 が, f_2 を検出することができるすべてのテストベクトルの集合 T_2 と等しければ

$(T_1 = T_2)$, f_1 と f_2 は等価であるという.

等価故障がある場合，その中の１つの故障だけにテスト生成を行えば，他の等価故障はそのテストで検出できるため，テスト生成の対象となる故障の数を削減することができる．このとき等価故障の中でテストの対象として選ぶ１つの故障を**代表故障**と呼ぶ．なお，等価故障がない故障は，その故障自身が代表故障となる．

縮退故障の代表故障は，1 つのゲートの入出力信号線上の縮退故障の等価性を調べることで選ぶことができる．それぞれのゲートの等価故障を示す.

定義 2.2（縮退故障の等価性） 次に示す縮退故障は等価故障である.

(1) AND ゲートの各入力信号線の 0 縮退故障と出力信号線 0 縮退故障.

(2) OR ゲートの各入力信号線の 1 縮退故障と出力信号線の１縮退故障.

(3) NAND ゲートの各入力信号線の 0 縮退故障と出力信号線の１縮退故障.

(4) NOR ゲートの各入力信号線の 1 縮退故障と出力信号線の 0 縮退故障.

(5) NOT ゲートの入力信号線の 1 縮退故障と出力信号線の 0 縮退故障.

(6) NOT ゲートの入力信号線の 0 縮退故障と出力信号線の 1 縮退故障.

それぞれのゲートの等価故障から代表故障が決まる．例えば，２入力 AND ゲートを考えると，定義 2.2 の (1) から出力信号線の 0 縮退故障を代表故障として選び，２本の入力信号線上の 0 縮退故障はテスト生成の対象から外す.

表 2.1: 2 入力 NAND ゲートのゲート内短絡故障の故障表

a	b	c	d	ab	ac	ad	av	ag	bc	bd	bv	bg	cd	cv	cg	dv	dg	vg
0	0	X	1			1	1			1	1						1	1
0	1	0	1	1		1	1		1			1	1	1			1	1
1	0	1	1	1				1	1	1	1				1		1	1
1	1	0	0		1	1		1	1	1		1		1		1		1
0	X	X	1			1	1										1	1
X	0	X	1							1	1						1	1
1	X	X	X					1										1
X	1	0	X						1			1		1				1
代表故障				r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	–	r11	–	r12	r13

ゲート内短絡故障の代表故障は，トランジスタレベル回路における論理シュミレーションを行ない，検出できる故障を調べることで選ぶことができる．表 2.1 は図 2.2 の 2 入力 NAND ゲートの故障表である．最初の 4 列は，ゲートの入力 a,b の値とそれから決まる内部ノード c およびゲートの出力 d の値を示してある．残りの列はそれぞれのゲート内短絡故障がどの入力で検出できるかを示している．例えば故障 $\langle ab \rangle$ は入力 $(a,b) = (0,1),(1,0)$ で検出できる．故障 $\langle bg \rangle$ と $\langle cv \rangle$ は常に同じ入力で検出できるため等価故障であるため，どちらか 1 つを代表故障とすれば良い．最後の行には 13 個の代表故障を示した．

2.2.3 テスト生成の流れ

図 2.3 に一般的なテスト生成の流れを示す．最初，テスト生成の対象回路に対して故障モデルを決め，代表故障からなる故障リストを作成する．故障リストから１つの故障を取り出し，この故障を検出するためのテストベクトルをアルゴリズムによって求める．アルゴリズムによるテスト生成が成功した場合，得られたテストベクトルに対して他に検出できる故障があるかどうかを調べるために，故障シミュレーションを行い，同じテストベクトルで検出できる故障を故障リストから除く．故障リスト内に，まだ検出できるかどうか分からない未検出故障が残っているならば，再び１つの故障を故障リストから取り出してアルゴリズムによるテスト生成を行う．

次節では，縮退故障を対象とした論理テストに対するアルゴリズムによるテスト生成手法について述べる．このテスト生成手法は経路活性化法と呼ばれる．なお，ゲート内短絡故障を対象とした IDDQ テストのアルゴリズムによるテスト生成手法も，経路活性化法の一部

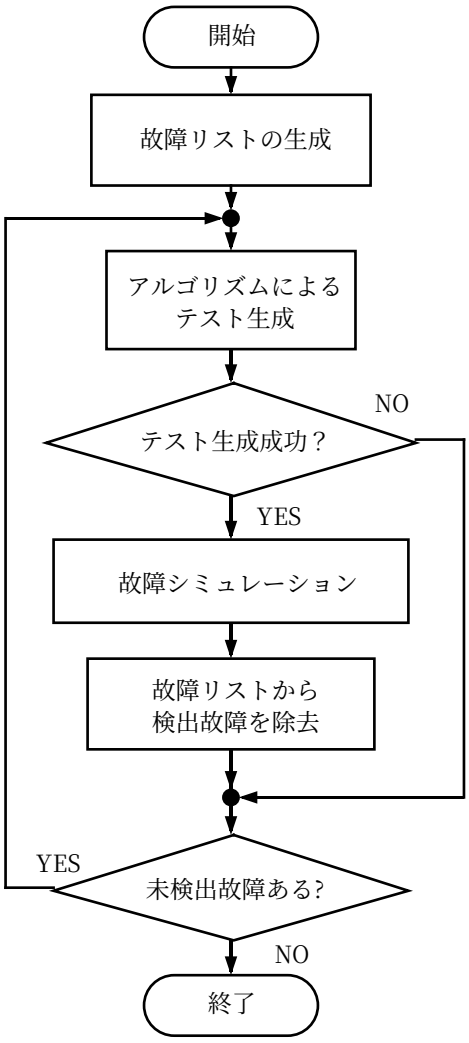


図 2.3: テスト生成の流れ

の操作を用いて行なえる．

2.2.4　経路活性化法

経路活性化法では，正常回路と故障回路の信号値を同時に扱う信号値を用いる．本論文では５種類の信号値 0, 1, D, $\overline{\mathrm{D}}$, X からなる５値モデルをもちいる．これらの信号値は次のように定義される．

- 信号値 0：正常時も故障時も論理値 0 を表す信号値.
- 信号値 1：正常時も故障時も論理値 1 を表す信号値.
- 信号値 D ： 正常時は 1，故障時は 0 となる信号値.
- 信号値 $\overline{\mathrm{D}}$： 正常時は 0， 故障時は１となる信号値.
- 信号値 X：信号値 0 と信号値 1 のどちらでも良い状態であるドントケアを表す信号値

５値モデルでの AND 演算と NOT 演算の結果を表 2.2 に示す．AND 演算については，表の１列目と１行目が入力信号値を示し，NOT 演算については表の１行目が入力信号値を表している．

表 2.2: ５値モデルでの AND と NOT の演算表

AND	0	1	D	$\overline{\mathrm{D}}$	X
0	0	0	0	0	0
1	0	1	D	$\overline{\mathrm{D}}$	X
D	0	D	D	0	X
$\overline{\mathrm{D}}$	0	$\overline{\mathrm{D}}$	0	$\overline{\mathrm{D}}$	X
X	0	X	X	X	X
NOT	1	0	$\overline{\mathrm{D}}$	D	X

経路活性化法では，**故障の顕在化操作**，**後方操作**，**故障の伝播操作**，**含意操作**という 4 つの基本操作を用いて，縮退故障を仮定した信号線の信号値を D または$\overline{\mathrm{D}}$にすることにより，少なくとも１つの外部出力に D または$\overline{\mathrm{D}}$を伝搬させる入力ベクトルを求める．

それぞれの基本操作の内容を説明する．

故障の顕在化操作は，経路活性化法の最初に行われる．これは，0(1) 縮退故障を仮定する信号線に信号値 D($\overline{\mathrm{D}}$) を設定する操作である．例えば図 2.4 の回路においてゲート A の出力信号線に 0 縮退故障を仮定する．故障の顕在化操作はこの信号線に信号値 D を与える．

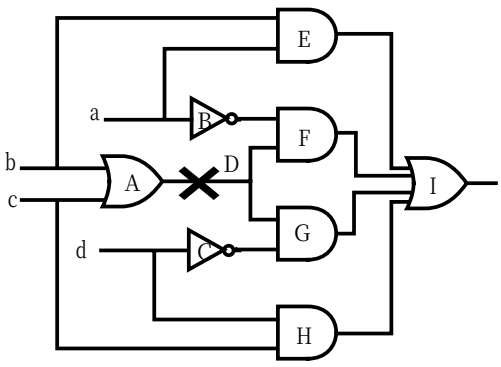


図 2.4: 故障の顕在化操作

後方操作は，外部入力の値を決定するために既に値の決定している信号線の論理値と矛盾しないように，内部の信号線に論理値を割り当てる操作である．図 2.5 に示したように，OR ゲート A の出力信号線に信号値 D がある場合，ゲート A の入力信号線 b,c のいづれかに信号値 1 を割り当てる必要がある．図 2.5 の例では後方操作により信号線 b に信号値 1 が割り当てることを考える．もし, このあとの操作において信号線 b の値を 1 にしたことが原因で，信号値の矛盾が起こったり故障の影響を外部出力に伝播できないことが分かると，バックトラックによってこの後方操作を行なう前の信号値割当の状態に戻り，信号線 c に信号値 1 を与えることになる．

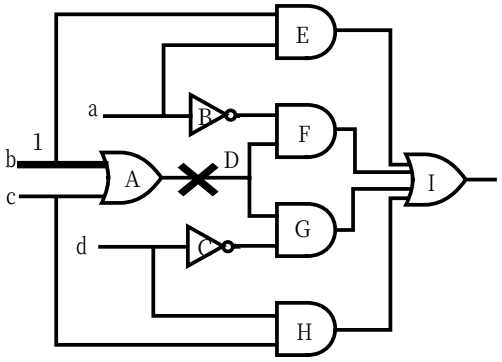


図 2.5: 後方操作

故障の伝搬操作は，D または$\overline{\text{D}}$で表される故障の影響を外部出力に伝搬するようにゲートの入力値を割り当てる操作である．図 2.6 は，故障の伝搬操作の例を示す．この例では，故障の顕在化操作で割り当てた信号値 D をゲート G の出力に伝搬するためにゲート C の出力であるゲート G の入力に信号値 1 を割り当てる．故障の伝搬操作で割り当てる信号値は一意に決まる信号値割当ではない．例えば，ゲート F への故障の伝搬も考えられるため，ゲート B の出力に 1 を割り当てる場合も考えられる．このため，後方操作と同様にバックトラックを起こす可能性がある．

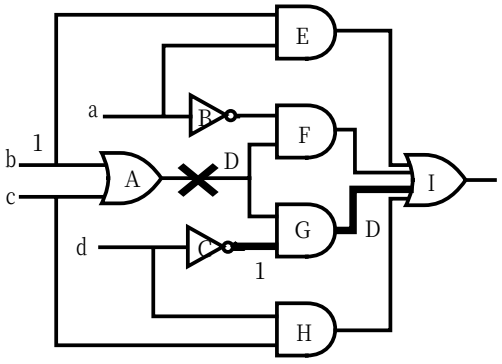


図 2.6: 故障の伝搬操作

含意操作は，回路内にある信号線の値が決定したことにより，一意的に決まる他の信号線の信号値を決定する操作である．例えば図 2.6 の回路に対して含意操作を行えば，図 2.7 に示した信号値割当が得られる．この例では，信号線 b の値が 1 に決定したことにより，ゲート E の１つの入力値は 1 となる．また，ゲート C の出力が 1 に決まったことにより，信号線 d の信号値が 0，そしてゲート H の出力値も 0 に決まる．

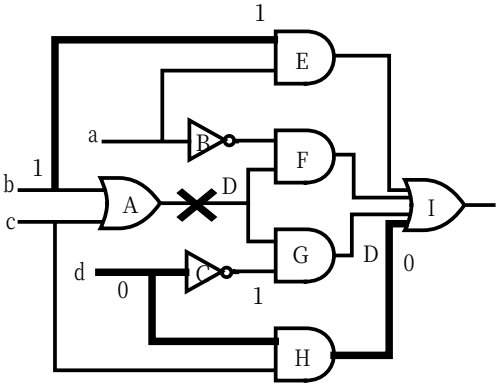


図 2.7: 含意操作

図 2.7 の信号値割当からさらに，故障の伝搬操作によりゲート E の出力に 0 を割り当て，含意操作を行えば，図 2.8 のようにテストベクトル $(a,b,c,d) = (0,1,X,0)$ が得られる．

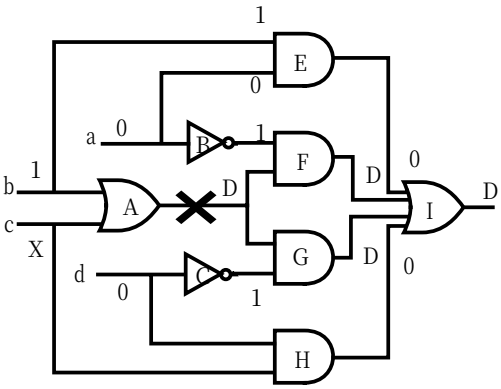


図 2.8: テストベクトル生成の成功例

後方操作や故障の伝搬操作で割り当てる信号値は，矛盾を起こすことがあるため，経路活性化法ではバックトラックを起こす可能性がある．例えば，図 2.7 の状態から故障の伝搬操作によって，ゲート F の出力に 0 を割り当てた場合，含意操作により図 2.9 のように外部出力の値が１になるため，故障の影響が外部出力に伝搬できなくなる．このような状態になった場合は，バックトラックにより，最後に値を割り当てる前の信号値割当の状態に戻り，他の信号値割当を行う．

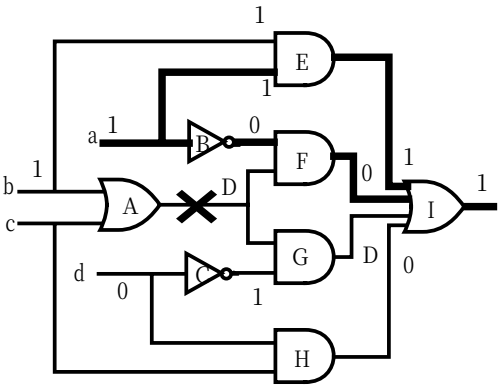


図 2.9: バックトラックを起こす場合

信号線 s 上の $a \in \{0,1\}$ 縮退故障を検出するテストベクトルを生成するための，経路活性化法の１つのアルゴリズムを示す．

操作 2.1 (経路活性化法を用いたテスト生成アルゴリズム)

step(1) 信号線 s の a 縮退故障に対する故障の顕在化操作を行う.

step(2) 信号線 a から外部出力にまでの経路を１つ選び，その経路に沿って故障の伝播操作を行う.

step(3) 回路内の信号値割当から，後方操作および含意操作を行う.

step(4) 矛盾なく外部入力値が決定すれば終了（テスト生成成功）. それ以外は step(5) へ.

step(5) step(3) の後方操作において，バックトラックの対象となる信号値割当があれば，バックトラックを行い step(3) へ. バックトラックの候補がなければ step(6) へ.

step(6) step(2) で選ばれていない信号線 a から外部出力までの経路が存在すれば，新たにその経路に沿った故障の伝播操作を行い (バックトラック)，step(3) へ. なければ，終了（対象とした故障に対するテストは存在しない）.

アルゴリズムによるテスト生成では，実用的な時間内でのテスト生成の停止を保証するために，バックトラックの回数によりアルゴリズムを打ち切る. 打ち切りの回数は生成されたテストベクトルの検出可能な故障数に影響する. バックトラックを減らすためには，経路活性化法の 4 つの基本操作を行なう順番を考慮したり，それぞれの操作を拡張することが行なわれている [3]-[7].

2.3 テスト生成手法を用いた回路簡単化

2.3.1 検出不能な縮退故障と冗長部分回路除去

回路簡単化は，図 2.1 の論理設計の論理合成のなかで行われる処理であり，一度合成された回路をより規模の小さい等価な回路に変換することを意味している. 組合せ回路の等価性は次のように定義する.

定義 2.3 (組合せ回路の等価性) n 入力 m 出力の組み合せ回路 C と C' を考える. 入力ベクトル $X = (x_1, x_2, \ldots, x_n)$ に対する，C, C' の出力関数をそれぞれ $f_i(X)$，$f'_i(X)$ とする $(1 \geq i \geq m)$. このとき，

$$\forall X \; \forall i : f_i(X) = f'_i(X) \tag{2.1}$$

ならば，組合せ回路 C と C' は等価である.

本論文では，回路の規模を論理回路を忠実に論理関数で表したときの論理式に現れる変数の数 (リテラル数) で表すことにする. 例えば，図 2.10(a) 回路の論理関数は $f = \overline{(a \vee b)} \vee a$ となるが，このときのリテラル数は 3 となる.

自動化設計ツールで合成した論理回路には，設計者が意図しない冗長部分回路が存在する場合がある．冗長部分回路は次のように定義できる．

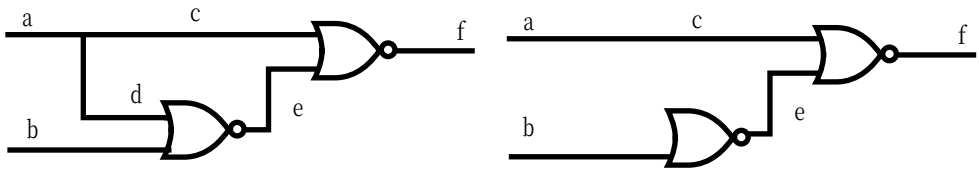
定義 2.4 (冗長部分回路) 回路 Cから部分回路 Pを除去した回路を C'とする．もし，回路 Cと C'が等価ならば，部分回路 Pは冗長部分回路である．また，冗長部分回路に含まれる信号線とゲートをそれぞれ，冗長信号線および冗長ゲートと呼ぶ．

冗長部分回路を発見して除去することで回路を簡単化することができる．例えば，図 2.10(a) の回路が論理合成により得られたとする．この回路において冗長部分回路を探すと信号線 d が冗長信号線であることが分かる．よって，信号線 d を除去し図 2.10(b) の回路に変換できる．この回路変換を論理関数の変形で見ると，

$$f = \overline{\overline{(a \vee b)} \vee a} \tag{2.2}$$

$$= \overline{a} \wedge b \tag{2.3}$$

のように，リテラル数 3 の論理式 (2.2) がリテラル数が 2 の論理式 (2.3) に変換できたことになる．



(a) 冗長信号線を含む回路　　(b) 冗長信号線を除去した回路

図 2.10: 論理回路の簡単化

冗長部分は，以下で定義する検出不能故障を発見することで見つけることができる．検出不能故障を定義するために故障差関数を定義する．n 入力 m 出力の組み合せ回路の入力ベクトル $X = (x_1, x_2, \ldots, x_n)$ に対する，外部出力 $j \in \{y_1, y_2, \ldots, y_m\}$ の出力関数を $f_j(X)$ とする．また，縮退故障 a^v が回路内に存在する故障回路における，外部出力 j の出力関数を $f_j^{a^v}(X)$ と表すとき，次のように故障差関数を定義する．

定義 2.5 (故障差関数) 入力ベクトル $X = (x_1, x_2, \ldots, x_n)$ に対して，

$$F_j^{a^v}(X) = f_j(X) \oplus f_j^{a^v}(X) \tag{2.4}$$

なる関数を外部出力 j の故障差関数という．ある外部出力 j において $F_j^{a^v}(X) = 1$ を満たす入力ベクトル X は縮退故障 a^v を検出するためのテストベクトルである．

故障差関数を用いて，検出不能故障を定義する．

定義 2.6 (検出不能故障) 縮退故障 a^v に関する故障差関数 $F_j^{a^v}(X)$ において，

$$\forall X \; \forall j : F_j^{a^v}(X) = 0 \tag{2.5}$$

である場合，縮退故障 a^v を検出不能故障と呼ぶ．

冗長信号線上の縮退故障は検出不能であるため，回路内の縮退故障の検出可能性を調べることで冗長部分回路を発見することができる [14]-[19]．例えば, 図 2.10(a) の回路において信号線 d 上の 0 縮退故障 d^0 に関する故障差関数を考えると，故障のない正常時の論理関数は $f = \overline{\overline{(a \vee b)} \vee a} = \overline{a} \wedge b$ であり，故障回路の論理関数は，信号線 d の信号値が 0 に固定するため $f = \overline{\overline{(0 \vee b)} \vee a} = \overline{a} \wedge b$ となる．よって，故障差関数は常に 0 となり，故障 d^0 は検出不能故障であることが分かる．

この例で示したように，縮退故障 a^v が回路内に存在する故障回路の，外部出力 j の出力関数を $f_j^{a^v}(X)$ は，信号線 a の値を v に固定することで得られる．これは，縮退故障 a^v が検出不能であれば，信号線 a の値を v に固定した影響は外部出力に伝わらない，すなわち信号線 a が冗長であることを意味する．

前節で示した経路活性化法を用いたテスト生成アルゴリズムでは，バックトラックの回数に制限を加えなければ，対象としている縮退故障が検出可能か不可能であるかを必ず判断することができる．よって，テスト生成を行うことで検出不能故障を見つけることが可能となる．

1 つの冗長信号線が分かると，いくつかの信号線やゲートも冗長となり除去できる．冗長部分回路を判定する方法を説明するために，論理ゲートの制御値と非制御値を定義する．

定義 2.7 (制御値, 非制御値) *AND, OR, NOR, NAND* ゲートにおいて，ゲートの 1 つの入力値だけで出力値が一意に決まる時, その入力値をゲートの制御値と呼ぶ．また制御値の反転値をゲートの非制御値と呼ぶ．

AND ゲートと NAND ゲートの制御値は 0 であり非制御値は 1 である， OR ゲートと NOR ゲートの制御値は 1 であり非制御値は 0 である．なお，EXOR ゲートには制御値も非制御値も存在しない．

検出不能故障 a^v から判定される冗長部分回路の判定方法は次のようになる．

操作 2.2 (検出不能故障 a^v からの冗長部分回路判定)

step(1) 信号線 a に信号値 $v(\in \{0,1\})$ を設定する．また，a を対象信号線，v を対象信号値とする．

step(2) 対象信号線を入力に持つゲートの制御値であるならば，制御値により一意に決まる出力値を対象信号値にし，出力信号線を対象信号線として，step(2) を繰り返す．そうでなければ，step(3) に進む．

step(3) 対象信号線から入力方向に向かって，分岐の枝または，外部入力までにある信号線とゲートが冗長部分回路である．

2.3.2　冗長部分回路の付加と除去による回路簡単化

論理回路が冗長部分回路を含まない場合でも，回路の一部に信号線やゲートを加えることで，冗長でない部分回路を冗長な部分回路に変換することができる．この変換と先程の冗長信号線を発見する操作と併せれば，非冗長な回路でもより簡単な回路に変形することができる [20]-[22].

