設計自動化 62－28
（1992. 5.28）

共有二分決定グラフを用いた順序回路のテスト生成法

崔 滉鎔　小原 隆司　石浦 菜岐佐　白川 功

大阪大学工学部情報システム工学科

あらまし　本稿では、共有二分決定グラフを用いた同期式順序回路に対するテスト生成アルゴリズムを提案する。本手法は、正常-故障状態対を非明示的に数え挙げ、リセット状態から時間軸の前方向へ幅優先探索でテスト系列の探索を行なうアルゴリズムである。これは検査入力が存在すれば、必ずこれを求めることができる完全なアルゴリズムであり、特にテスト困難な故障と冗長な故障の検出に有効である。この手法に基づくテスト生成システムを実現し、ISCAS89 ベンチマーク回路について実験を行なった結果、中規模の回路に対しては、現実的な時間内で検出率100%のテスト生成ができた。

Test Generation for Sequential Circuits Using Shared Binary Decision Diagrams

Hoyong CHOI　Takashi KOHARA　Nagisa ISHIURA　Isao SHIRAKAWA

Department of Information System Engineering, Faculty of Engineering, OSAKA University

Abstract　This paper presents a complete test pattern generation algorithm for synchronous sequential circuits based on Boolean function manipulation using shared binary decision diagrams. This algorithm enumerates all the fault-free–faulty state pairs implicitly and completely, and generates test patterns by breadth-first traversal from a reset state. It is a complete test generation algorithm for sequential circuits, that is, a fault can be surely detected if a test pattern sequence for the fault exists. Experimental results on ISCAS sequential benchmark circuits demonstrate that this algorithm can generate test pattern for all faults in reasonable time for medium-sized circuits and is very efficient for hard-to-detect and redundant faults.

1 はじめに

論理回路の故障を検査するテスト系列の生成は、回路の信頼性の確保のために必須であり、近年の半導体技術の発達に伴う論理回路の大規模化、複雑化によって、テスト生成手法の研究の重要性はますます増大している。組合せ回路のテスト生成アルゴリズムは、Dアルゴリズム[1]、PODEM[2]、FAN[3]、SOCRATES[4]など数多く提案されている。これらのアルゴリズムによれば、大部分の故障に対して現実的な時間で検査入力を生成することが可能となり、実用的なテスト生成システムが作成されている。しかし、順序回路のテスト生成は、以下の理由で組合せ回路のそれよりも難しい。

- 回路の内部状態の設定及び観測が困難である。
- 一つの故障を検出するために、長い検査系列が必要となる場合が多い。このような場合には、検査系列生成の処理の中で回路の内部状態を保持しなければならないため、膨大な記憶量が必要となる。
- 元の回路では単一故障であっても、時間展開した組合せ回路上では、これを多重故障とみなして取り扱わなければならない。

この問題を解決するために、スキャンパス設計方式[5]などのテスト容易化設計により、順序回路のテストの生成の問題を、組合せ回路のテスト生成の問題に帰着する方法がある。しかし、テスト容易化のためのハードウェアの増加、回路性能の低下、テスト系列の増大などの問題がある。したがって、順序回路に対して直接検査入力を生成する技術に対する要望は強く、そのためのアルゴリズムが研究・提案されてきた[6, 7, 8, 9, 10]。これまでに提案された方法は、解の探索をどのように行なうかにより、大きく二つに分けられる。一つは、経路探索に基づいた方法[6, 7, 8]で、各々のゲートの信号線に値を与えて、解の空間を経路探索により行なうものである。しかし、順序的冗長な故障とテスト困難な故障に対しては、非常に広い空間の探索を行なわなければならない。したがって、テストの探索に膨大な時間と記憶容量を必要とし、そのために完全な探索が行なえないことがある。もう一つの手法[9, 10]は、状態遷移グラフに基づいた方法であるが、正常回路の状態遷移図のみで状態正当化と状態区別化を行なっているため、テストの正当性をテスト生成後故障シミュレーションで確かめなければならず、また、テスト生成が不可能な場合もある。