回路の一部に信号線やゲートを加える操作では，変形後の回路が表す論理関数が変形前の論理関数と等価でなければならないため，付加する部分回路が常に冗長な部分回路でなければならない．図 2.11 に冗長信号線の付加と除去による回路簡単化の例を示す．この例では, 信号線 $b3$ を付加することによって，信号線 $b1, b2$ は非冗長な信号線から冗長な信号線に変わり，除去することができる．この回路簡単化を論理関数で表すと,

$$f = ((a \wedge b) \vee (b \wedge c)) \wedge d \tag{2.6}$$

$$= ((a \wedge b) \vee (b \wedge c)) \wedge d \wedge b \tag{2.7}$$

$$= (a \vee c) \wedge d \wedge b \tag{2.8}$$

となりその論理関数は常に等価であることが分かる．また，冗長付加時に一時的にリテラル数が 5 から 6 に増加するものの, 最終的にはよりリテラル数の少ない簡単な論理関数 (リテラル数 4) を得ることができ，回路規模が小さくなる．

冗長部分回路の付加と除去を繰り返すことによる回路簡単化手法のアルゴリズムは次のようになる．なお，簡単化の対象となる最初に与えられる回路を C とする．なお簡単のために冗長部分回路付加を冗長付加，冗長部分回路除去を冗長除去と呼ぶことにする．

操作 2.3 (冗長付加と除去による回路簡単化)

step(1) 回路 C に対して冗長付加を行なう．

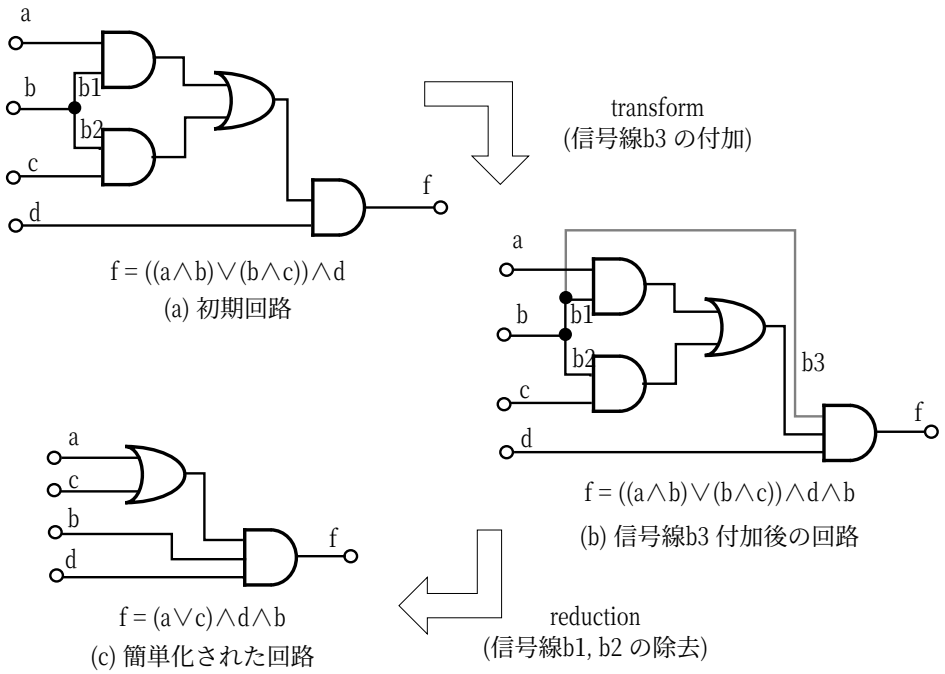


図 2.11: トランスダクション法の例

step(2) 冗長付加前は非冗長であったが，付加後に冗長となった部分回路を見つける．

step(3) 冗長除去を行ない，得られた回路を C' とする．

step(4) 回路 C' が回路 C よりも小さいならば，回路 C' を回路 C とし step(1) に戻る．

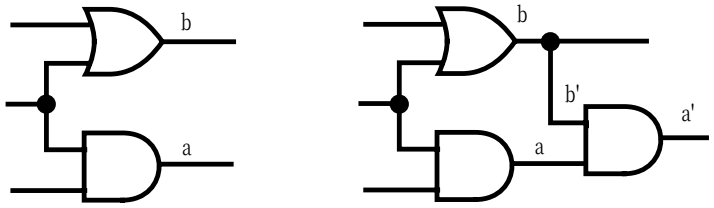
step(5) 他に付加すべき冗長な部分回路があるならば step(1) にもどる．なければ終了．

冗長部分回路を付加する方法は，テスト生成手法の含意操作により得られる含意関係に基づいて行なうことができる [12, 13, 20, 21]．

定義 2.8 (含意関係) 信号線 s に信号値が v を割り当てた場合 $s = v$ と表す．信号値割当 $p = v_p$ から一意に決まる信号値割当 $q = v_q (p \neq q)$ があるとき (v_p,$v_q \in \{0, 1\}$)，この関係を $p = v_p \Rightarrow q = v_q$ と表記し，含意関係と呼ぶ．

図 2.12 に，含意関係に基づいた冗長付加の例を示す．図 2.12(a) で $a = 0$ という信号値割当から含意操作を行なうと，含意関係 $a = 1 \Rightarrow b = 1$ が成立していることが分かる．この含意関係が成り立っている時は，図 2.12(b) のように AND ゲートと信号線からなる冗長部分回路を付加することができる．この部分回路が冗長であることを示すには，信号線 a と a' の論理関数が等価であることを

示せばよい．a が 0 の場合，$a' = b' \wedge 0 = 0$ となり，a が 1 の場合，含意関係 $a = 1 \Rightarrow b = 1$ により，$a' = 1 \wedge 1 = 1$ となるため，信号線 a と a'の論理関数は等価であることが分かる．よって，付加する部分回路は冗長である．



(a) 冗長部分回路付加前の回路　　(b) 冗長部分回路付加後の回路

図 2.12: 冗長付加の例

2.4　あとがき

本章では，論理回路のテスト生成に必要な基礎概念として，テスト方法および故障モデル，そして代表的なアルゴリズムによるテスト生成手法である経路活性化法について述べた．次に，テスト生成手法を用いた回路簡単化手法として，検出不能な縮退故障と冗長部分解路の関係，そして，冗長部分回路の付加と除去による回路簡単化手法について説明した．

第3章　含意操作の拡張

3.1　まえがき

経路活性化法の含意操作において，一意に決まる信号値の数を多く見つけることができれば，テスト生成中に起こりうるバックトラックの回数が減るために，テスト生成手法の高速化につながる．このためいくつかの含意操作を拡張する手法が提案されている [5, 6, 7, 19]．文献 [5] では，静的学習という，含意操作を行なう前に含意関係の対偶を調べて記憶しておき，テスト生成の含意操作時に利用する手法が提案されている．また，文献 [6] では，信号値が一意に決まらない場合でも並列的に信号値割り当てを続け，最後にそれぞれの信号値割当の積集合を調べることで，一意に決まる信号値を得る含意操作が提案されている．

含意操作を拡張することは回路内に成立する多くの含意関係を見つけることになるために，論理簡単化手法にとっても重要である．冗長付加と除去による回路簡単化では，冗長付加を含意関係に基づいて行うが，このときたくさんの含意関係が分かっていれば，それだけ多くの冗長付加が行なえることになり，回路簡単化の能力が高まることになる．

静的学習においてできるだけ多くの含意関係を発見するためには，静的学習に用いる含意操作でより多くの信号値を割り当てる必要がある．そこで含意操作でそれまでの静的学習で得られた含意関係を利用するが，そこで使われる含意関係は過去にどの信号値の割当に対して学習を行ったかにより異なる．このため静的学習の含意操作を行う信号線の順番を操作することで，最終的に得られる含意関係の数が変化する．この静的学習の含意操作を行うときに選ばれる信号線の順番を最適にすることで，より多くの含意関係を求めることができる．しかし，この信号線選択の順序は重要なことであるにもかかわらず，これまでほとんど議論されていない．

本論文では，最初に，静的学習と再帰学習の２つの含意操作の拡張方法について述べる．次に，静的学習における学習順序に着目し，得られる間接含意の数の最大化について考察する．そして，いくつかの異なる学習順序に対して間接含意を求める実験を行い，学習順序と得られる間接含意数との関係を調べる．

3.2　含意操作の拡張

3.2.1　静的学習

経路活性化法における含意操作は与えられた信号値割当から一意に決まる信号値を割り当てる操作である．最初に含意操作の手順を示す.

操作 3.1 (含意操作 (implicatioin procedure))

step(1) 与えられた信号値割当の集合を $ASSIGN$ とする.

step(2) $ASSIGN$ から 1 つの信号値割当 $a = v$ を取り出し，集合 $ASSIGN$ から除く.

step(3) 信号線 a がゲートの入力であり，ゲートの論理演算により出力値が一意に決まるならば，出力値を出力に割り当てる．またその出力値割当を $ASSIGN$ に加える．それ以外の場合はなにもしない.

step(4) 信号線 a がゲートの出力であり，ゲートの論理演算により一意に決まる入力値があるならば，それぞれの入力値を入力に割り当てる．またその入力値割当を $ASSIGN$ に加える．それ以外の場合はなにもしない.

step(5) 信号線 a が分岐の幹であるならば，同じ分岐のすべての枝に信号値 v を割り当てる．またその信号値割当を $ASSIGN$ に加える．それ以外の場合はなにもしない.

step(6) 信号線 a が分岐の枝であるならば，同じ分岐の他の枝と分岐の幹に信号値 v を割り当てる．またその信号値割当を $ASSIGN$ に加える．それ以外の場合はなにもしない.

step(7) $ASSIGN = \phi$ならば，終了．それ以外は step(2) に戻る.

静的学習は, テスト生成の前処理として回路内の含意関係を抽出する操作である．静的学習により抽出した含意関係は, テスト生成中の含意操作で利用することにより，一意に決まる信号値を増やすことができる．得られた含意関係を記憶したものを**含意辞書 (implication dictionary)** と呼ぶことにする．含意関係 $a = 0 \Rightarrow b = 0$ が含意辞書に記憶されている場合，含意操作中に $a = 0$ が割り当てられたら，$b = 0$ を割り当てる．含意辞書を用いた含意操作の手順を示す.

操作 3.2 (含意辞書を用いた含意操作 (implication procedure with implication dictionary))

静的学習により得られる含意辞書を $IMPDIC$ とする.

step(1)-(6) (含意操作の step(1) から step(6) の操作と同じ．)

step(7) $IMPDIC$ 内に $a = v$ からの含意関係があれば，その含意関係に従い信号値割当を行う．またその信号値割当を $ASSIGN$ に加える.

step(8) $ASSIGN = \phi$ならば，終了．それ以外は step(2) に戻る.

含意関係 $a = v_a \Rightarrow b = v_b$が成立する時, その対偶である含意関係 $b = \overline{v_b} \Rightarrow a = \overline{v_a}$もまた成立する．静的学習では含意関係の対偶をしらべることで，含意辞書を作成する．図 3.1 の回路を用いて静的学習における含意関係の抽出操作の例を示す．最初に信号値 $c = 0$ から含意操作を行うと, $d = 0$, $e = 0$, $f = 0$, $g = 1$ の４つの信号値が割り当てられる． これは, 4 つの含意関係

$$c = 0 \Rightarrow d = 0, c = 0 \Rightarrow e = 0, c = 0 \Rightarrow f = 0, c = 0 \Rightarrow g = 1 \tag{3.1}$$

が得られたことを意味する． ここでそれぞれ含意関係の対偶

$$d = 1 \Rightarrow c = 1, e = 1 \Rightarrow c = 1, f = 1 \Rightarrow c = 1, g = 0 \Rightarrow c = 1 \tag{3.2}$$

も成立することになる．

得られたすべての含意関係の対偶を含意辞書に記憶しておくことは，多くの記憶領域を必要とするため好ましくない．含意関係の対偶の中には，図 3.1 の含意関係 $d = 1 \Rightarrow c = 1$ のように，$d = 1$ からの含意操作で求まるものも存在する．そこで含意操作では見つからない含意操作だけを選んで記憶しておくために，次のような学習基準を設け, この基準を満たす含意関係だけ含意辞書に記憶する [5].

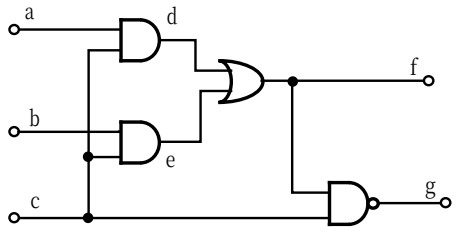


図 3.1: 静的学習の例

定義 3.1 (学習基準 (learning criterion)) 静的学習の含意操作で，含意関係 $a = v_a \Rightarrow b = v_b$ (v_a,$v_b \in \{0, 1\}$) が得られた時，次の 2 つの条件を満たす含意関係の対偶だけを記憶する．

1. 信号線 b を出力に持つゲートのすべての入力の値が非制御値である.
2. 含意操作で信号値割当 $b = v_b$が決まった時に, 信号線 b を出力にもつゲートのすべての入力の信号値が決まっていた.

図 3.1 の例では 含意関係 $c = 0 \Rightarrow f = 0$ だけが学習基準を満たしているため，この含意関係の対偶である $f = 1 \Rightarrow c = 1$ だけを含意辞書に記憶しておくことになる．

静的学習のアルゴリズムは図 3.2 のようになる．内部の 2 つの for ループは，回路の内のすべての信号線の 0 または 1 の信号値から含意操作を行なうことを意味している．静的学習で行なう含意操作

では，それまでに作成された含意辞書を用いた含意操作を行なう (6 行目)．これにより，静的学習中の含意操作で一意に決まる信号値が増え，より多くの含意関係の対偶が得られる可能性が増す．3.3 節以降では，得られる含意関係の対偶を増やすために，静的学習の信号線処理の順番を考慮する．

```
1  static_learning(){
2  {
3      for(each signal line){
4          for(v ∈ {0, 1}){
5              assign line a to value v;
6              perform implication procedure with implication dictionary;
7              if(implication relation a = v ⇒ b = u satisfies the learning criterion)
8                  memorize b = ū ⇒ a = v̄ in implication dictionary;
9          }
10     }
11 }
```

図 3.2: 静的学習のアルゴリズム

3.2.2　再帰学習

再帰学習 (recursive learning) では，含意操作中の複数の信号値割当の候補が考えられる場合に，すべての信号値割当を試み，その結果から一意に決まる信号値割当を得る操作である [6]．図 3.3 の回路において，OR ゲート H の出力値が 1 と決まったときにその入力値は一意には決まらないものの，入力 f と g のどちらかに信号値 1 を割り当てればよいことがわかる．そこで，$f = 1$ として含意操作を続けた場合の信号値割当と，$g = 1$ として含意操作を続けた場合の信号値割当を調べ，どちらを選んでも決まる信号値割当が分かれば，$h = 1$ から一意に決まる信号値割当が分かる．$f = 1$ を選んだ

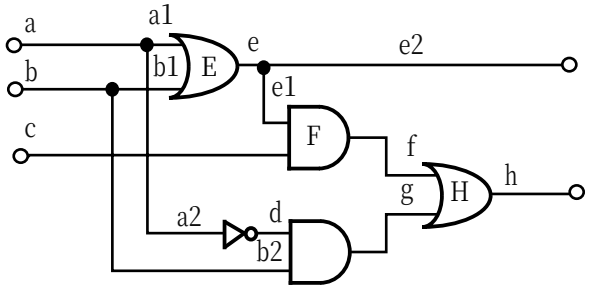


図 3.3: 再帰学習の例

ときは，4 つの信号値割り当て $\{e1=1, e=1, e2=1, c=1\}$ が得られ，$g=1$ を選んだときは，10 個の信号値割り当て $\{b2=1, b=1, b1=1, e=1, e2=1, e1=1, d=1, a2=0, a=0, a1=0\}$ が得られる．この 2 つの信号値割当を比べると，$\{e1=1, e=1, e2=1\}$ が共通するために，これらの信号値は $h=1$ から一意に決まる信号値であることがわかる．

この例のゲート H のように，入力値および出力値が決まったゲートで，$f=1$ または $g=1$ のように同じゲートの入出力線に複数の信号値割当の候補が存在するゲートを**未正当化ゲート (unjustified gate)** と呼ぶ．また，信号値割当 $f=1$ や $g=1$ を**正当化割当 (jusitification)** と呼ぶ．未正当化ゲートと正当化割当は次のように定義される．

定義 3.2 (未正当化ゲート) 未正当化ゲートを定義するために，最初に正当化ゲートを定義する．ゲート G が正当化ゲートである必要条件は，次のようになる．

1. すべての入力値が決まっている，または，
2. 少なくとも 1 つの入力値が制御値である (ただし，ゲート G は EXOR ゲートでない)．

正当化ゲートでないゲートを，未正当化ゲートという．

定義 3.3 (正当化割当) 未正当化ゲートを正当化ゲートにするための，そのゲートの入出力信号線への信号値割当を正当化割当という．

図 3.4 に再帰学習のアルゴリズム **recursive_learning** を示す．**recursive_learning** は再帰呼出しを行ない，パラメータ r が呼出しの深さを表していて，r_max が呼出しの深さの上限を表している．最初に $r=0$ とし，与えられた信号値割当から含意操作を行ない (3 行目)，未正当化ゲートのリスト U_gを作成する (4 行目)．そして，U_gから 1 つの未正当化ゲートを取り出し，そのゲートの正当化割当のリストを作る (7 行目)．そしてそれぞれの正当化割当を回路に与え (9 行目)，r を 1 つ増やし，再帰呼び出しを行なう (10 行目)．呼び出された **recursive_learning** は含意操作を行ない，正当化割当から一意に決まる信号値を計算する．すべての正当化割当から再帰呼び出しを行なった後に，どの正当化割当からも一意に決まる信号値割当 $f_j=v_j$ を得る (12- 15 行目)．以上の説明は再帰呼び出しの深さが 1($r\_max=1$) の場合の説明であるが，再帰呼び出しの深さが 2 になると，もう一度それぞれの **recursive_learning** が同様な操作を行なう．

```
1  recursive_learning(r,r_max)
2  {
3     perform implication procedure;
4     make a list U_g of unjustified gates;
5     if (r < r_max){
6        for(each gate G ∈ U_g){
7           make list J of justification for unjustified gate g;
8           for(each j_k ∈ J){
9              make assignments contained in j_k;
10             recursive_learning(r + 1,r_max);
11          }
12          if (there is one or several signal lines f_j in the circuit,
               which assume the same logic value v_j for all consistent justification j_k ∈ J)
13                   Assign f_j = v_j;
14          Erase all other values assigned in level r;
15       }
16    }
17 }
```

図 3.4: 再帰学習のアルゴリズム

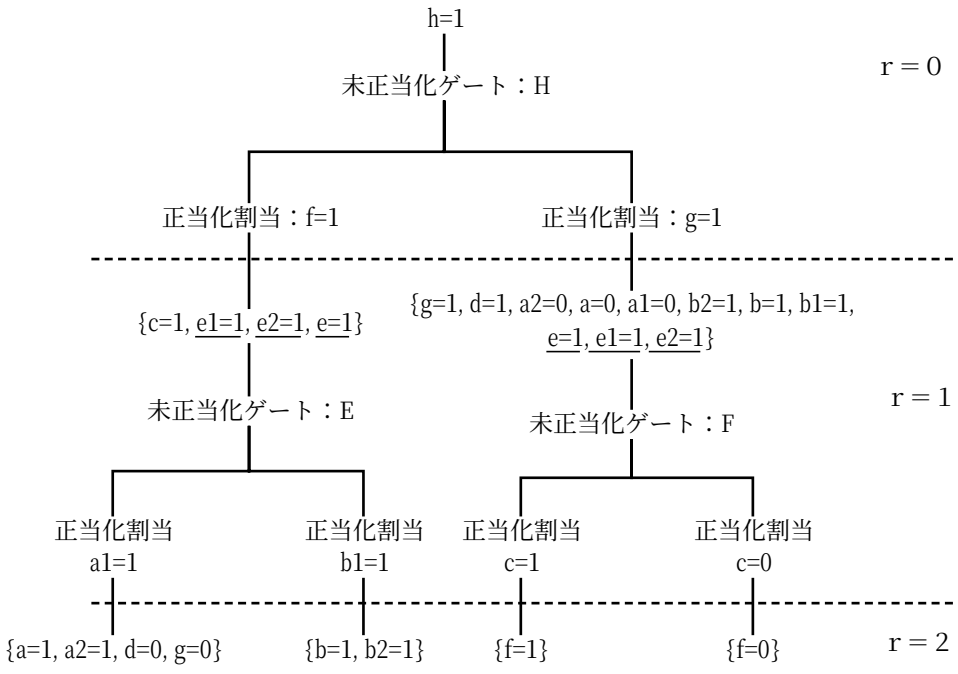


図 3.5: 探索木

図 3.5 は図 3.3 の回路における，$h = 1$ からの再帰学習 ($r\_max = 2$) の様子を表している．この

節の最初に示したように，再帰呼び出しの深さが 1 のときに，含意操作では見つからない 3 つの信号値割当が得られることが分かる．さらに，それぞれの信号値割当に対して未正当化ゲートを調べ，正当化割当を計算し，再帰呼び出しの深さが 2 になるまで処理を続けている．なお，この例では再帰呼び出しの深さが 2 の信号値割当では，一意に決まる新たな信号値は得られない．

静的学習で得られた含意関係や，深さ 1 以上の再帰学習で得られた含意関係は，拡張していない含意操作では得られない．以降本論文では，拡張していない含意操作で得られる含意関係を**直接含意**と呼び，それ以外の，静的学習や深さ 1 以上の再帰学習で得られる含意関係を**間接含意**と呼ぶ．

3.3 学習順序と獲得含意数

静的学習においてできるだけ多くの含意関係を発見するためには, 静的学習時に用いる含意辞書を用いた含意操作でより多くの信号値を割り当てる必要がある．含意辞書に記憶されている含意関係は過去にどの信号値の割当に対して学習を行ったかにより異なる．このため静的学習の含意操作を行う信号線の順番が変われば, 最終的に得られる間接含意の数が変化する．この静的学習の含意操作を行うときに選ばれる信号線の順番を**学習順序**と呼ぶことにする．学習順序はは図 3.2 の 3 行目の **for** ループで信号線を選ぶ順番に相当する．また，図 3.2 の 5 行目で設定する含意操作を開始する信号値割当を**開始信号値割当**と呼ぶ．

図 3.2 のアルゴリズムでは学習順序を考慮していないが, 学習順序により, このアルゴリズムによって得られる間接含意の集合は異なる．図 3.6 の回路において，2 つの学習順序を考える．学習順序 A は f を処理した後に c を処理し，学習順序 B は c を処理した後に f を処理する．学習順序 A は $f = 1$ から含意操作を行っても, 新たに決まる信号値は無いため, 得られる間接含意はない．このため、後に $c = 1$ からの処理では, 先ほど説明した間接含意 $f = 1 \Rightarrow c = 1$ が得られるだけである．一方，学習順序 B では最初に $c = 1$ からの処理により，間接含意 $f = 1 \Rightarrow c = 1$ が得られる．この間接含意が含意辞書にある状態で，$f = 1$ からの処理を行うことになるため, 含意辞書を用いた含意操作により新たに $c = 1, g = 0$ を得

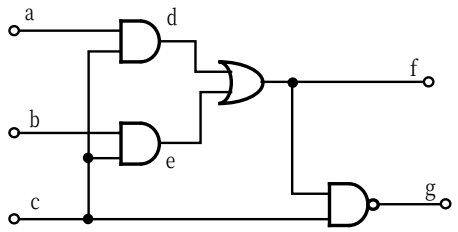


図 3.6: 学習順序の例

ることができる．よって直接含意 $f = 1 \Rightarrow g = 0$ の対偶から間接含意 $g = 1 \Rightarrow f = 0$ が得られる．つまり, この回路において静的学習を行う場合には, 学習順序 B の方が学習順序 A よりも多くの間接含意を学習できることが分かる．この 2 つの学習順序に対する静的学習の結果を表 3.1 に示す．

表 3.1: 学習順序の例

	開始信号値割当	得られた間接含意
学習順序	1. f=1	なし
A	2. c=0	$f = 1 \Rightarrow c = 1$
学習順序	1. c=0	$f = 1 \Rightarrow c = 1$
B	2. f=1	$g = 1 \Rightarrow f = 0$

図 3.6 の回路から静的学習で得られる間接含意は図 3.7 に示すような有向グラフで示される関係を持つ．なお，静的学習では学習中に既に得られた間接含意しか利用しないため，グラフはループをもたない．最も左側に位置する頂点は開始信号値割当を示していて，他の頂点は間接含意を表している．開始信号値割当と間接含意をつなぐ有向辺は，その開始信号値割当から間接含意が得られることを示している．また，間接含意間をつなぐ有向辺は，始点の間接含意が終点の間接含意を得るために必要であることを示している．例えば間接含意 I_4 を得るためには信号値割当 $f = 1$ から含意操作を始めて，間接含意 I_1を利用しなければならない．この図 3.7 に示すように，間接含意には他の間接含意を必要としないもの (I_1, I_2, I_3) と，他の間接含意を必要とするもの (I_4, I_5) に分類される．前者はどのような学習順序で静的学習を行っても得られるが，後者は間接含意間の関係が影響するため，学習順序によって得られる間接含意が異なる．

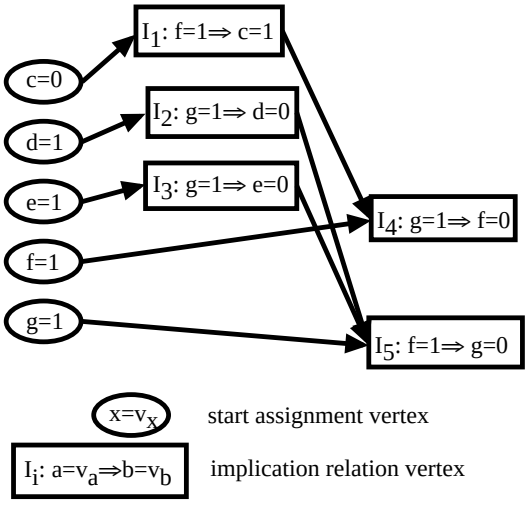


図 3.7: 含意関係間の関係

静的学習で得ることができるすべての間接含意を求めるためには, 静的学習で得ることができる間接含意の数が増加する間, 静的学習を繰り返すとよい．一回目の静的学習ですべての間接含意が学習できたとしても，それを確かめるためにもう一度静的学習を行わなければならないため，最低でも

静的学習は2回行われることになる．一方，静的学習の繰り返し回数の上限値は次のようになる．m を入力が2本以上のゲートの個数であるとすると, 考えられる間接含意数の上限は m 本のゲートの出力から2本を選ぶ場合の数に等しいため, $\frac{m\times(m-1)}{2}$ となる． 最悪の場合を考えているのでこれらの含意が 1 回の繰り返しで 1 つしか見つからないと仮定すると, 静的学習の繰り返し数の上限値は間接含意の数と等しくなるため$\frac{m\times(m-1)}{2}$となる．よって，静的学習を繰り返す場合も，学習順序を最適にすることが重要となる．

3.4 提案学習順序

n 個の開始信号値割当があれば $n!$ 種類の学習順序が存在するため, 最適な学習順序を求めることは困難である．そこで4つの学習順序を提案し計算機実験により有効な学習順序を考察する．提案する学習順序は, 外部入力から幅優先（学習順序 FW）, 外部出力から幅優先（学習順序 BW）, 外部入力から深さ優先（学習順序 FD）, そして外部出力から深さ優先（学習順序 BD）の4種類である.

学習順序を決めるために, 回路内の各信号線のレベルをつぎのように定義する.

定義 3.4 (レベル) 信号線 a に対して, 外部入力から信号線 a までの経路上に存在する2入力以上のゲート数の最大値を I レベルと呼び $Li(a)$ と表す．また，外部出力から信号線 a までの経路上に存在する2入力以上のゲート数の最大値を O レベルと呼び $Lo(a)$ と表す.

それぞれの学習順序は次のような操作で決定する.

操作 3.3 (学習順序 FW 決定操作)
step(1) 学習順序 (信号線の系列)$LO=\phi$，$k=0$ とする.

step(2) $Li(a)=k$ なる信号線 a の信号線系列を作り LO の最後尾に加える.

step(3) 全ての信号線が LO に含まれたら LO を返して，終了.
それ以外は $k=k+1$ として step(2) に戻る.

学習順序 BW の決定操作は, 上記の操作の step(2) において信号線 a が満たす条件を $Lo(a)=k$ に置き換えた操作となる.

操作 3.4 (学習順序 FD 決定操作)
step(1) 学習順序 (信号線の系列)$LO=\phi$とする.

step(2) 任意の外部入力 p を 1 つ選び, p から外部出力方向への経路上にある信号線のうち，まだ LO に含まれていない信号線の集合を S とする．$k=0$ とする.

step(3) $a \in S$ かつ $Li(a) = k$ なる信号線 a の信号線系列を作り LO の最後尾に加える.

step(4) 集合 S の信号線を全て並べ終えたら step(5) へ.
それ以外は $k = k + 1$ として step(3) に戻る.

step(5) 全ての信号線が LO に含まれたら LO を返して，終了．それ以外は step(1) に戻る.

学習操作 BD の決定操作は上記の操作において, 外部入力を外部出力に, 外部出力を外部入力に置き換え, step(3) の信号線 a が満たす条件を $Lo(a) = k$ に置き換えた操作となる.

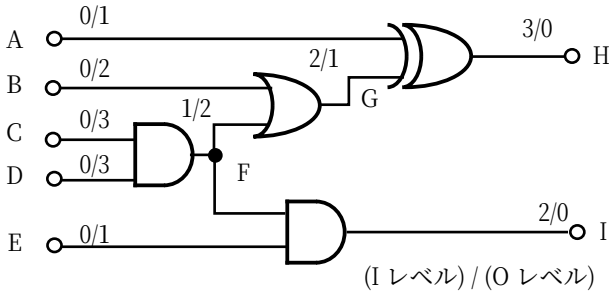


図 3.8: I レベルと O レベル

図 3.8 はある回路の各信号線のレベルを表している．この回路上でのそれぞれの学習順序は次のようになる．$\{A, B\}$ は信号線 A と B の順序は任意であることを示す．また FD, BD は複数ある順序の中の１つを示している.

FW: $\{A, B, C, D, E\}; F; \{G, I\}; H$

BW: $\{H, I\}; \{G, A, E\}; \{B, F\}; \{C, D\}$

FD: $A; H; B; G; C; F; I; D; E$

BD: $H; A; G; B; F; C; D; I; E$

3.5　実験結果

PC (CPU: Pentium pro 150MHz, 主記憶:128 M, OS: FreeBSD) 上で, ISCAS'85 のベンチマーク回路および ISCAS'89 のベンチマーク回路の組合せ回路部分を対象として学習順序の違いを調べた．それぞれの学習順序により得られた間接含意の数を表 3.2 に示す．それぞれの列は左から, 回路名, それぞれの学習順序で得られる間接含意数と最大の間接含意数 ("max")，そして最大の間接含意数に対するそれぞれの学習順序で得られた間接含意数の割合を示している．最大の間接含意数は，静的

表 3.2: それぞれの学習順序による間接含意の数とその割合

回路名	間接含意数					最大間接含意数に対する割合			
	FW	BW	FD	BD	max	FW	BW	FD	BD
c432	66	57	57	57	66	1.00	0.86	0.86	0.86
c499	40	40	40	40	40	1.00	1.00	1.00	1.00
c880	85	85	85	85	85	1.00	1.00	1.00	1.00
c1355	208	208	208	208	208	1.00	1.00	1.00	1.00
c1908	1021	517	648	609	1024	1.00	0.50	0.63	0.59
c2670	1171	843	908	905	1174	1.00	0.72	0.77	0.77
c3540	4687	4063	4157	4466	4714	0.99	0.86	0.88	0.95
c5315	2928	1429	1458	2025	2933	1.00	0.49	0.50	0.69
c6288	1027	663	678	1072	1081	0.95	0.61	0.63	0.99
c7552	9341	3633	4112	5475	9753	0.96	0.37	0.42	0.56
s9234	28596	19639	20964	26765	29593	0.97	0.66	0.71	0.90
s13207	100844	73470	74934	100520	102793	0.98	0.71	0.73	0.98
s15850	39901	24909	34943	40091	41854	0.95	0.60	0.83	0.96
s35932	2811	2811	2811	2811	2811	1.00	1.00	1.00	1.00
s38417	20247	18618	18833	19753	20321	1.00	0.92	0.93	0.97
s38584	252420	239100	240397	249227	253374	1.00	0.94	0.95	0.98

学習を繰り返して行ない，得られる間接含意数が増加しなくなった時の数である．この表3.2 をグラフ化したものを 3.9 に示す．この実験結果が示すように, 得られる間接含意の数は学習順序により大きく異なることが分かる．例えば c1908, c5315, c7552, では, 最大間接含意数に比べて学習順序 BW のそれは約 5 割程でしかないが，学習順序 FW では最大間接含意数にほぼ近い数の間接含意が得られることが分かる．この実験の結果から, 特に多くの間接含意が得られる学習順序は FW であることが分かる．

表 3.3 は最大間接含意数を得るための静的学習回数と学習順序 FW における 1 回の静的学習に必要な時間を示す．この場合も表3.2 と同じように学習順序によって結果が大きく異なることが分かる．特に, c7552 に対する学習順序 BW による静的学習では，最大個の間接含意を得るために 10 回の繰り返し数が必要となる．静的学習の繰り返し数は回路の規模には関係しないが, 回路規模が大きくなると 1 回の静的学習に必要な処理時間が増加する．このため大規模回路で繰り返し回数が 1 回増えただけでも, 処理時間の増加は大きいものとなる.

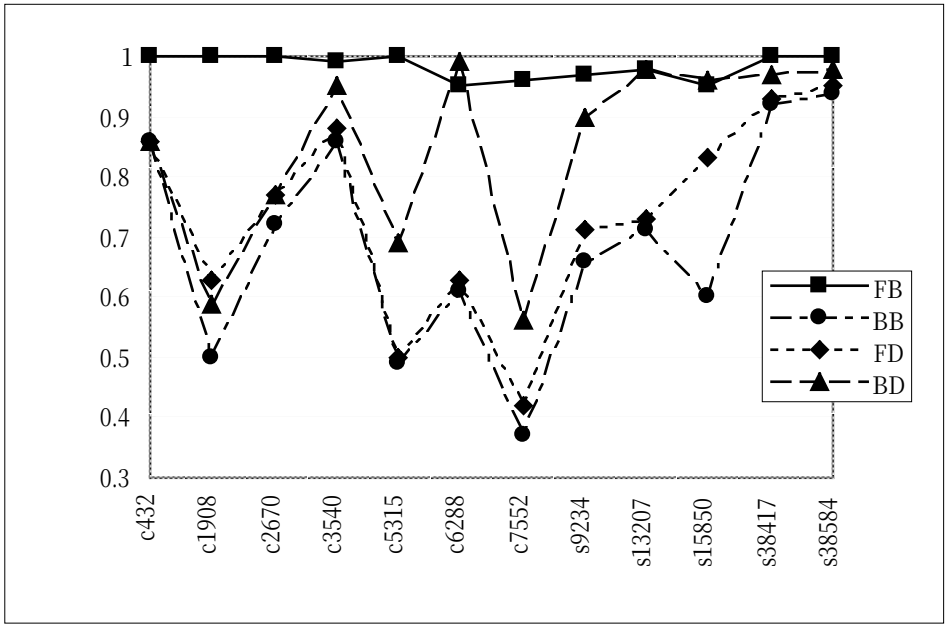


図 3.9: それぞれの学習順序による間接含意の数とその割合

表 3.3: 最大間接含意数を得るために必要な静的学習の繰り返し回数と時間

回路名	静的学習の繰り返し回数				静的学習時間 (sec)
	FW	BW	FD	BD	
c432	1	2	2	2	0.06
c499	1	1	1	1	0.17
c880	1	2	1	1	0.16
c1355	1	1	1	1	1.19
c1908	2	7	6	4	1.24
c2670	2	7	5	6	1.43
c3540	2	4	4	3	6.92
c5315	1	6	6	4	3.97
c6288	2	2	2	2	3.19
c7552	2	10	9	6	10.94
s9234	2	4	4	3	31.83
s13207	2	2	3	2	220.60
s15850	2	3	3	2	67.25
s35932	1	1	1	1	414.26
s38417	2	3	3	2	260.38
s38584	2	4	4	3	1182.00
average	1.62	3.68	3.43	2.68	147.04

最後に，1 つの信号値割当からの含意操作による信号値割当の傾向を見るための実験を行なった．図 3.10 に示すように 1 つの信号線に対して，2 つの領域 A と B を決める．領域 A はある信号線からの外部出力に至るまでの経路上の信号線であり, 領域 B は信号線から外部入力までの経路上の信号線である．実験では，1 つの信号線に信号値 0 および 1 を与えて含意操作を行ない，領域 A と B 内で信号値が決まった信号線の数を数えた．この操作をすべての信号線に対して行ない，領域 A と B 内で値の決まった信号線数の平均値を調べた．表 3.4 はその結果を示す．なお, ファンイン数が 1 のゲートの出力信号線は無視した．ほとんどの回路で領域 A の方が領域 B よりも割り当てられた間接含意の数が多いことがわかるが, これは入力側からの学習順序の方が出力側からの学習順序に比べて, 初期の段階で多くの信号値割当を得られることを意味している．よって, 初期の段階で多くの間接含意が学習でき, 後半の含意辞書を用いた含意操作で多くの信号値を得ることができる．学習順序 FW が，他の学習順序に比べて，より多くの間接含意を得ることができるのはこのためであると考えられる.

表 3.4: 信号値割当の傾向

	割当信号値平均			割当信号値平均	
回路名	A	B	回路名	A	B
c432	1.51	1.04	c6288	1.33	1.71
c499	1.02	1.30	c7552	2.01	1.83
c880	1.72	1.03	s9234	2.96	1.92
c1355	2.98	3.93	s13207	3.28	2.06
c1908	3.00	2.87	s15850	2.81	2.17
c2670	1.81	1.50	s35932	1.16	0.98
c3540	3.25	2.73	s38417	2.14	1.68
c5315	1.67	1.25	s38584	2.31	1.46

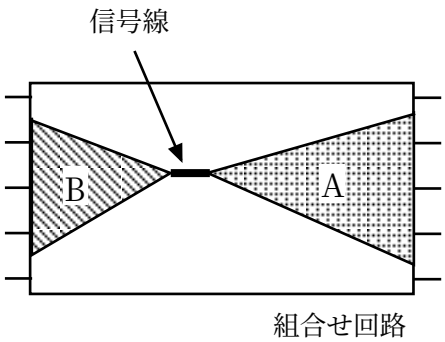


図 3.10: 信号線に対する領域

3.6　あとがき

本章では, 含意操作の拡張手法である静的学習と再帰学習を説明した．さらに, 静的学習において学習を行う信号線の順序が得られる含意関係の数に大きく影響を与えることを示した．実験からは, 入力側に近い信号値から学習する学習順序 FW が効率的であることが分かった．本論文では静的学習で得られる間接含意の数の大小しか比較していないが，数が少なくとも必要な間接含意を含んでいる場合も考えられるため, 今後の研究では回路構造や論理関数に基づいた必要な間接含意の判断も必要である．また, テスト生成, 論理合成, また論路検証に対する実験を行い, 間接含意の有効性を考えていくことも考えられる．

第4章　含意関係の不変性を考慮した回路簡単化

4.1　まえがき

これまでに，縮退故障のテスト生成において，故障の検出可能性から回路の冗長性を判断し，その冗長性を除去することで回路を簡単化できること [14][19]，更に非冗長な回路であっても，冗長な回路を付加することによりそれまで非冗長であった箇所を冗長にし，それらを取り除くことにより更に回路を小さくできることが示されている [20]-[13, 22]．この際の付加される冗長な回路は，静的学習に基づいて回路内部の信号の含意関係を調べることにより探し出すことができ [21, 13]，また，その後新たに冗長になった箇所を見つける処理にも静的学習の結果を利用することができる．このように，静的学習で回路内部の信号の含意関係を求め，それらを利用することは，テスト生成において重要な役割を果たしており，冗長除去や冗長の付加と除去の繰り返しによる回路簡単化にも応用できる．

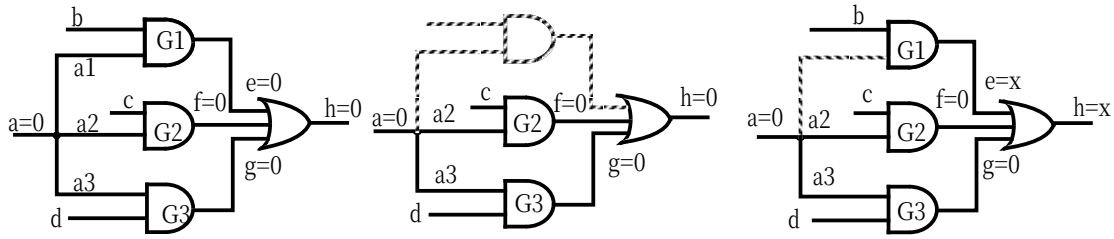
静的学習が一般のテスト生成に用いられる場合には，前処理として一度だけ行えばよいため，総処理時間のうち静的学習に費やされる時間は少ない．しかしながら含意関係は回路構造に依存するため回路の一部を変換すると，変換前の回路に対しては成立した含意関係が回路変換後には成立しなくなる可能性があり，これら成立しなくなった含意関係を用いるなら，テスト生成や論理合成で誤った結果が得られる可能性がある．そのため従来の方法では，回路を変換するごとに，変換前の回路で得られた含意関係を一度すべて破棄し，新たに静的学習を行っていた．従って，何度も回路変換を行う場合には，静的学習が何度も繰り返され，特に規模が大きな回路になると，静的学習に多くの時間が費やされることになる．

本章では，回路内の一部のゲートや信号線を除去するような回路変換において，変換前の回路に対する静的学習で得られた含意関係の回路変換前後の不変性について考察し，部分回路除去で含意関係が無効となるための必要条件を示す．次に，その手法を冗長除去による回路簡単化に応用した手法を提案する．一度に除去される信号線やゲートの数は一般的に少ないため，無効となる含意関係も少ないと考えられる．このため，冗長除去では，最初に静的学習で得られた含意関係の中から無効になる可能性があるものを取り除き，確実に不変なものだけを用いても，冗長判定は十分高速に行えると考えられる．即ち，利用できる含意関係が少なくなるが，一から静的学習を行わないことで計算時

間を短縮できる．

4.2　部分回路変換と含意関係

静的学習により得られた間接含意は，回路の一部を除去することにより成立しなくなる場合がある．例えば与えられた回路が図 4.1(a) のような回路を含んでいるとする．この回路に対して，$a = 0 \Rightarrow h = 0$ という直接含意の対偶である $h = 1 \Rightarrow a = 1$ という間接含意が得られる．今仮に，ゲート G_1とその入出力 a_1, b, e を除去し図 4.1(b) のような回路が得られたとする．この回路に対しても，元の回路で得られた間接含意 $h = 1 \Rightarrow a = 1$ は成立する．しかしながら，信号線 a_1のみを除去したとすると図 4.1(c) のような回路が得られ，この回路に対しては，$h = 1 \Rightarrow a = 1$ という間接含意は成立しなくなる．ここでは例を示さないが，部分回路除去により新たに間接含意が得られる場合があることを記しておく．

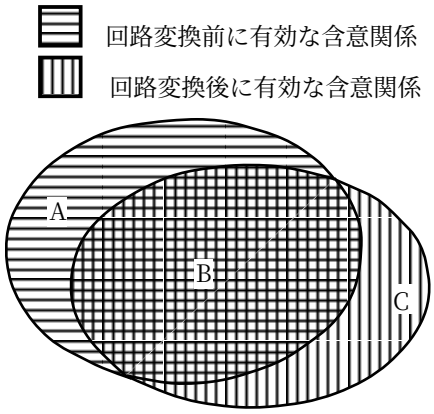


(a) 初期回路　　(b) ゲートG1 を除去後の回路　　(c) 信号線 a1 を除去後の回路

図 4.1: 部分回路除去に対する含意関係の不変性

図 4.2 に，ある部分回路除去の前後で得られる間接含意の関係を示す．A は回路変換によって無効となる間接含意の集合を，B は回路変換後も成り立つ間接含意の集合を，C は回路変換後の回路に対し新たに得られた間接含意の集合を表す．すなわち，$A \cup B$ が回路変換前に有効な間接含意の集合で，$B \cup C$ が回路変換後に有効な間接含意の集合である．部分回路除去を繰り返すときは，回路変換後成り立たなくなった間接含意は変換後の回路において使用できないため，$A \cup B$ の間接含意を一度破棄し，改めて静的学習を行い $B \cup C$ の間接含意を学習し直すことが必要となる．しかし一回の処理で除去される信号線やゲートの数が，数本または数個と少ない場合には，回路内のほとんどの信号線の内部関数は変化しないため，A に含まれる間接含意は少ないと考えられる．これは，多くの間接含意は変換後の回路においても有効であるにもかかわらず，それらを破棄して同じ間接含

意を学習することになり，部分回路除去を繰り返せば計算時間が増加する大きな原因の一つになる．本論文では，このような回路の除去において間接含意の再学習を行わず，A の間接含意を完全に取り除き，残った B の間接含意のみを以後の処理に用いることを試みる．C の間接含意もまた A の間接含意と同様に少ない可能性が高いため，B の間接含意だけでも引き続き行う処理に十分対応できる．静的学習のやり直しを避けることにより，計算時間の短縮が期待できる．B の間接含意だけでは不十分な場合には，そのとき初めて静的学習をやり直せばよい．



A: 回路変換に対して無効となる含意関係
B: 回路変換後も有効な含意関係
C: 回路変換後の回路に対して新たに得られた含意関係

図 4.2: 回路変換前後での含意関係

次節では，図 4.2 の A の部分に相当する無効となる間接含意がどのように生じるかを考察し，それらを取り除く手法について述べる．なお本手法は，無効となる間接含意が生じるための必要条件のみ（必要十分条件ではない）を考慮するため，図 4.2 の B の部分に相当する無効にならない間接含意も除去される可能性があるが，A の無効になる間接含意は必ず除去される．

4.3 部分回路変換における含意関係の不変性

4.3.1 学習関与直接含意

間接含意は含意操作の結果として得られるので，回路の一部を除去したときの間接含意の不変性は，その間接含意を学習するときに用いた含意関係の不変性を考慮すればよい．学習するときに用いられる含意関係は直接含意だけでなく，既に学習された間接含意も含まれる．このため，間接含意

は，直接含意だけ使って学習したものと，直接含意と間接含意の両方を使って学習したものとの２種類に分けることができる．

図 4.3 は静的学習において間接含意を得るために行った含意操作で用いた含意の関係を有向グラフで示したものである．節点 D_i と I_iは，それぞれ直接含意と間接含意を示す．有向辺の始点となる含意は有向辺の終点の間接含意を得るのに必要とした含意である．例えば，間接含意 I_3は直接含意 D_3と間接含意 I_1を用いて学習したことを示している．なお直接含意は，一つの論理ゲートの入出力間での含意（以下，ゲートでの含意），と分岐点の幹と枝の間での含意（以下，分岐点での含意）に分解して表現できるため，D_iはそのどちらかを表す．また，ゲートの複数の入力が制御値を持つ場合のように，複数の含意が重複して信号値を決めることもあるが，本手法では含意操作において最初に適用した含意のみを扱う．このように各間接含意を求めるときに用いた含意は，ループのない有向グラフで表現でき，どの間接含意に至るパスも必ず直接含意から始まることになる．ここで，間接含意 I_iに到達可能な直接含意を I_iの**学習関与直接含意**と呼ぶことにする．このとき，このグラフの性質から，部分回路除去の際の間接含意の不変性に関して次の補題が成り立つ．

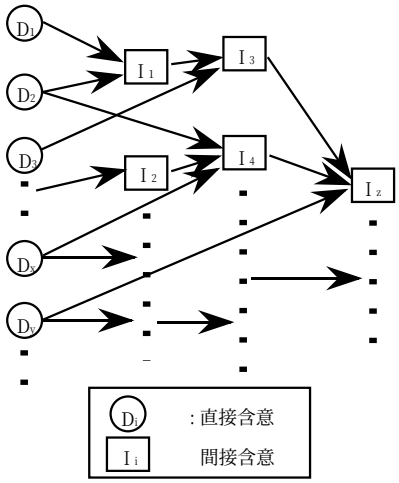


図 4.3: 静的学習に使用した含意関係

補題 4.1 部分回路除去により間接含意 $I_i : a = V_a \Rightarrow b = V_b$ が無効となるための必要条件は，以下の２つの条件のいずれかが成り立つことである．

1. I_i の学習関与直接含意のうち少なくとも１つが無効となる．
2. 信号線 a, b のいずれかが除去される．

補題 4.1 の証明 (1) は，対偶を考察することより自明であり，(2) も，信号線 a または b がなくなれば，間接含意 I_i そのものが無意味になるので明らかである．(証明終り)

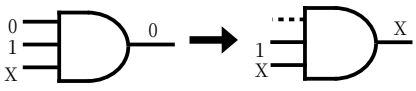
図 4.3 において, 例えば I_3が無効になるとき，学習関与直接含意 D_1，D_2，D_3の内少なくとも１つが無効であるか，または，I_3を表現する信号線が存在しないことになる．上記 (2) の条件を満たす

間接含意を見つけることは容易であるので，以下では，(1) の条件に関して，無効となる直接含意について考察する．ある直接含意が回路の除去前と除去後で異なるとすれば，その原因は，除去された部分と残った部分の境界部分（入力の一部が除去されたゲート，または一部の分岐枝が除去された分岐点）における含意操作で，異なる信号値が生じることにある．これは，回路除去とは無関係な範囲内でしか含意操作が行われないなら，除去前後で決して異なる信号値が生じないので，明らかである．つまり，入力の一部が除去されたゲートか一部の分岐枝が除去された分岐点において，除去前の回路で用いた直接含意 $j = V_j \Rightarrow i = V_i$のうち，信号線 j の除去により，除去されずに残った信号線 i の値を V_iにできなくなる含意関係が含まれていれば，V_iの値の違いが他に波及して，含意操作の結果が異なってくる．このように，除去された回路とされなかった回路で境界部分の直接含意が異なる場合を判断することにより，無効となる直接含意はすべて見つけることができる．

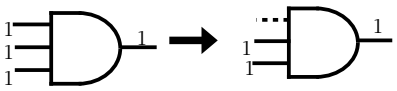
4.3.2 論理ゲートの入力の除去と含意関係

直接含意は，ゲートでの含意関係と分岐点での含意関係の組合せで表現できる．本節では，あるゲートの入力を除去したとき，そのゲートでの除去前の含意関係が無効になる場合を考える．なお，ゲートの出力を除去する場合はゲートそのものが無くなるため考える必要がない．ゲートの入力が決まることにより出力が決まる含意関係は，入力値が制御値を含む場合と，全て非制御値である場合に分けることができる．除去によりあるゲートでの含意関係が無効になるか否かは，除去した信号線で制御値と非制御値のどちらの含意関係を考えているかによる．

図 4.4 に AND ゲートの入力を除去する例を示す．左図が除去前に成り立っていた含意関係を示していて，右図がその含意関係が入力除去後に成立するか否かを示している．含意関係が成立しなくなるのは図 4.4(a) のような場合である．除去前の回路で考えている含意関係は，除去した信号線の値 '0'（制御値）により出力値が決まる含意関係であり，残りの入力の信号値は出力値の決定にには関与していない．よって，変換後の回路ではこの含意関係は無効となる．図 4.4(b) は，除去したゲートの入力に非制御値である '1' を与える含意関係を考えた場合である．このときは残りのゲートの入力値が出力値をそのままに保つため，入力の除去に対して含意関係は不変である．よって次の補題が明らかに成り立つ．



(a) 制御値を持った入力信号線の除去



(b) 非制御値を持った入力信号線の除去

図 4.4: 入力信号線の除去と出力値の関係

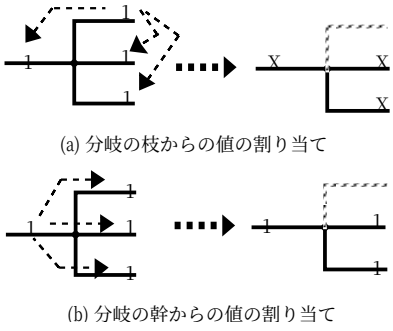
補題 4.2　2 入力以上持つゲート G の入力の集合を $I = \{l_1, \ldots, l_m\}$, ある部分回路除去により除去された G の入力の集合を $RI \subset I$，残った入力の集合を $NRI = I - RI$ とする．また，RI は空集合ではないとする．この部分回路除去により無効となる G での直接含意は，G が EXOR ゲート以外の時，G の出力値が除去前の回路で入力 $l_a(l_a \in RI)$ での制御値により決まる含意関係である．また G が EXOR ゲートの時は, $l_a(l_a \in RI)$ での信号値にかかわらず無効となる．

複数のゲート入力に制御値が割り当てられ，そのうち一部の入力が除去される場合，ゲートの出力は除去後も同じ値を取ることができる．しかし，本手法では，ゲートの出力を決めるため最初に適用した含意関係のみを扱うため，無効とした含意関係には含意操作に影響しないものを含む場合がある．また，EXOR ゲートは，定義からは制御値は持たないが，入力信号線の除去に出力値が影響をうけるため，変換後の回路ではそのゲートでの含意関係は無効となる．

4.3.3　分岐の枝の除去と含意関係

次に，分岐の枝の除去が分岐点での含意関係に与える影響を考える．なお分岐の幹を除去した場合は，その分岐点自体が無くなるため考える必要はない．分岐点における含意関係にはそれぞれ図 4.5(a) と図 4.5(b) の様に，前方から後方への含意関係と後方から前方への含意関係がある．図 4.5(a) は最初に分岐の枝の信号値が決まり，それから幹や残りの分岐の枝の信号値が決まるような，前方から後方への含意関係を示している．この含意関係は，最初に信号値の決まった分岐の枝を除去した分岐点では成り立たたないことになる．即ち，このような含意関係は回路変換後は無効となる．図 4.5(b) は分岐の幹の値が最初に決まり，それから分岐の枝が決まるような，後方から前方への含意

関係を示している．この含意は，分岐の枝のうちの一本を除去した分岐点でも成り立つ．これより次の補題が明らかに成り立つ．



(a) 分岐の枝からの値の割り当て

(b) 分岐の幹からの値の割り当て

図 4.5: 分岐の枝の除去と信号値の変化

補題 4.3 ある分岐点 FP において，分岐の幹を st，枝の集合を $BR = br_1, \ldots, br_m$とし，BR の一部を除去する回路変換により除去された分岐の枝を $RBR \subset BR$ ，残った枝を $NRBR = BR - RBR$ とする．$br_a(br_a \in RBR)$ から $st, br_b(br_b \in NRBR)$ への含意関係は，変換後の回路では無効となる．

4.3.4 無効となる可能性をもつ間接含意

以上の議論に基づいて，部分回路除去により無効となる可能性をもつ間接含意の条件を具体的に示す．これは除去した信号線に対し，ある間接含意を得るときに行った含意操作で，変換前の回路においてどのような信号値を与えていたかを調べることで判断できる．なお，4.3.1 で述べたように，除去したすべての信号線について調べる必要はなく，除去した信号線のうち，回路に残っているゲートまたは信号線に接続していたものだけである．これらの信号線を**境界信号線**と呼ぶことにすると，境界信号線のうち出力側にあるものは，必ず出力が残っているゲートの除去された入力信号線であり，また入力側にある境界信号線は，必ず幹が残っている分岐の除去された枝となる．

ある間接含意を学習するために行った含意操作において，以下の条件のうち１つでも満たせば，その間接含意は無効となる可能性を持つ．

条件 (1) ：出力側の境界信号線に制御信号値を割り当てたとき．

条件 (2) ：EXOR ゲートの入力である境界信号線に信号値が割り当てられたとき．

条件 (3) ：入力側の境界信号線に，同じ分岐点の幹や他の枝より先に信号値を割り当てたとき．

条件 (4) ：含意操作が無効であると判断された間接含意を用いたものであるとき．
条件 (5) ：間接含意に示す信号線を除去したとき．

条件 (1) と条件 (2) は補題２から，条件 (3) は補題３から，条件 (4) は 4.3.1 節での考察から，そして，条件 (5) は補題１から得られる．

4.4　アルゴリズム

ここでは，4.3 節で得られた方法を回路の組合せ回路的冗長の除去に適用する場合について述べる．回路の冗長除去では，一つの冗長信号線を取り除くと他の検出不能故障が検出可能になる場合や，検出可能な故障が検出不能故障になる場合があるため，テスト生成を繰り返すことが求められており [14, 15, 16]，そのため静的学習も繰り返し行われる．そこで，前節で得られた手法を用いて，無効となる間接含意を取り除くことで，静的学習を繰り返し行わない手法を提案する．図 4.6 にそのアルゴリズムを示す．このアルゴリズムは，検出不能故障のクラス化による組合せ回路の冗長除去の方法 [16] を基本としたものであり，太い線で囲った部分が本手法で新たに付加した部分となる．静的学習時には無効となる可能性を持つ間接含意破棄手法の前処理として，図 4.3 のように学習時に必要とした含意関係間の関係を示すグラフに相当する情報の記憶を行っている (*1)．具体的には，回路内の全ての信号線に対してそれぞれの信号線が除去されたときに，３．４の条件 (1)〜(5) により無効となる可能性を持つと判断できるすべての間接含意を記憶している．本手法では無効となる可能性をもつ間接含意の破棄 (*2) だけを考えているため，変換前の回路では得ることができなかったが変換後の回路では得ることができるはずの間接含意（図３の C の部分に相当）を得ることができない．このため，冗長性判定の能力は，静的学習を回路変換の度に繰り返す手法と比べて落ちると考えられる．このアルゴリズムでは，一つの故障に対するテスト生成 (*3) において，３回以内のバックトラックで生成できなかった場合，その故障に対するテスト生成は失敗したとして (*4)，静的学習をやり直している．なお一度静的学習をやり直したにもかかわらずテスト生成に失敗する故障については，本手法で用いたテスト生成アルゴリズムでは判定できない故障なので，２度以上の静的学習のやり直しは行わない (*5)．

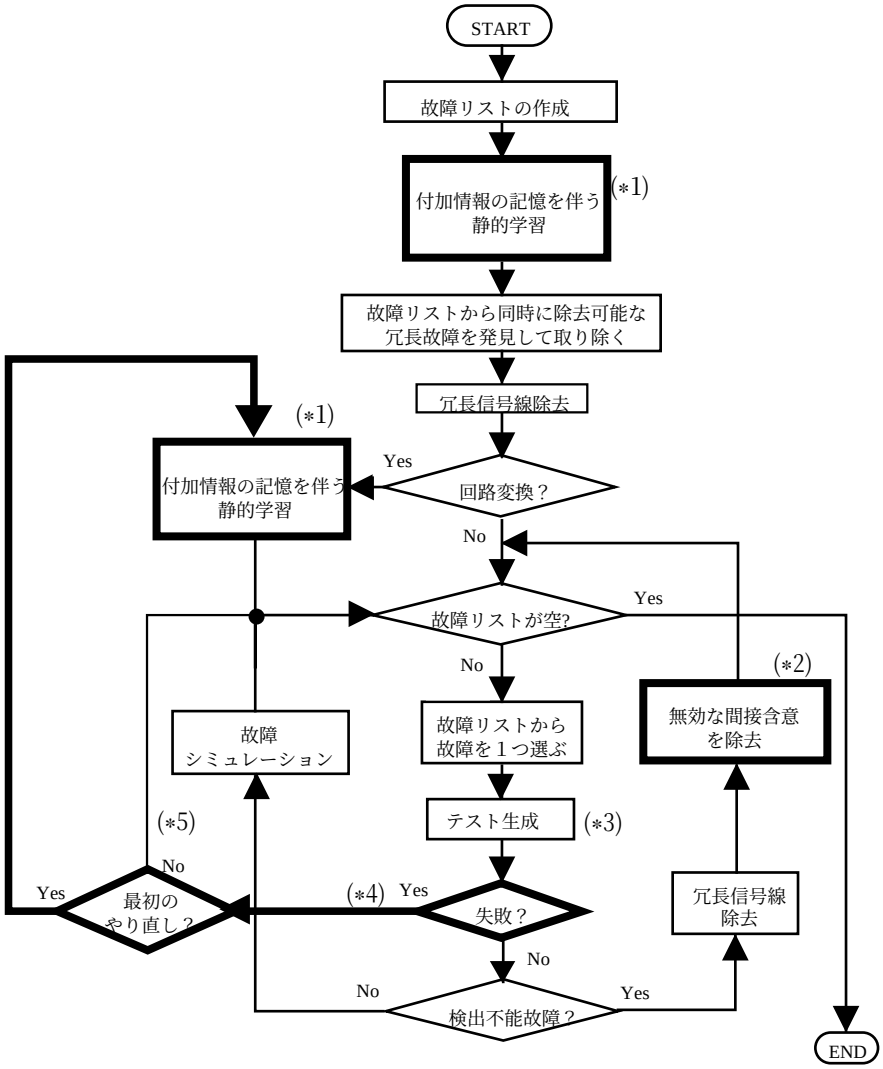


図 4.6: 冗長除去のアルゴリズム

4.5 実験結果

上記のアルゴリズムを富士通 S-4/LC ワークステーション上にC言語を用いてプログラム化し，ISCAS'85 のベンチマーク回路 [41] と ISCAS'89 のベンチマーク回路 [42] の組合せ回路部分を用いて実験を行った．本手法の有効性を示すため，静的学習を回路変換の度にやり直す冗長除去の手法 [16] と比較し，その結果を表 4.1 に示す．表中の 「従来手法」の欄と「提案手法」の「総時間 (sec)」欄にはそれぞれ，文献 [16] の手法による処理時間と提案手法による処理時間を示している．また,「学習回数」と「時間 (sec)」の欄にはそれぞれの手法において静的学習が行われた回数と一回目の静的学習に費やした時間を示している．すなわち「学習回数」と「時間 (sec)」に示す値の積が，それぞ

れの手法の処理時間のうち 静的学習に費やした時間とほぼ等しいことになる．提案手法では，付加情報の記憶のため 1 回に必要な静的学習の時間が増えるものの，静的学習の回数を減らすことで，手法 [16] の静的学習に必要な時間を短縮できることが分かる．また「加速率」の欄には，提案手法が手法 [16] と比較して何倍高速になったかを示している．規模が小さい回路では，1 回の静的学習に必要な時間が短く，また，静的学習の繰り返し回数が少ないため，本手法の有効性は余り認められない．逆に，1 回の静的学習に必要な時間が長く，また，静的学習の繰り返し回数が多い回路 (s13207, s15850, s35932, s38584) では，本手法の有効性が顕著に現れている．特に，s35932 では約 60 倍もの高速化が見られた．唯一，c6288 に対しては，本手法が従来手法より多くの時間を要しているが，これは間接含意の有効性を判断するときに必要な付加情報の計算と記憶のため，従来法と比べて，静的学習により多くの時間を費やしたことが理由として考えられる．最後に，結果として示していないが，提案手法と従来手法では，除去される故障の順序が異なる場合があるため，結果として得られる回路は違うものになる可能性があるが，除去された信号線数やゲート数の差はどの回路においても非常に小さく，この観点から手法の優劣を論ずることはできなかった．また，提案手法において，新たに生じる間接含意をつけ加えていないためテスト生成に失敗して，再学習を必要とした回路は

表 4.1: ベンチマーク回路に対する実験結果

回路	従来手法			提案手法			加速率
	総時間 (sec)	学習回数	時間 (sec)	総時間 (sec)	学習回数	時間 (sec)	
c432	3	5	0.1	3	1	0.2	1.0
c499	4	9	0.1	3	1	0.3	1.3
c880	3	1	0.1	3	1	0.5	1.0
c1355	20	9	0.5	17	1	1.3	1.2
c1908	17	9	0.5	16	2	1.4	1.1
c2670	31	41	0.6	25	2	2.2	1.2
c3540	95	37	1.9	51	2	5.1	1.9
c5315	124	58	1.1	81	2	5.8	1.5
c6288	68	3	0.9	91	1	8.0	0.7
c7552	786	139	2.8	437	2	12.0	1.8
s9234	3445	169	4.6	2887	2	17.2	1.2
s13207	4956	383	13.2	917	2	40.8	5.4
s15850	2806	294	8.3	725	2	39.3	3.9
s35932	73635	657	77.3	1221	1	278.0	60.3
s38417	2995	70	9.1	2898	2	159.4	1.0
s38584	27875	254	110.6	4670	2	354.5	6.1

一つもなっかた.

4.6 あとがき

本章では，部分回路除去に対する間接含意の不変性について考察し，静的学習のやり直しを行わない冗長除去手法を提案した．また，ベンチマーク回路に対する実験は，従来法と比較することで，本手法の有効性を確かめることができた．本手法は，回路の除去する場合を対象とするだけでなく，含意関係を基に回路を付加しながら回路変換を行う手法に対しても，同様の考察を行い効率化することが考えられる.

第5章　含意操作による冗長指摘を用いた冗長付加と除去による回路簡単化

5.1　まえがき

大規模な多段組合せ回路の簡単化において，冗長な部分回路を回路に付加し，その影響で非冗長から冗長に変化した部分回路を除去する方法が提案されている [12, 13, 20, 21, 22]．回路簡単化手法では，最適な論理回路を得ることが目的であるが，現在の半導体デバイスの製品寿命が短いことを考えると，回路簡単化が必要とする処理時間を削減することも重要な要素である．

回路内の冗長な部分回路を見つける手法は冗長指摘と呼ばれる．２章では冗長指摘の能力の高いテスト生成手法を行うことによる冗長指摘方法を紹介したが，処理時間が長いことが問題である．特に回路の規模が大きくなると，冗長指摘は何度も繰り返されるため，回路簡単化の処理時間は冗長指摘の処理時間に大きく依存する．

文献 [17, 18] で提案されている冗長指摘手法はテスト生成を行う代わりに含意操作を用いる．これらの手法は，ある信号線上の信号値の矛盾を仮定し，矛盾する信号値を検出のために必要とする縮退故障を発見する．発見された縮退故障は検出するための信号値が矛盾するため検出不能故障となる．文献 [17] の手法では，冗長部分回路を除去した部分に仮定できる信号値割当の矛盾から冗長指摘を行う．そのため，この手法は冗長付加後の回路には適用できない．文献 [18] は回路の分岐の幹と再収れんしているゲートの入力線上に信号値の矛盾を設定し，そこから含意操作を用いて冗長指摘を行う．この手法は，回路内に存在する任意の冗長信号線を見つけることができるため，冗長付加後の回路に適用できる．

本章では，文献 [18] の手法を基に，冗長付加後の回路に特化した冗長指摘手法を提案する．まず初めに，冗長付加後の回路において，冗長付加前は矛盾がなかったが冗長付加後は矛盾する信号値割当の対を見つける．この信号値の対を**不当割当対**と呼ぶが，このときすべての不当割当対を冗長指摘の対象とする必要はない．冗長付加に用いた含意関係との兼ね合いで，不当割当対の中には検出不能故障の原因となり得ないものが含まれているため，このような不当割当対を冗長指摘に用いないようにすることで，冗長指摘操作の処理を減らすことができる．そして，不当割当対から含意操作を行なうことにより冗長指摘を行なう手順を示す．

5.2　含意関係に基づく冗長付加

提案手法で行なう冗長付加は，２章で述べた信号値の含意関係に基づく冗長付加を行う．より多くの付加する冗長部分回路の候補をえるために，これまで用いてきた含意関係に加えて故障の影響の伝搬を考慮した信号値間の関係も利用する [21]．２種類の含意関係を区別するために，この章では，これまで用いてきた含意関係を C 含意と呼ぶことにし，ここで新たに定義する含意関係を D 含意と呼ぶことにする．D 含意を定義するために，最初に必要割当を定義する．

定義 5.1 (必要割当) 故障を外部出力に伝搬するときに必ず必要となる信号値割当をその故障の必要割当と呼ぶ．

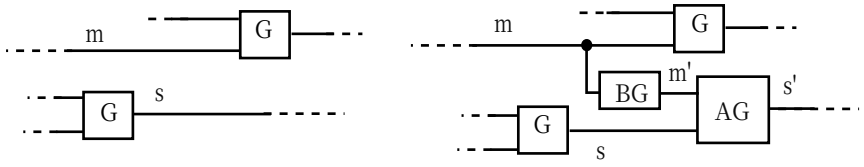
定義 5.2 (D 含意) ある縮退故障 s^v の必要割当 $m = u$ が存在するとき $(v, u \in \{0, 1\})$，信号値 $s = \overline{v}$ と $m = u$ 間の関係を D 含意と呼び，$s = \overline{v} \Rightarrow^D m = u$ と表記する．

D 含意は経路活性化法の故障の顕在化操作，故障の伝搬操作，含意操作を用いることで求めることができる [6]．例えば，2 つの入力 a, b をもつ AND ゲートにおいて，信号線 a 上に 0 縮退故障 a^0 を仮定する．このとき，故障の伝搬操作を行なうと，AND ゲートの出力に伝搬するために信号線 b に 1 が割り当てられる．この $b = 1$ は a^0 の必要割当であるため，D 含意 $a = 1 \Rightarrow^D b = 1$ が得られる．

C 含意と D 含意に基づく冗長付加を図 5.1 と表 5.1 にまとめる．図 5.1(a) と図 5.1(b) はそれぞれ，冗長部分回路を付加する前と付加した後の回路を示している．表 5.1 は図 5.1 のような冗長部分回路付加における含意関係とゲート AG と BG の関係を表したものである．含意関係 $s = 0 \Rightarrow^{(D)} m = 0$ は C 含意 $s = 0 \Rightarrow m = 0$ と D 含意 $s = 0 \Rightarrow^D m = 0$ のいづれかを表している．「BG」の行の「なし」はゲート BG は存在せず信号線 m と m' が直接繋がることを意味し，「NOT」はゲート BG が NOT ゲートであることを意味する．例えば，図 5.1 の信号線 s と m の間に含意関係 $s = 0 \Rightarrow m = 0$ が成り立っていれば，ゲート AG は OR ゲートであり，ゲート BG は存在せず信号線 m と m' が直接繋がることになる．

表 5.1: 成立する含意関係とゲートの種類

含意関係	AG	BG
$s=0 \Rightarrow^{(D)} m=0$	OR	なし
$s=0 \Rightarrow^{(D)} m=1$	OR	NOT
$s=1 \Rightarrow^{(D)} m=0$	AND	NOT
$s=10 \Rightarrow^{(D)} m=1$	AND	なし



(a)冗長部分回路付加前の回路　　(b) 冗長部分回路付加後の回路

図 5.1: 冗長部分回路付加

5.3 含意操作を用いた冗長指摘

5.3.1 基本的なアイデア

2 章ではテスト生成を行うことによる冗長指摘手法として，故障の検出可能性を判定すればよいことを述べた．本章で提案する冗長指摘手法も故障の検出可能性を判断することで，冗長部分回路を発見する．ここでは，テスト生成手法を行なうことによる冗長指摘と，提案する冗長指摘の違いについて述べる．

最初にテスト生成を行なうことによる冗長指摘手法の例を示す．冗長付加として，信号線 e'および h'，ゲート AG を付加した図 5.2 の回路に対して，$e1^1$ が検出可能か否かを判定する場合を考え

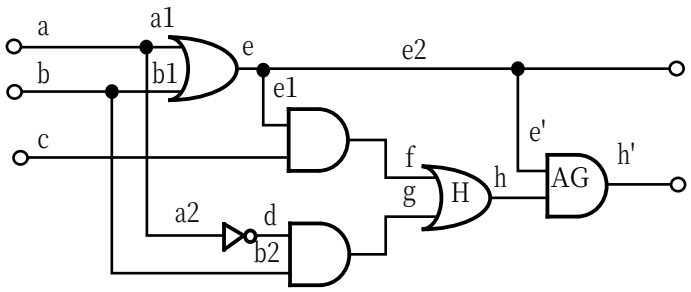


図 5.2: 冗長部分回路付加後の回路

る．$e1^1$ の必要割当は，故障の顕在化操作で割り当てる $e1 = 0$ と，$e = 0$ からの含意操作で割り当てる $e' = 0$ がある．一方，故障 $e1^1$の影響は必ず外部出力 h'に伝搬する必要があるため故障の伝搬操作により $e' = 1$ が必要割当となり $e' = 0$ と矛盾する．この矛盾から故障 $e1^1$は検出不能であると判断できる．同様に故障 $b2^1$のに対しても，$b2^1$の顕在化操作で割り当てる $b2 = 0$ と，故障の影響を信号線 g に伝播するための伝搬操作で割り当てる $d = 1$，そしてこの 2 つの信号値割当から含意操作により割り当てる $e' = 0$ が必要割当となる．一方，$e1^1$の場合と同様に信号線 h'に $b2^1$の影響を伝搬するために $e' = 1$ が必要割当となり矛盾が生じ，$b2^1$ も検出不能故障であることがわかる．

テスト生成を行なうことによる冗長指摘は，1 つ 1 つの故障について必要割当が矛盾するか否かを計算していく．そのため，冗長付加により検出不能となる可能性がある故障の数だけテスト生成を繰り返す必要がある．一般的に冗長部分回路を付加することにより新たに冗長になる故障は数個であると考えられるため，冗長指摘の多くの時間は非冗長な故障のテスト生成を行なっていることになる．

冗長付加により生じる非冗長から冗長に変化する部分回路は，必ず付加した部分回路で信号値割当の矛盾を起こす．上記の例でも信号線 e'で信号値割当が矛盾している．このため，冗長付加前は矛盾していなかったが冗長付加後は矛盾する信号値割当を見つけ，その矛盾する信号値割当を検出のために必要とする故障を見つけることができれば，テスト生成を行なうよりも高速に冗長指摘ができる．冗長付加部分におこる，冗長付加前は矛盾していないが冗長付加後に矛盾する信号値割当の組を不当割当対と呼ぶ．次節では，不当割当対について述べる．

5.3.2　不当割当対

冗長付加は図 5.1(a) の回路から図 5.1(b) の回路への変形として表すことができる．このとき不当割当対は次のように定義される．

定義 5.3 (不当割当対) 信号値割当 $s = v_s$と $m = v_m$を冗長付加後の図 5.1(b) の回路の信号線 s と m に同様に割り当てた時，信号線 s と s'の信号値が異なる場合，これら $s = v_s$と $m = v_m$のペアを不当割当対と呼ぶ．

表 5.2: 冗長部分回路付加による不当割当対

		信号線 s	信号線 m
P1		$\overline{ctr_{AG}}$	$ctr_{AG} \oplus inv$
P2	A	D	$ctr_{AG} \oplus inv$
	B	$\overline{\mathrm{D}}$	
P3	A	$\overline{ctr_{AG}}$	D
	B		$\overline{\mathrm{D}}$
P4	A	D	$\overline{\mathrm{D}} \oplus inv$
	B	$\overline{\mathrm{D}}$	$\mathrm{D} \oplus \mathrm{inv}$

表 5.2 に図 5.1 に示した冗長付加により生ずるすべての不当割当対を示す．なお表中では，ctr_G，$\overline{ctr_G}$はそれぞれ，ゲート G の制御値, 非制御値を表す. inv は図 5.1(b) のゲー INV が存在する場合は 1，存在しない場合は 0 となる変数である．例えば図 5.2 の冗長付加の場合 P1 は $h = \overline{ctr_{AG}} = 1$ と $e2 = ctr_{AG} \oplus 0 = 0 \oplus 0 = 0$ のペアということになる．この場合 $h' = 0 \wedge 1 = 0$ となるため，不当割当対の定義を満たしていることがわかる．

不当割当対から新たに生じた冗長故障を指摘するが，次の補題 5.1 と補題 5.2 が成り立つため，すべての冗長付加に対してすべての不当割当対を考慮する必要はない [49].

補題 5.1 C 含意に基づく冗長付加をした場合，その C 含意は表 5.2 の不当割当対 $P1$ と矛盾する．また，D 含意に基づく冗長付加をした場合，その D 含意は ctr_{AG} が 0 のときは不当割当対 P2-A，P4-A と，ctr_{AG} が 1 のときは不当割当対 P2-B，P4-B と矛盾する．

補題 5.1 の証明

まず C 含意の成立は不当割当対 P1 と矛盾することを示す．図 5.1 の冗長付加が，C 含意に基づいて冗長付加が行われたとすると，用いた C 含意は

$$s = \overline{ctr_{AG}} \Rightarrow m = \overline{ctr_{AG}} \oplus inv \tag{5.1}$$

と表される．このとき P1 の値 $s = \overline{ctr_{AG}}$と $m = ctr_{AG} \oplus inv$ が同時に成立すると仮定すると，明らかに C 含意 (5.1) と矛盾する．

次に後半の D 含意の成立がそれぞれの不当割当対と矛盾することを示す．図 5.1 の冗長付加が，D 含意に基づいて冗長付加が行われたとすると，用いた D 含意は

$$s = \overline{ctr_{AG}} \Rightarrow^{D} m = \overline{ctr_{AG}} \oplus inv \tag{5.2}$$

と表される．この D 含意は，信号線 s 上の ctr_{AG}縮退故障の必要割当の 1 つが $m = \overline{ctr_{AG}} \oplus inv$ であることを示している．このとき ctr_{AG}が 0 であるとすると，信号線 s 上の 0 縮退故障の必要割当が $m = \overline{ctr_{AG}} \oplus inv$ であることを意味しているため， P2-A および P4-A と矛盾する．ctr_{AG}が 1 のときも同様な理由により P2-B, P4-B と矛盾することになる.（証明終り）

補題 5.2 冗長付加後の回路において新たに生じる検出不能故障の必要割当となる可能性のある不当割当対は P1 と P2 だけである．

補題 5.2 の証明

P3 と P4 を原因とする検出不能故障はないことを示せばよい．なお，以下の説明では簡単のためゲートは 2 入力ゲートであるとする．また説明には図 5.3 を用いて行う．

最初に証明で用いる用語を定義する．

[パス] 信号線 l_{p+1}は信号線 l_pの出力信号線であるとき，信号線の系列 $\{l_0, l_1, \ldots, l_p, l_{p+1}, \ldots, l_n\}$ を信号線 l_0から l_nへのパスとする．

[活性化パス] あるパス上のすべてのゲートにおいて，パス上にないすべての入力線の信号値が各ゲートの非制御値であるとき，このパスを活性化パスと呼ぶ.

[故障の伝搬パス] 信号値 D,$\overline{\mathrm{D}}$が割り当てられている活性化パスを故障の伝搬パスと呼ぶ.

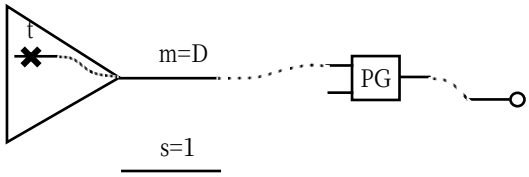
[パリティ] パス上の NAND, NOR, NOT ゲートの総数が偶数個なら 0，奇数個なら 1 と定義される値をパリティと呼ぶ.

[ファンイン信号線] ある信号線 s から入力方向に外部入力または分岐の枝までの経路上にある信号線を信号線 s のファンイン信号線と呼ぶ.

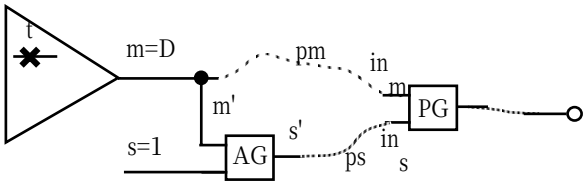
(A) P3 を必要割当とする故障が存在しないことの証明

まず最初に P3 を必要割当とする故障が存在しないことを証明する．冗長付加前の回路として図 5.3(a) の回路を考える．この回路において P3 の 2 つの信号値割当 $m = \mathrm{D}(\overline{\mathrm{D}})$ と $s = \overline{ctr_{AG}}$を必要割当とする非冗長な縮退故障 t^vを信号線 t に仮定する．故障 t^v信号値割当 $m = \mathrm{D}(\overline{\mathrm{D}})$ を必要割当とする仮定から，信号線 t が信号線 m のファンイン信号線であること，故障 t^vの影響が信号線 t から信

号線 m を通って伝搬すること，そして信号線 m から外部出力までの活性化パスが少なくとも 1 つは存在することが必要条件となる．この活性化パスを**初期故障伝搬パス**と呼ぶことにする．



(a) 冗長部分回路付加前の故障伝播経路



(b)冗長部分回路付加後の故障伝播経路

図 5.3: P3 を必要割当とする故障が存在しないことの証明

図 5.3(b) は冗長信号線付加後の回路を表している．ここで $s = \overline{ctr_{AG}}$であるため，故障 t^v の影響は信号線 m を通って信号線 s'にも伝搬することになり，**新しい故障伝搬パス**が生じることになる．故障 t^vが冗長故障となるためには，初期故障伝搬パスと新しい故障伝搬パスがあるゲートで再収れんし，お互いの故障の影響の伝搬を妨げる必要がある．

図 5.3(b) に示したように，2 つの伝搬パスがお互いの故障の伝搬を妨げるゲートを PG とする．また，信号線 $m(s)$ からゲート PG までの故障伝搬パスを $pm(ps)$ と呼び，そのゲート PG の入力を $in_m(in_s)$ と呼ぶことにする．また，信号線 m からパス ps を通って in_m までのパスを m'_ps と呼ぶ．

以下では，ゲート PG が AND, OR, NOR, NAND ゲートの場合と EXOR ゲートの場合にわけて考える．

(A-1) PG が AND, OR, NOR, NAND ゲートの場合

ゲート PG の定義から，次の 2 つの条件が必要条件となる．

条件 (1): パス pm と m'_ps は伝搬パスである．

条件 (2): パス m'_ps のパリティはパス pm のパリティと異なる．

もし両方の条件が満たされれば，$in_m = \mathrm{D}(\overline{\mathrm{D}})$ および $in_s = \overline{\mathrm{D}}(\mathrm{D})$ となり，故障 t^v の影響は外部出力に伝播しなくなる．以下では条件 (2) を満たすために，便宜的に，パス m'_ps のパリティを 1，パス pm のパリティを 0 として考える．

もう一度，図 5.3 を考える．冗長付加を行うために成立している含意関係は，

$$s = \overline{ctr_{AG}} \quad \Rightarrow^D \quad m = ctr_{AG} \tag{5.3}$$

$$s = \overline{ctr_{AG}} \quad \Rightarrow^D \quad m = \overline{ctr_{AG}} \tag{5.4}$$

$$s = \overline{ctr_{AG}} \quad \Rightarrow \quad m = ctr_{AG} \tag{5.5}$$

$$s = \overline{ctr_{AG}} \quad \Rightarrow \quad m = \overline{ctr_{AG}} \tag{5.6}$$

の４種類であるが，２つの C 含意は２つの D 含意にそれぞれ含まれるため，ここでは 2 つの D 含意 (5.3) と (5.4) だけを対象とする．

D 含意 (5.3)$s = \overline{ctr_{AG}} \Rightarrow^D m = \overline{ctr_{AG}}$が成り立っているとする．図 5.3(a) の冗長付加をする前の回路において，信号線 s 上の故障の影響を外部出力にまで伝搬するには，$m = \overline{ctr_{AG}}$と $in_m = \overline{ctr_{PG}}$ が必要である．条件 (1)，(2) に従えば，pm は活性化パスであり pm のパリティが 0 であるため，信号線 in_mと m の信号値は等しくなる．よって，

$$\overline{ctr_{PG}} = \overline{ctr_{AG}} \tag{5.7}$$

となる．

一方，図 5.3(a) の冗長付加をする前の回路において，故障 t^v の影響を外部出力にまで伝搬するためには，pm が伝搬パスであり，$in_s = \overline{ctr_{PG}}$および $s = \overline{ctr_{AG}}$が必要である．条件 (1) によれば，パス ps は活性化パスである．また条件 (2) によれば m'_ps のパリティが 1 であるため ps のパリティも 1 となる．このため，信号線 in_sと s に割り当てられた信号値は異なる値となる．よって，

$$\overline{ctr_{PG}} \neq \overline{ctr_{AG}} \tag{5.8}$$

となる．等式 (5.7) と (5.8) が矛盾することから，条件 (1) と (2) が同時に満たされることはない．よってゲート PG は存在しないことになる．

D 含意 (5.4)$s = \overline{ctr_{AG}} \Rightarrow^D m = ctr_{AG}$の場合についても同様にゲート PG は存在しないことが言える．

(A-2) PG が EXOR ゲートの場合

PG が EXOR ゲートの場合は，条件 (1) が必要条件である．これから，図 5.3(a) の冗長付加をする前の回路において，pm は活性化ゲートであるため，in_mの信号値を 0 にしたときと，1 にしたときでは信号線の m の信号値は異なることになる．これは，冗長付加に利用した含意関係から m の値が一意決まることと矛盾する．よって，条件 (1) は満たされない．

以上のことから P3 を必要割当とする故障は存在しない．

(B) P4 を必要割当とする故障が存在しないこと証明

次に P4 を必要割当とする故障が存在しないことを示す．冗長付加後の回路において P4 を必要割当とする故障 k^vを考え，k^v が存在する信号線を k とする．信号値 $m = \mathrm{D}(\overline{\mathrm{D}}), s = \overline{\mathrm{D}}(D)$ が必要割り当てであるため，図 5.4 のように信号線 k は信号線 s と m のファンイン信号線であり，k から s, mまでのパスは活性化パスである．

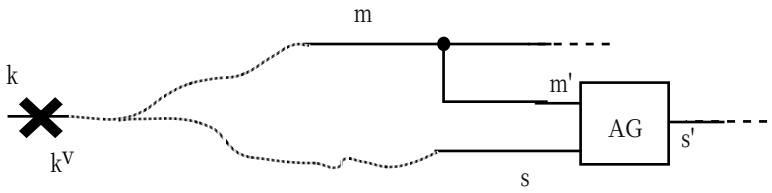


図 5.4: P4 を必要割当とする故障が存在しないことの証明

一方，補題 5.1 から $s = \overline{ctr_{AG}} \Rightarrow m' = \overline{ctr_{AG}}$ が成り立っているので，$m' = s = \overline{ctr_{AG}}$ となる．これは，常に信号線 k から 信号線 s, m までのパスのパリティが同じであることを意味する．よって，冗長付加前の回路において P4 が成り立つことはないため，故障 k^vは存在しない． (証明終り)

補題 5.1 と補題 5.2 より簡単に次の定理が導かれる．

定理 5.1 C 含意に基づく冗長付加をした場合は，表 5.2 の不当割当対 P2 だけが冗長信号線付加によって生じる新しい検出不能故障の必要割当となる．一方，D 含意に基づく冗長信号線付加をした場合，ctr_{AG} が 0 のときは P1 および P2-B が，ctr_{AG} が 1 のときは P1，P2-A がそれぞれ冗長信号線付加によって生じる新しい検出不能故障の必要割当となる．

定理 5.1 によると，4 つの必要割当のうち 2 つの必要割当 P3,P4 は考慮する必要がないことが分かる．そのため，冗長指摘の処理時間は定理 5.1 を考えることで半分以下にすることができる．

5.3.3　含意操作を用いた冗長指摘

不当割当対から生成冗長信号線を指摘する方法，すなわち不当割当対を必要割当とする縮退故障を指摘する方法について述べる．

ある信号値割当を必要割当とする縮退故障は，信号値の反転値からの含意操作により判定することができる．図 5.5 のように，信号値割当 $a = v_a$を必要割当とする縮退故障を見つけることを考える．信号値割当の信号値の反転値 $a = \overline{v_a}$から含意操作を行って，信号値割当 $b = v_b$が得られたとする．これは含意関係 $a = \overline{v_a} \Rightarrow b = v_b$が成り立っていることになる．この含意関係の対偶 $b = \overline{v_b} \Rightarrow a = v_a$ に注目すると，$b = \overline{v_b}$を必要割当とする縮退故障は，$a = v_a$も必要割当とすることがわかる．$b = \overline{v_b}$ を必要割当とする縮退故障は２種類考えられる．１つは信号線 b 上の v_b縮退故障である．もう１つは，もし$\overline{v_b}$が信号線 b を入力に持つゲート G_Bの制御値ならば，信号線 b 以外のゲート G_Bの入力のファンイン信号線上の 0 および 1 縮退故障である．

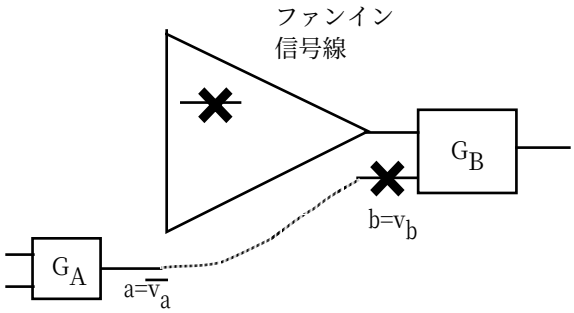


図 5.5: 信号値割当 $a = v_a$を必要割当とする縮退故障

以上をまとめると次のようになる．

補題 5.3 (信号値割当を必要割当とする縮退故障) 信号値割当 v_aを必要割当とする縮退故障は，信号値割当 v_aの反転値からの含意操作を行なった後に得られる，以下のような縮退故障である．

1. 信号値 0(1) が割り当てられた信号線上の 0(1) 縮退故障.
2. ゲート G の入力信号線にゲート G の制御値が割り当てられた場合，他の入力信号線のファンイン信号線上の 0 および 1 縮退故障.

定理 5.1 から，2 つの冗長指摘操作が提案できる．1 つは C 含意にもとづく冗長付加を行った場合

の冗長指摘操作で，もう 1 つは D 含意に基づいて冗長付加を行った場合の冗長指摘操作である．定理 5.1 によれば，C 含意に基づく冗長信号線付加を行った場合は，不当割当対 P2 を必要割当とする故障が発見の対象となる故障である．不当割当 P2 の $s = \overline{\mathrm{D}}(\mathrm{D})$ は，信号線 s のファンイン信号線上の故障の影響が信号線 s を通って伝搬できないことを意味するため，すなわち，信号線 s のファンイン信号線上の故障はすべて $s = \overline{\mathrm{D}}(\mathrm{D})$ を必要割当とする縮退故障となる．

操作 5.1 (C 含意に基づく冗長指摘操作)

step(1) 不当信号値割当 P2 の $m = ctr_{AG} \oplus inv$ に対して，信号値$\overline{ctr_{AG} \oplus inv}$を信号線 m に与え，含意操作を行う．$m = ctr_{AG} \oplus inv$ を必要割当とする縮退故障の集合を $F1$ とする．

step(2) 不当信号値割当 P2 の $s = D(\overline{\mathrm{D}})$ に対して，信号線 s のファンイン信号線上の 0 および 1 縮退故障の集合を $F2$ とする．

step(3) 指摘する検出不能故障は $F1 \cap F2$ となる．

一方，定理 5.1 によれば，D 含意に基づく冗長信号線付加に対する冗長信号線指摘は，不当割当対 P1 および P2 から行えばよい．

操作 5.2 (D 含意に基づく冗長指摘操作)

step(1) 不当信号値割当 P1 と P2 の $m = ctr_{AG} \oplus inv$ に対して，信号値$\overline{ctr_{AG} \oplus inv}$を信号線 m に与え，含意操作を行う．$m = ctr_{AG} \oplus inv$ を必要割当とする縮退故障の集合を $F1$ とする．

step(2) 不当割当対 P1 の $s = \overline{ctr_{AG}}$に対して，(1) と同様の操作を行い，得られる縮退故障の集合を $F2$ とする．

step(3) 不当信号値割当 P2 の $s = \mathrm{D}(\overline{\mathrm{D}})$ に対して，信号線 s のファンイン信号線上の 0 および 1 縮退故障の集合を $F3$ とする．

step(4) 指摘する検出不能故障は $F1 \cap (F2 \cup F3)$ となる．

5.4 冗長部分回路の同時除去性

発見した冗長部分回路が常に同時に除去できるとは限らない．図 5.6 のような冗長付加後の回路が与えられたとする．信号線 $s, k2, s'$ とゲート AG が付加した冗長信号線およびゲートであり，このとき新たに検出不能となった故障は $b1^1$ と $a1^1$である．ここで信号線 $b1$ を除去した場合，$a1^1$の指摘に必要な含意関係 $s = 0 \Rightarrow d = 0 \vee a1 = 1$ が無効となり $a1^1$は検出可能となる．同様に信号線 $a1$ を除去した場合も含意関係 $s = 0 \Rightarrow b1 = 1 \vee a1 = 0$ が無効となり $b1^1$は検出可能となるため，信号線 $b1$ と $a1$ はどちらも冗長信号線であるにもかかわらず，同時に除去することができない．指摘した

冗長信号線がある信号線を除去した回路に対しても冗長か否かを知るためには，冗長指摘に用いた含意が有効か無効かを判定しなくてはならない．そのため，第４章で述べた冗長除去に対して無効になる含意を指摘して削除する手法を用いて冗長指摘時に用いた含意操作における含意関係の有効性を調べ，冗長部分回路の同時除去性を判断する．

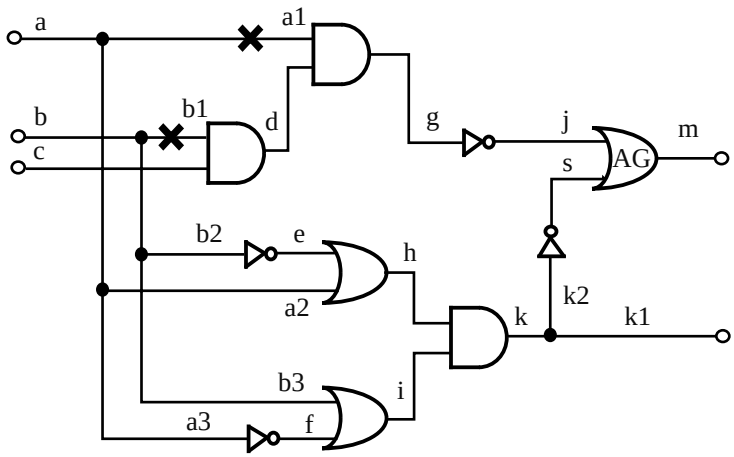


図 5.6: 冗長部分回路の同時除去

5.5　実験結果

以上の冗長指摘操作を用いた回路簡単化手法を計算機上で実現し，ISCAS'85 のベンチマーク回路 [41] に対して実験を行った．また，テスト生成による冗長指摘を用いた従来法である回路簡単化手法 [16] に対しても同様に実験を行い，計算時間と回路簡単化能力の比較を行った．用いた計算機は，Pentium-133MHz を CPU にもつ PC である．なお，実験に用いたテスト生成手法は故障シミュレーションと決定論的なテスト生成の両方を行うものである．また，冗長付加に利用した C 含意を見つけるための操作と，冗長指摘操作中の含意操作は，再帰学習を利用した．そのときの，再帰呼び出しの深さは 2 とした．

表 5.3: C 含意に基づく冗長付加と除去による回路簡単化

	初期	テスト生成による手法		提案手法		２つの手法の比較	
回路	リテラル	リテラル	時間 (sec)	リテラル	時間 (sec)	簡単化率	加速率
c432	306	272	71	271	9	0.99	7.89
c499	375	375	66	375	4	1.00	16.50
c880	650	603	215	622	31	1.03	6.94
c1355	999	790	2393	790	146	1.00	16.39
c1908	1284	940	5541	953	724	1.01	7.65
c2670	1346	1266	1675	1262	169	0.99	9.91
c3540	2343	2097	80325	2116	5482	1.00	14.65
c5315	3866	3365	253962	3242	3043	0.96	83.46
c6288	4698	3771	90147	3770	3307	0.99	27.26
c7552	5117	N/A	N/A	3908	21486	N/A	N/A
					平均	1.00	21.18

N/A: 処理時間による打ち切り

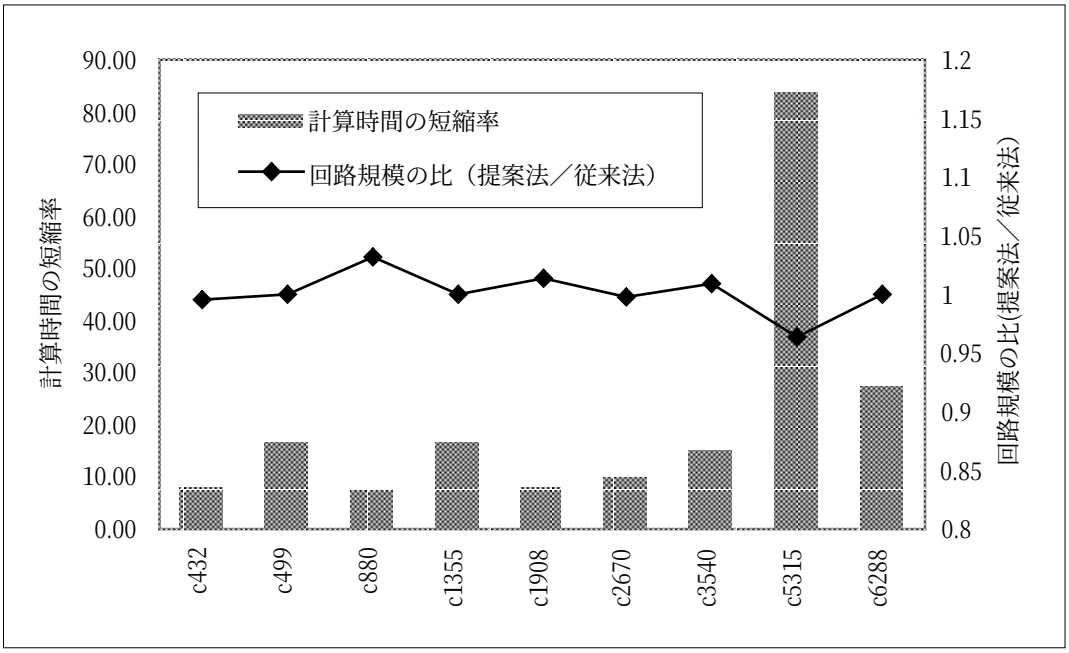


図 5.7: C 含意に基づく冗長付加と除去による回路簡単化

表 5.3 に C 含意に基づいて冗長付加をした場合の実験結果を示す．それぞれの列は左から順に，回路名，初期回路のリテラル数，テスト生成を行うことによる冗長指摘手法を用いて簡単化した回路の

リテラル数と処理時間，提案した冗長指摘手法を用いて簡単化した回路のリテラル数と処理時間，そして 2 つの冗長指摘手法の比較である簡単化した回路規模の割合 (提案手法を用いた場合のリテラル数／テスト生成による冗長指摘を用いた場合のリテラル数) と処理時間の加速率 (テスト生成による冗長指摘を用いた場合の処理時間 / 提案手法を用いた場合の処理時間) を示す．表 5.3 の結果をグラフ化したものを図 5.7 に示す．この実験結果から，提案手法による処理時間の加速率の平均は約 21 倍であり，大幅な計算時間の削減が行えた．またテスト生成による手法と提案手法の回路簡単化能力の比較は，リテラル数の比が平均 1.00 であることから，ほぼ同等であることが分かる．特に c5315 に対する実験では，83.46 倍もの高速化が可能であったと同時に，その簡単化能力も多少向上していることがわかる．

表 5.4 に D 含意に基づく冗長付加と除去による回路簡単化の結果を示す．この結果も表 5.3 と同様に，テスト生成を行なうことによる冗長指摘と，提案する冗長指摘を比較しているが，提案手法が処理時間を短縮していることが分かる．

表 5.4: D 含意に基づく冗長付加と除去による回路簡単化

	テスト生成による手法		提案手法		2 つの手法の比較	
回路	リテラル	時間 (sec)	リテラル	時間 (sec)	簡単化率	加速率
c432	291	846	288	42	1.01	20.14
c499	368	601	368	59	1.00	10.19
c880	589	3021	605	152	0.97	19.88
c1355	992	6033	992	347	1.00	17.39
c1908	1218	22508	1239	6143	0.98	3.66
c2670	1316	86661	1326	12055	0.99	7.19
c5315	3624	699626	3776	13700	0.96	51.07
c6288	4698	39996	4698	1701	1.00	23.51
c7552	N/A	N/A	4933	77627	N/A	N/A
				平均	1.01	19.13

N/A: 処理時間による打ち切り

5.6　あとがき

冗長部分回路付加と除去による論理回路簡単化手法において，処理時間増加の問題を解決するために，テスト生成を行うことによる冗長指摘手法の代わりに含意操作を用いた冗長指摘手法を提案

した．その手法は，冗長付加部分に生成する信号値の矛盾である不当割当対を必要割当とする故障を含意操作を用いて指摘するものである．そのため本研究では，不当割当対に対する議論を行い，その不当割当対から故障を指摘するための操作を示した．また，ベンチマーク回路を用いて実験を行い，手法の簡単化能力はそのままで処理時間が平均 20 分の 1 に短縮できることを示した．

第6章　テストベクトル数が制限された条件下でのテスト生成

6.1　まえがき

テスト生成手法は，(1) テスト生成時間を最小にする，(2) 生成するテストベクトルの故障検出率および故障検出効率を最小にする，(3) 生成するテストベクトル数を最小にする，という3つの要求を満足することを目標に開発が行われてきた [1, 2]．近年のテスト生成の研究では, テスト印加時に必要な時間やテスターのテスト記憶用メモリの削減のために, よりコンパクトなテストベクトル集合およびテストベクトル系列を生成することが重要となってきている [26]-[35]．特に IDDQ テストのためのテスト生成では, IDDQ 値の測定に時間を要するため, コンパクトなテストベクトル集合に対する要求は高い [30]-[35]．

順序回路では，順序回路が持つ記憶素子 (FF) のため回路の内部状態を考えてテスト生成を行なう必要がある．そのため，得られたテストベクトルはある順序をもって印加されるテストベクトル系列となる．順序回路の IDDQ テストベクトル系列は，2種類のテストベクトルからなる．1つは I_{DDQ}を測定するための**測定ベクトル**，もう1つは回路の状態を遷移することが目的の**遷移ベクトル**である．N 個の測定ベクトルと M 個の遷移ベクトルをからなる長さ $M+N$ のテストベクトル系列のテスト時間は次式で与えられる．

$$Testing\_time = N \times CL_m + M \times CL_t \tag{6.1}$$

ここで, CL_m と CL_tはそれぞれ, 測定ベクトルと遷移ベクトルを被テスト回路に印加する時のクロック間隔である．一般的に，状態遷移は通常動作時の早いクロック周波数で行なうことができるが，I_{DDQ} の測定時のクロック周波数は通常動作時のクロック周波数にくらべて非常に遅いため，$CL_m \gg CL_t$ となる．このため, テスト時間はほぼ N に比例すると考えてよく，順序回路の IDDQ テストでは測定ベクトル数を減らすことがテストベクトル系列を圧縮することになる．

測定ベクトルを減らすためのテストベクトル系列の圧縮手法が提案されている [31, 34, 35] が, これらの手法では，選ばれた測定ベクトルが，すべてのベクトルを測定ベクトルとした時の故障検出率を維持していることが必要条件となっている．このため，選ばれた測定ベクトル数が最小でも，その数が非常に大きければ，現実的にすべての測定ベクトルに対して IDDQ を測定できない可能性が

ある．

一般に故障検出率とテスト時間はトレードオフの関係にあるため，テストコストは故障検出率とテスト時間の関数として考えられるべきである．特に IDDQ テストのようにテスト時間が長くなる場合は，テストコストにおけるテスト時間の重要性は大きく，故障検出率の最大化だけを考えていたのではテストコストを最小にすることはできない．

本章では, 順序回路の IDDQ テストにおいて，測定ベクトル数に上限が存在する条件下で, より多くの故障を検出できる測定ベクトルを得るための新しいテスト手法を考える．測定ベクトル数に上限がある場合のテスト生成では, 全故障を対象とした故障検出率に従って生成したテストベクトル系列と代表故障を対象とした故障検出率に従って生成したテストベクトル系列に違いが生じるため, 全故障を対象とした故障検出率を求めることが特に重要となる．本章では, 全故障を対する故障検出率を計算するために，重みつき故障リストを用いて代表故障からでも全故障を対象とした故障検出率を計算できるための方法を示す．そして，与えられたテストベクトル系列からより多くの故障を検出できる測定ベクトルを選択する手法を提案する．提案手法では，故障シミュレーションにより得られる故障の検出回数からベクトルの評価値を計算し，この評価値に従って測定ベクトルを選ぶ．

6.2　測定ベクトル数の制限下でのテスト生成問題

測定ベクトル数と最大故障検出率の関係は, 一般に図 6.1 のようになる [37, 38]．図中の曲線は各測定ベクトル数で得ることができる最大の故障検出率を示している．文献 [31, 34, 35] の手法では, テストベクトル系列の最大故障検出率を保ちつつ, 測定ベクトル数を最小とすることが目的であるため, 図 6.1 の A 点に相当する測定ベクトル集合が最適解となる．一方, 本論文で扱う測定ベクトル選択問題は, 測定ベクトル数の上限が与えられている時に, 最大の故障検出率を得るベクトルを選択することが目的である．測定ベクトル数の上限が N 個の場合, 最適解は図 6.1 の点 B の故障検出率をもつテストベクトル系列を求めることである.

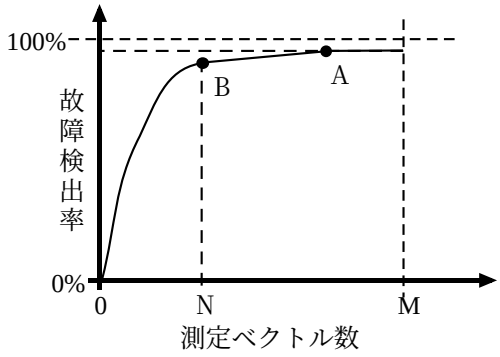


図 6.1: 測定ベクトル数と故障検出率の関係

6.3 故障検出率の計算方法

6.3.1 全故障に対する故障検出率の計算の重要性

第 2 章で述べたように，等価故障の中でテスト生成の対象に選ばれる故障を代表故障という．ここで，代表故障を対象として計算した故障検出率を代表故障検出率と呼ぶことにし, (検出できる代表故障数) ／ (すべての代表故障数) $\times 100(\%)$ と定義する．また, 代表故障を選ぶ前のすべての故障を対象に計算した故障検出率を全故障検出率と呼び, (検出できる故障数) ／ (すべての故障数) $\times 100(\%)$ で定義する.

測定ベクトル数に上限がない一般のテスト生成では, 次の補題 1 が成り立つ.

補題 6.1 測定ベクトル数の上限がないときのテスト生成の場合, 最大の代表故障検出率を持つテス

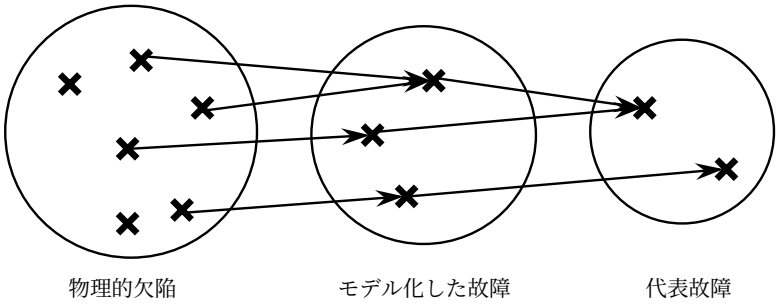


図 6.2: 物理的欠陥のモデル化と代表故障

トベクトル系列は, 最大の全故障検出率を持つテストベクトル系列である.

補題 6.1 の証明

最初に代表故障検出率が 100 %の場合を考える．図 6.2 に示したように, 全故障集合の代表故障集合への写像は全射である．このため, すべての代表故障を検出できれば, すべての故障は検出できることになる．よって, 補題 1 が成立する.

次に, 代表故障検出率が 100 %ではない場合を考える．測定ベクトル数に制限がないため, テストベクトル系列が生成できない代表故障は, 検出不能故障である．したがって, この代表故障と等価な故障はすべて検出不能故障となる．よって, 最大の全故障検出率を持つテストベクトル系列集合は, 最大の代表故障検出率をもつテストベクトル系列と等しい．よって補題 1 が成立する．(証明終り)

測定ベクトル数に上限がない場合のテスト生成においては, 一般的には全故障検出率に従ってテストを生成しているものの, 補題 1 によれば, 代表故障検出率に従ってテスト生成を行っても問題がないことがわかる．ところが, 測定ベクトル数に上限があると補題 1 は成り立たない.

補題 6.2 測定ベクトル数に上限があるときのテスト生成の場合, 最大の代表故障検出率を持つテスト集合が, 最大の全故障検出率を持つとは限らない.

補題 6.2 の証明

4 つのテストベクトルからなるテストベクトル系列 $T = \{t_1, t_2, t_3, t_4\}$ から，それぞれ全故障と代表故障を対象として 2 つの測定ベクトルを選ぶ場合を考える．図 6.1 はそれぞれのテストベクトルを測定ベクトルとした時に検出できる全故障と代表故障を示している.「検出される全故障」のなかで, { と } で囲まれた故障は等価故障であり，等価故障の内最初に示した故障が代表故障である．例えば, ベクトル t_1 を測定ベクトルとすると，f_1, f_2, f_3 の 3 つの全故障が検出できるが，代表故障だと f_1, f_3 の 2 つの故障が検出される計算になる.

表 6.1: 全故障と代表故障の故障表

テスト	検出される全故障	検出される代表故障
t_1	$\{f_1, f_2\}, f_3$	f_1, f_3
t_2	f_4, f_5	f_4, f_5
t_3	$\{f_6, f_7, f_8, f_9\}$	f_6
t_4	f_{10}	f_{10}

全故障を対象として測定ベクトルを選んだ場合，それぞれの検出される故障の集合は独立である

ため，t_1とt_3を選べば，最大の全故障検出率を得ることができる．一方，代表故障を対象にして測定ベクトルを選ぶ場合は，t_1とt_2 が最大の代表故障検出率をもつ測定ベクトルの選択であることになり，全故障を対象にして測定ベクトルを選んだ場合とことなる．よって，測定ベクトル数に上限があるときのテスト生成の場合, 最大の代表故障検出率を持つテスト集合が, 最大の全故障検出率を持つとは限らない．(証明終り)

補題 6.2 から, 測定ベクトル数に上限があるテスト生成では, 代表故障検出率を用いてテストを生成しても, 最大の全故障検出率をもつテストを生成できないことがわかる．図 6.2 で示したように, 代表故障検出率よりも全故障検出率の方が物理的欠陥の検出率により強い相関があるため, より多くの物理的欠陥を見つけるためには, 代表故障検出率よりも全故障検出率を最大にすることが望ましい．すなわち, 測定ベクトル数に上限がある場合は, 全故障検出率に従ってテストベクトル系列を求める必要がある．

6.3.2 重み付き故障リスト

全故障検出率を代表故障リストから計算するため, 本論文では各代表故障と等価な故障の数を記憶した故障リストを構成することを提案する．ここで, 各代表故障とそれと等価な故障の数のことを重みと呼び, 重みを記憶した故障リストを重みつき故障リストと呼ぶ．等価な故障がない故障の重みは１である．表 6.1 の場合は，故障 f_1と f_6に重み 2 と 4 を与えることで，代表故障から全故障検出率を計算できるようになる．

6.4 測定ベクトル選択手法

6.4.1 遷移ベクトルの評価値

本論文では, テストベクトル系列内のすべてのベクトルを遷移ベクトルと仮定した後, 故障シミュレーションを用いて測定ベクトルに変更するベクトルを, 与えられた測定ベクトルの上限の数だけ選ぶ手法を提案する．

最大の故障検出率をもつ測定ベクトルを選ぶための確実な方法は, すべての測定ベクトルの組合せに対して故障シミュレーションを行ない, それぞれの故障検出率を調べることである．しかしながら, 長さ M の系列から N 個の測定ベクトルを選ぶ選び方は${}_MC_N$通り存在するため, M が大きくなり N

が $M/2$ に近づくと, すべての組合せに対して故障シミュレーションを行なうことは現実的に不可能である.

実用的な計算時間で選択する手法として, 検出できる故障数が多いベクトルから順に選ぶ貪欲法 (greedy algorithm) がある [31, 45]. しかし貪欲法は, ある故障が複数のベクトルで検出されることを考慮していないため, 選択した測定ベクトルの多くが同一の故障を検出する可能性がある. そこで, 他のベクトルで検出される故障を考慮した遷移ベクトルの評価値を導入し, その評価値にしたがって測定ベクトルを選ぶ手法を提案する.

ベクトルの評価値を定義する前に, そこで使う用語を定義する. 遷移ベクトルを測定ベクトルに変えたことにより新たに検出される故障を, その非測定ベクトルの**潜在検出故障**と呼ぶ. また, テストベクトル系列内のすべての非測定ベクトルの潜在検出故障を調べた結果, 1 つの故障が潜在検出故障として数えられる回数を**潜在検出回数**と呼ぶ. このとき, 故障の検出困難度を次のように定義する.

定義 6.1 (検出困難度) テストベクトル系列 T を考える. テストベクトル系列 T に対する故障 f_iの潜在検出回数をそれぞれ a_iとする. このとき故障 f_iの検出困難度 dh_iは,

$$dh_i = \begin{cases} \frac{1}{a_i} & (a_i \leq M - N \text{ のとき}) \\ 0 & (\text{それ以外のとき}) \end{cases} \tag{6.2}$$

と定義する.

故障の検出困難度は，与えられたテストベクトル系列に対する故障の検出のしにくさを表している. つまり，検出困難度が低い故障は多くのベクトルで検出されることを意味する. ある測定ベクトルで検出できる故障が，後に選ばれる測定ベクトルで検出されることは，選択できる測定ベクトル数に上限があるため望ましくない. そのため，測定ベクトルを選択する時はなるべく困難度が高い故障を検出できるベクトルを選ぶことにする. よって，ベクトルの評価値は故障の検出困難度を用いて次のように定義する.

定義 6.2 (遷移ベクトルの評価値) テストベクトル系列 T を考える. 遷移ベクトル $v \in T$ の潜在検出故障集合を $F = \{f_1, f_2, \ldots f_n\}$ とする. このとき, ベクトル v の評価値は,

$$\sum_{i=1}^{n} dh_i \tag{6.3}$$

と定義する.

例として, テストベクトル系列 v_1,v_2,v_3,v_4と故障集合 $\{f_1, f_2, f_3, f_4, f_5, f_6\}$ を考え, このテストベクトル系列から 2 つの測定ベクトルを選択することを考える．最初すべてのベクトルは非測定ベクトルであると仮定する．表 6.2 は, 各非測定テストベクトルの潜在検出故障を示したものである．例えば, 遷移ベクトル v_1の潜在検出故障は, 3 つの故障 f_1, f_2, f_3である.

表 6.2: 故障表

	f_1	f_2	f_3	f_4	f_5	f_6
v_1	○	○	○			
v_2				○	○	○
v_3		○	○		○	○
v_4		○	○			○
潜在検出回数	1	3	3	1	2	3
検出困難度	1	0	0	1	$\frac{1}{2}$	0

貪欲法で測定ベクトルを選ぶ場合, 潜在検出故障数が多いベクトルから順に選ばれるため, 最初にベクトル v_3が選ばれ, 次に v_1 または v_2が選ばれる．仮に v_1と v_3が測定ベクトルに選ばれたとすると, 故障 f_4だけがこの測定ベクトルでは検出できず, 故障検出率は $4/5 = 80\%$となる.

次に提案手法であるベクトルの評価値を用いた選択方法について説明する．表 6.2 の 6 行目には, それぞれの故障の潜在検出回数を示しており, 各故障の検出困難度は 7 行目に示した値になる．故障 f_2, f_3, f_6は全部で 3 回検出されるため, 検出困難度は 0 である.

各ベクトルの評価値は, 潜在検出故障の検出困難度の和となる．表 6.3 は各故障の検出困難度から計算したそれぞれのベクトルの評価値を示す．4 つのベクトルの中で v_1と v_2がより高い評価値をもつため測定ベクトルとして選ばれる．v_1と v_2を選んだ場合はすべての故障が検出できるため, 100%の故障検出率が得られる.

表 6.3: ベクトルの評価値

テスト	評価値
v_1	1.0
v_2	1.5
v_3	0.5
v_4	0.0

6.4.2 選択アルゴリズム

1 つの遷移ベクトルを測定ベクトルに変えると, 他の遷移ベクトルの潜在検出故障数は変わるため, ベクトルの評価値も変化する．ベクトルの評価値を更新するためには, 故障シミュレーションをやり

直して, 各遷移ベクトルの潜在検出故障数を再計算しなければならない．提案する選択アルゴリズムでは, 故障シミュレーションの実行回数が与えられるものとし, 評価値の更新は測定ベクトルの選択中に等間隔に行なう．例えば, 15 個の測定ベクトルを選択するとして故障シミュレーションの実行回数が 3 回の場合, 5 個の測定ベクトルを選ぶたびに評価値の更新を行なうことになる．

操作 6.1 (測定ベクトル選択アルゴリズム)

step(1) 故障シミュレーションの実行回数 R と, 選択する測定ベクトル数 N を決める．

step(2) すべてのベクトルを遷移ベクトルとする．

step(3) 故障シミュレーションを行い, それぞれの非測定ベクトの潜在検出故障数を調べる．

step(4) 各故障の潜在検出回数を元に, 遷移ベクトルごとの評価値を決める．

step(5) 評価値の高い $\lfloor N/R \rfloor$ 個のベクトルを測定ベクトルとする．

step(6) 故障シミュレーション回数が R よりも小さいならば, step(3) に戻る．

step(7) 選択した測定ベクトル数が N に満たないならば, 評価値の高いベクトルを測定ベクトル数が N になるまで選ぶ．

step(8) N 個の測定ベクトルを返して, 終了する．

6.5　実験結果

ISCAS'89 のベンチマーク回路に対して, PC(Pentium Pro 150MHz) 上で実験を行なった．実験の際には, あらかじめ他のテスト生成手法により生成したテストベクトル系列を準備した．本報告の実験結果は, 疑似縮退故障 [39] を対象として HITEC[40] により生成したテストベクトル系列を用いた．

図 6.3 は s386 のベンチマーク回路に対して, 提案手法と貪欲法を比較結果を示したものである．用いたテストベクトル系列は 142 個のベクトルからなり, 故障シミュレーションの実行回数は 1 回であった．横軸は選択した測定ベクトル数, 縦軸は最大故障検出率に対する割合を表している．図から明らかなように, 提案手法は, 測定ベクトル数によらず常に, 貪欲法よりも多くの故障を検出する測定ベクトルを選ぶことができる．

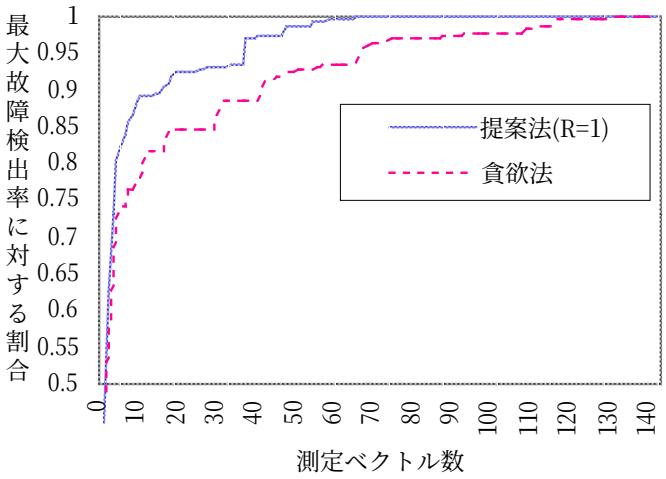


図 6.3: 提案法と貪欲法の比較 (s386)

表 6.4 は, それぞれのテストベクトル系列から 10 個と 20 個の測定ベクトルを選んだ場合の結果である．なお, 故障シミュレーションの繰り返し回数は 1 回である．回路名に続いて, テストベクトル系列に含まれるテストベクトル数, ゲート数, ゲート内短絡故障数, 最大故障検出率, 提案手法と貪欲法で選んだ 10 個の測定ベクトルの故障検出率, 提案手法と貪欲法で選んだ 20 個の測定ベクトルの故障検出率, そして提案手法を用いて測定ベクトルを選択するために必要な計算時間を示している．表 6.4 から, s298 と s820 の 2 つの場合を除く回路で, 提案手法が貪欲法よりも多くの故障を検出する測定ベクトルを選ぶことが可能なことがわかる.

表 6.4: 提案手法 (R=1) と貪欲法の比較

回路	テスト数	ゲート数	故障数	最大故障検出率 (%)	$N = 10$		$N = 20$		処理時間 (sec.)
					提案	貪欲	提案	貪欲	
s208	109	96	1242	79.71	73.51	72.46	77.38	76.97	0
s208.1	12	104	1365	34.51	34.43	34.43	—	—	0
s298	179	119	2103	90.58	70.52	59.91	*73.47*	74.99	2
s344	47	160	1973	90.52	86.72	83.53	90.07	86.47	0
s349	47	161	2006	90.23	86.49	84.05	89.98	88.98	0
s382	1276	158	2520	99.64	73.57	66.15	81.59	70.91	16
s386	142	159	2982	89.37	79.68	69.55	82.56	75.52	1
s400	1260	162	2646	99.62	71.35	68.86	82.35	74.11	16
s420	117	196	2516	68.96	65.82	53.82	65.86	63.99	1
s420.1	18	218	2905	31.91	31.57	31.26	—	—	0
s420_r	123	196	2516	69.00	63.91	53.82	65.30	64.47	1
s444	501	181	2877	96.21	70.91	67.33	72.54	68.96	7
s510	10	211	3335	15.71	15.71	15.71	—	—	0
s526	95	193	4022	76.45	71.03	63.77	75.46	70.84	2
s641	105	379	3978	89.06	85.37	82.76	87.73	85.70	2
s713	115	393	4428	87.20	81.82	80.26	86.04	83.13	2
s820	345	289	7098	99.90	*77.16*	77.37	84.22	81.39	20
s832	369	287	7260	99.90	78.14	76.45	85.07	80.52	21
s838	129	390	5032	62.52	60.21	52.70	60.21	58.92	3
s838.1	22	446	5958	30.87	30.51	30.46	30.87	30.87	0
s953	26	395	5658	41.76	41.27	41.04	41.73	41.60	0
s1196	135	529	7893	91.66	80.78	77.90	87.25	82.81	5
s1238	152	508	8301	90.89	81.60	74.22	87.04	81.38	6
s1423	66	657	8593	82.54	78.56	61.26	81.04	74.99	4
s1488	233	653	11396	89.72	76.33	66.51	82.00	71.50	96
s1494	299	647	11488	89.78	72.88	63.94	82.48	72.37	119
s5378	814	2779	31535	99.33	77.51	67.35	81.47	74.14	898
s9234	131	5597	56848	23.42	22.98	21.95	23.31	22.33	638
s9234.1	1031	5597	56848	94.75	55.86	52.96	57.54	53.69	5316
s13207	100	7951	80188	26.98	26.19	24.02	26.89	24.87	1161

図 6.4 は s386 に対して故障シミュレーションの回数を 1, 2, 3 回と変えて実験を行なった結果である．この結果から, 故障シミュレーションの実行回数を 1 回から 2 回にした場合はより高い故障検出

率をもつ測定ベクトルが選ばれたが，3 回にした場合はほとんど効果がみられなかった.

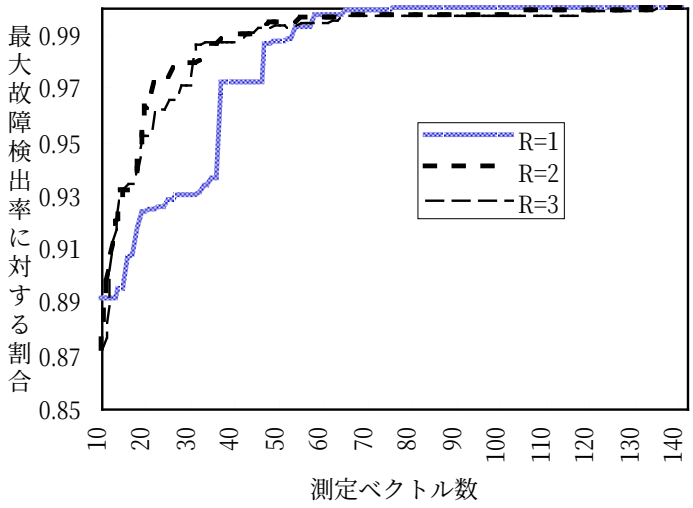


図 6.4: 故障検出率の繰り返しの効果 (s386)

6.6 あとがき

本章では, 測定ベクトル数に制限がある場合の IDDQ 測定ベクトル選択手法を提案した．測定ベクトル数の制限によりテスト時間を制御でき, 故障検出率とテスト時間を考慮したテストコストを考えることが可能となる．測定ベクトルの選択を行なう場合，代表故障を用いた場合と全故障を対象とした場合には得られるベクトルが異なるため，重みつき故障リストを提案することでこの問題を解決した．提案した選択手法は, 遷移ベクトルの評価値を用いることで, 計算時間がほぼ同じな貪欲法よりも, 多くの故障を検出することができる測定ベクトルを選ぶことができた.

第7章　結論

本論文では，テスト生成手法と，それを利用した回路簡単化手法について論じた．

第 3 章では, テスト生成手法である経路活性化法の含意操作の拡張手法である，静的学習と再帰学習について述べた．そして，静的学習の学習順序が含意辞書を得るための静的学習の能力を左右することを示した．最適な学習順序が分かれば，静的学習により抽出される含意関係の数が増えるため，含意辞書を用いた含意操作で一意に決まる信号値割当の数を増やしたり，回路簡単化における含意関係に基づく冗長付加の候補を増やすことができる．提案した 4 つの学習順序に対して計算機実験を行なった結果，回路の入力側から出力に向けて，レベルの低い信号線から処理を行なって行く学習順序 (FW) が最も静的学習の能力を高めることが分かった．

第 4 章では，部分回路除去に対する含意関係の不変性について考察し，静的学習のやり直しを行わない高速な冗長除去手法を提案した．一般的に含意関係は部分回路変換に対して無効になる場合があるため，部分回路変換の度に調べた含意関係をすべて破棄し，あらたに静的学習をやり直す必要があった．提案手法では，静的学習中に含意関係の不変性に関する情報を得ておき，その情報をもとに無効になる含意関係だけを除去することを行なう．静的学習中に行なう間接含意の有効性を判定するための情報を得るための時間はわずかなものであるため，全体として大きな処理時間の節約につながることになる．ベンチマーク回路に対する実験では，従来の静的学習を繰り返す冗長除去手法と比べて，計算時間に対して大きいもので約 80 倍もの差が見られた．本手法は，回路の部分的な変換に対する含意関係の不変性を調べることができるため，冗長部分回路の除去による回路簡単化だけでなく，第 5 章で示したように含意関係を基づく冗長回路付加および除去にも用いることができる．さらに，回路を簡単化しながら同時にテスト生成を行うことができるため，回路設計の流れを見直すことでより短時間で回路設計ができることになる．

第 5 章では，冗長付加と除去による論理回路簡単化手法における処理時間増加の問題を解決するために，含意操作を用いた高速な冗長指摘手法を提案した．テスト生成手法を行なうことによる冗長指摘では，回路内に存在する多くの縮退故障の検出不能性を調べる必要があるために，処理時間が長くなる．冗長付加後の回路で，新たに検出可能から検出不能にかわる縮退故障は，冗長付加部分に生じる信号値の矛盾が原因であるため，この信号値の矛盾から検出不能な縮退故障を調べることを行なえれば効率的に検出不能故障を見つけることができる．この考え方に準じて，提案手法では，冗長付加後に生じる信号値の矛盾を不当割当対として定義し，そこから含意操作を用いて冗長指摘を行な

う．また，それぞれの冗長付加に対して 4 種類の不当割当対が定義できるが，この不当割当対と検出不能故障の関係について考察を行なうことで，少なくとも 2 つの不当割当対は新たな検出不能故障の原因とならないことを示した．考える必要のある不当割当対が半分以下になったことで，冗長指摘の処理も軽減できることになる．ベンチマーク回路を用いた実験では，テスト生成を行なうことによる冗長指摘に比べて，回路簡単化の能力はそのままで処理時間が大幅に短縮できることが分かった．

第 6 章では, 測定ベクトル数に制限がある場合の IDDQ 測定ベクトル選択手法を提案した．測定ベクトル数の制限によりテスト時間を制御でき, 故障検出率とテスト時間を考慮したテストコストを考えることが可能となる．測定ベクトルの選択を行なう場合，代表故障を用いた場合と全故障を対象とした場合には得られるベクトルが異なるため，重みつき故障リストを提案することでこの問題を解決した．提案した選択手法は, 非測定ベクトルの評価値を用いることで, 計算時間がほぼ同じな貪欲法よりも, 多くの故障を検出することができる測定ベクトルを選ぶことができた．より高い故障検出率をもつテストベクトル系列を得るためには，与えられたベクトル系列から決められた数の測定ベクトルを選ぶのではなく，最初から測定ベクトル数を制限したテストベクトル系列を生成する方法が望まれる．

本論文で提案した手法は，テスト時間，テスト生成時間，回路簡単化時間といった，処理時間の短縮を目的としたものであった．これは，今後ますます回路規模が大きくなり設計・製造全体の総処理時間が増えることを考えると，これからも必要不可欠な研究テーマである．さらに，本論文ではテスト生成手法を回路簡単化手法に応用したが，これにより回路簡単化を効率化することができた．このような，回路設計の複数の分野にわたる研究のアプローチは，それまでの考え方の枠を超えて，新しい発想に繋がる可能性をもっていため，今後より一層重要になると考えられる．

謝辞

本研究は大阪大学大学院工学研究科応用物理学専攻において，樹下行三教授の御指導のもとで行ったものである．本研究を遂行するにあたり，終始御指導を賜り，また，有益な議論，および，御助言を頂きました樹下行三教授に心より感謝いたします．

本論文の作成に関し，詳細な御検討，貴重な御教示を頂きました大阪大学大学院工学研究科応用物理学専攻　伊東一良教授，豊田順一教授，小松雅治助教授，山本吉孝博士に深く感謝いたします．

同じく大阪大学大学院工学研究科応用物理学博士課程を修めるにあたり，大変お世話になりました，同専攻　増原宏教授，志水隆一教授，河田聡教授，笠井秀明教授，八木厚志教授，石井博昭教授，川上則雄教授，後藤誠一教授，岩崎裕教授，萩行正憲教授，および同研究科物質・生命工学専攻高井義造教授，一岡芳樹教授に深く感謝いたします．

大阪大学大学院工学研究科応用物理学専攻　板崎徳禎博士には本研究において大変有益な御助言を頂きました．板崎徳禎博士に深く感謝いたします．

本研究を通じて，九州工業大学情報工学部　梶原誠司助教授には研究を通して様々な御指導を頂きました．特に，研究の立ち上げ時および研究論文の推敲では大変有益な御助言を頂きました．梶原誠司助教授に深く感謝いたします．

University of Iowa の Sudhakar. M. Reddy 教授には，同大学留学中に大変熱心な御指導を頂きました．Sudhakar. M. Reddy 教授に心から感謝いたします．

愛媛大学工学部情報工学科　樋上喜信博士には研究を通しまして大変有益な御助言を頂きました．樋上喜信博士に深く感謝します．また，徳島大学工学部電気電子工学科　四柳浩之博士には，本研究において大変有益な御助言を頂き，とくに計算機実験を行う際には懇切な御指導をいただきました．四柳浩之博士に深く感謝いたします．

樹下・小松グループの諸氏には一方ならぬ御支援を頂きました．特に，前田敏行氏には，論文作成および計算機実験におきまして御助言，後討論をいただきました．ここに記して感謝いたします．

参考文献

[1] M. Abramovici, M. A. Breuer and A. D. Friedman, Digital Systems Testing and Testable Design, Computer Science Press, 1990.

[2] 樹下行三，藤原秀雄, デジタル回路の故障診断 (上), 工業図書, 1983.

[3] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," IEEE Trans. Computer, C-30, 3, pp. 215-222, Mar. 1981.

[4] H. Fujiwara and T. Shimono, "On the Acceleration of Test Generation Algorithms," IEEE Trans. Computer, C-32, pp. 1137-1144, Dec. 1983.

[5] M. H. Schulz, E. Trischler and T. M. Sarfert, "SOCRATES: A Highly Efficient Automatic Test Pattern Generation System," IEEE Trans. Computer-Aided Design, pp. 126-137, 1988.

[6] W. Kunz and D. K. Pradhan, "Recursive Learning: An attractive to the decision tree for test generation in digital circuits," Proc. Int'l Test Conf., pp. 816-825, 1992.

[7] S. T. Chakradhar, V. D. Agrawal and S. G. Rothweiler, "A Transitive Closure Algorithm for Test Generation," IEEE Trans. Computer-Aided Design, Vol. 12, No. 7, pp. 1015-1027, July 1993.

[8] 藤田昌宏, "多段論理回路合成技術の動向," 電子情報通信学会論文誌, Vol. J74-A, No. 2, pp. 152-161, Feb. 1991.

[9] R. K. Brayton, G. D. Hatchel and A. L. Sangiovanni-Vincentelli, Logic Minimization Algorithm for VLSI Synthesis, Kluwer Academic Publishers, 1984.

[10] R. K. Brayton, R. Rudell, A, Sangiovanni-Vincentelli and A. R. Wang, "MIS: A Multi-Level Logic Optimization System," IEEE Trans. Computer-Aided Design, Vol. CAD-6, No. 6, pp. 1062-1081, Nov. 1987.

[11] S. Muroga, Y. Kambayashi, H. C. Lai and J. N. Culliney, "The Transduction Method—Design of Logic Networks Based on Permissible Function," IEEE Trans. Computer, Vol. 38, No. 10, Oct. 1989.

[12] C. L. Berman and L. H. Trevillyan, "Global Flow Optimization in Automatic Logic Design," IEEE Trans. Computer-Aided Design, pp. 557-564, May 1991.

[13] M. Yuguchi, Y. Nakamura, K. Wakabayashi and T. Fujita, "Multi-Level Logic Minimization based on Multi-Signal Implications," Proc. Design Automation Conf., pp. 658-662, June 1995.

[14] D. Bryan, F. Brglez and R. Lisanke, "Redundancy Identification and Removal," Proc. MCNC Workshop on Logic Synthesis, May 1989.

[15] R. Jacoby, P. Moceyunas, H. Cho and G. Hachtel, "New ATPG Techniques for Logic Optimization," Proc. Int'l Conf. on CAD, pp. 548-551, Nov. 1989.

[16] S. Kajihara, S. Shiba, and K. Kinoshita, "Removal of Redundancy in Logic Circuits under Classification of Undetectable Faults," Proc. Int'l Sympo. on Fault-Tolerant Computer, pp. 263-270, July 1992.

[17] M. Abramovici and M. A. Iyer, "One-Pass Redundancy Identification and Removal," Proc. Int'l Test Conf., pp. 807-815, Sep. 1992.

[18] M. A. Iyer and M. Abramovici, "Low-Cost Redundancy Identification for Combinational Circuits," Proc. Int'l. Conf. on VLSI Design, pp. 315-318, 1994.

[19] J.-K. Zhao, E. M. Rudnick, and J. H. Patel, "Static Logic Implication with Application to Redundancy Identification," Proc. VLSI Test Sympo., pp. 288-293, 1997.

[20] K. T. Cheng and L. A. Entrrena, "Multi-Level Logic Optimization by Addition and Removal," Proc. European Conf. on Design Automation, pp. 373-377 (Feb. 1993).

[21] W. Kunz and P. R. Menon, "Multi-Level Logic Optimization by Implication Analysis," Proc. Int'l Conf. on CAD, pp. 6-13, 1994.

[22] S. C. Chang and M. Marek-Sadowska, "Perturb and Simplify: Multi-level Boolean Network Optimizer," Proc. Int'l Conf. on CAD, pp. 2-5, Nov. 1994.

[23] W. Kunz, "HANNIBAL: An Efficient Tool for Logic Verification Based on Recursive Learning," Proc. Int'l Conf. on Computer-Aided Design, pp. 538-543, 1993.

[24] S. Kajihara, K. Kinoshita and I. Pomeranz and S. M. Reddy, "A Method for Identifying Robust Dependent and Functionally Unsensitizable Paths," Proc. Int'l Conf. on VLSI Design, pp. 82-87, 1997.

[25] Randal E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," IEEE Trans. on Computers, Vol. C-35, No. 8, pp. 677-691, 1986.

[26] P. Goel and B. C. Rosales, "Test Generation and Dynamic Compaction of Tests," Proc. Test Conf., pp. 189-192, Oct. 1979.

[27] I. Pomeranz, L. N. Reddy and S. M. Reddy, "COMPACTEST: A method to generate compact test sets for combinational circuits," IEEE Trans. Computer-Aided Design, pp. 1040-1049, 1993.

[28] Y. Matsunaga, "MINT—An Extract Algorithm for Finding Minimum Test Set—,"IEICE Trans. Fundamentals, Vol. E76-A, No. 10, pp. 1652-1658, Oct. 1993.

[29] J. M. Soden, C. F. Hawkins, R. K. Gulati and W. Mao, "IDDQ Testing: A Review," Journal of Electronic Testing: Theory and Application, Vol. 3, pp. 291-303, 1992.

[30] W. Mao, R. K. Gulati, D. K. Goel and M. D. Ciletti, "QUIETEST: A Quiescent Current Testing Methodology for Detecting Leakage Faults," Proc. Int'l Conf. on Computer-Aided Design, pp.280-283, 1990.

[31] S. Chakravarty and P. J. Thadikaran, "Algorithms to Select IDDQ Measurement Points to Detect Bridging Faults," Journal of Electronic Testing: Theory and Application, Vol. 8, pp.275-285, 1996.

[32] T. Shinogi and T. Hayashi, "A Simple and Efficient Method for Generating Compact IDDQ Test Set for Bridging Faults," Proc. VLSI Test Sympo., pp.112-117,1998.

[33] R. S. Reddy, I. Pomeranz, S. M. Reddy and S. Kajihara, "Compact Test Generation for Bridging Faults under IDDQ testing," Proc. VLSI Test Sympo., pp.310-316, 1995.

[34] Y. Higami, K. K. Saluja and K. Kinoshita, "Observation Time Reduction for IDDQ Testing of Bridging Faults in Sequential Circuits," Proc. Asian Test Sympo., pp. 312-317, Dec. 1998.

[35] Y. Higami, K. K. Saluja and K. Kinoshita, "Efficient Technique for Reducing IDDQ Observation Time for Sequential Circuits," Proc. Int'l Conf. on VLSI Design, pp. 72-77, Jan. 1999.

[36] U. Mahlstedt, J. Alt and M. Heinitz, "CURRENT: A Test Generation System for IDDQ Testing ," Proc. VLSI Test Sympo., pp. 317-323, 1982.

[37] P. C. Maxwell and R. C. Aitken, "IDDQ Testing as a Component of a Test Suite: The Need for Several Fault Coverage Metrics," Journal of Electronic Testing: Theory and Application, Vol. 3, pp. 305-316, 1992.

[38] P. Nigh, D. Forlenza and F. Motika, “Application and Analysis of IDDQ Diagnostic Software,” Proc. Int’l Test Conf., pp. 319-327, 1997.

[39] S. T. Zachariah and S. Chakravarty, “A Comparative Study of Pseudo Stuck-at and Leakage Fault Model,” Proc. Int’l Conf. on VLSI Design, pp. 91-94, Jan. 1999.

[40] T. M. Niermann and J. H. Patel, “HITEC: A Test Generation Package for Sequential Circuits,” Proc. European Design Automation Conf., pp. 214-218, 1991.

[41] F. Brglez and H. Fujiwara, “A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran,” Proc. IEEE Int’l Sympo. on Circuits and Systems, pp. 663-698, June 1985.

[42] F. Brglez, D. Bryan, and K. Kozminski, “Combinational Profiles of Sequential Benchmark Circuits,” In Proc. IEEE Int’l Sympo. on Circuits and Systems, pp. 1929-1934 May 1989.

[43] 市原英行，梶原誠司，樹下行三, “部分回路除去に対する含意関係の不変性について,” 電子情報通信学会誌 D-I, Vol. J79-D-I, No. 12, pp. 1037-1045, Dec. 1996.

[44] H. Ichihara and K. Kinoshita, ”Logic Optimization: Redundancy Addition and Removal Using Implication Relations,” Trans. of IEICE Info. and Syst., Vol. E81-D, No. 7, pp. 724-730, July 1998.

[45] 市原英行, 梶原誠司, 樹下行三, “テスト数制限下でのテスト入力集合の選択手法について,” 電子情報通信学会誌 D-I, Vol. J82-D-I, No. 7, pp. 861-868, July 1999.

[46] H. Ichihara and K. Kinoshita, ”On Acceleration of Logic Circuit Optimization Using Implication Relations”, IEEE Proceedings of The Sixth Asian Test Symposium, pp. 222-227, Dec. 1997.

[47] H. Ichihara, S. Kajihara and K. Kinoshita, ”An Efficient Procedure for Obtaining Implication Relations and Its Application to Redundancy Identification” IEEE Proceedings of The Seventh Asian Test Symposium, pp. 58-63, Dec. 1998.

[48] H. Ichihara, S. Kajihara and K. Kinoshita, ”On test generation with a Limited Number of Tests,” IEEE Proceedings of Ninth Great Lakes Symposium on VLSI, pp. 12-15, March 1999.

[49] H. Ichihara and K. Kinoshita, ”New Redundancy Identification Method on Logic Optimization Using Implication Relation,” Technology Reports of Osaka University, Vol. 48, Nos. 2301-2318, pp.55-63, April 1998.

発表論文

論文誌

1. 市原英行，梶原誠司，樹下行三, “部分回路除去に対する含意関係の不変性について,” 電子情報通信学会誌 D-I, Vol. J79-D-I, No. 12, pp. 1037-1045, Dec. 1996.
2. H. Ichihara and K. Kinoshita, ”Logic Optimization: Redundancy Addition and Removal Using Implication Relations,” Trans. of IEICE Info. and Syst., Vol. E81-D, No. 7, pp. 724-730, July 1998.
3. 市原英行, 梶原誠司, 樹下行三, “テスト数制限下でのテスト入力集合の選択手法について,” 電子情報通信学会誌 D-I, Vol. J82-D-I, No. 7, pp. 861-868, July 1999.

学内報

1. H. Ichihara and K. Kinoshita, ”New Redundancy Identification Method on Logic Optimization Using Implication Relation,” Technology Reports of Osaka University, Vol. 48, Nos. 2301-2318, pp.55-63, April 1998.

国際会議

1. H. Ichihara and K. Kinoshita, ”On Acceleration of Logic Circuit Optimization Using Implication Relations”, IEEE Proceedings of The Sixth Asian Test Symposium, pp. 222-227, Dec. 1997.
2. H. Ichihara, S. Kajihara and K. Kinoshita, ”An Efficient Procedure for Obtaining Implication Relations and Its Application to Redundancy Identification” IEEE Proceedings of The Seventh Asian Test Symposium, pp. 58-63, Dec. 1998.
3. H. Ichihara, S. Kajihara and K. Kinoshita, ”On test generation with a Limited Number of Tests,”IEEE Proceedings of Ninth Great Lakes Symposium on VLSI, pp. 12-15, March 1999.
4. H. Ichihara, S. Kajihara and K. Kinoshita, ”On An Effective Selection of IDDQ Measurement Vectors for Sequential Circuits,” IEEE Proceedings of The Eighth Asian Test Symposium, Nov. 1999 (Accepted).
5. H. Ichihara, K, Kinoshita, I. Pomeranz and S. M. Reddy, ”Test Transformation to Improve Compaction for Statisitical Encoding,” IEEE Proceedings of The Thirteenth International Conference on VLSI Design, Jan. 2000(Accepted).

研究会

1. 市原英行, 梶原誠司, 樹下行三, “冗長除去における含意関係の不変性について,” 第 33 回 FTC 研究会, July 1995.
2. 市原英行, 樹下行三, “含意関係を用いた論理回路簡単化手法の高速化について,” 第 35 回 FTC 研究会, July 1996.
3. 市原英行, 樹下行三, “含意を用いた多段論理回路簡単化手法の高速化に関する研究,” 電子情報通信学会, FTS 研究会, Feb. 1997.
4. 市原英行, 梶原誠司, 樹下行三, “静的学習で獲得できる間接含意数の最大化について,” 第 37 回 FTC 研究会, July 1997.
5. 市原英行, 梶原誠司, 樹下行三, “IDDQ テストにおける故障検出率の最大化について,” 第 39 回 FTC 研究会, July 1998.