本稿では、正常–故障状態対の非明示的な数え挙げ(implicit state enumeration)手法[12]によって、リセット状態から時間軸の前方向へ幅優先探索でテスト系列の探索を行なうテスト生成アルゴリズムを提案する。本手法は、検査入力が存在すれば、必ずこれを求めることができる完全なアルゴリズムであり、また、特にテスト困難な故障と冗長な故障の検出に有効である。検査系列生成システムの実現の際には、論理関数及び集合の表現に共有二分決定グラフ[11]を使用し、システムの高速化と使用記憶量の削減を図った。この手法に基づくテスト生成システムを実現し、ISCAS89ベンチマーク回路について実験を行なった結果、回路規模が30フリップフロップ以下、600ゲート程度の中規模な回路に対しては、現実的な時間内で検出率100%のテスト生成ができた。

以下、2章では、まず準備として記号、用語の定義について述べる。3章では、論理関数処理に基づいた検査系列生成法であるSTARアルゴリズムについて述べる。さらに4章でこのアルゴリズムの実現法について述べ、5章ではISCAS89ベンチマーク回路についての実験結果を示す。6章ではまとめと今後の課題について述べる。

2 準備

2.1 順序回路

本稿で取り扱う順序回路は単相クロックによる完全同期式順序回路であり、図1に示すように、論理ゲートと結線により構成される組合せ論理部(combinational logic block: CL)と、フリップフロップ(FF)により構成される状態記憶部(state register block: SR)から成る。この回路は、外部入力をn本、外部出力をm本、FFをl個持つものとする。また、この回路はリセット状態という特別な状態を有しており、リセット信号によってこの状態に設定できるものと仮定する。故障モデルはゲートレベルでの単一縮退故障モデルを採用する。故障は組合せ論理部内の各信号線と現在、次状態線上でのみ生じるものと仮定し、FF内での故障は考えない。また、故障回路においてもリセット(リセット状態への初期設定)は可能であると仮定する。同期式順序回路は図2に示すように、時間的に展開した反復回路モデルで表現される。$x(t)$、$y(t)$、$Y(t)$、$z(t)$はそれぞれ時刻tにおける外部入力、現状態、次状態、外部出力を表す。

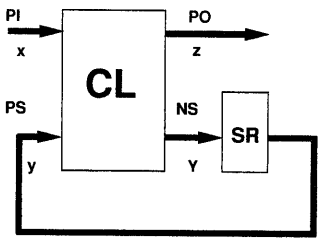

図1: 同期式順序回路

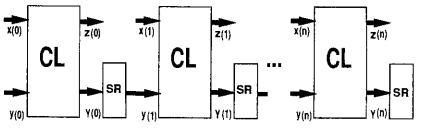

図2: 順序回路の時間展開

2.2 記法

n本の外部入力に対応する論理変数(外部入力変数)と、l個のFFの持つ状態に対応する論理変数(現在状態変数)をそれぞれ$x_i\ (i = 1, 2, \cdots, n)$、$y_j\ (j = 1, 2, \cdots, l)$と表す。そして、これらをそれぞれ一まとめにしてベクトル

表記し、外部入力変数ベクトル x 、状態変数ベクトル y と呼ぶことにする。すなわち、

$x = (x_1, x_2, \cdots, x_n)$,
$y = (y_1, y_2, \cdots, y_l)$

である。リセット状態を $r = (r_1, r_2, \cdots, r_l)$ で表す。

各外部出力 z_i と次状態 Y_j の論理関数を

$z_i = f_i = f_i(x, y); \quad i = 1, 2, \cdots, m$
$Y_j = g_j = g_j(x, y); \quad j = 1, 2, \cdots, l$

とし、これらも同じく

$f = (f_1, f_2, \cdots, f_m)$,
$g = (g_1, g_2, \cdots, g_l)$

とベクトル表記することにする。なお、以下で、故障回路での変数および関数は、正常回路での記号にプライムを付けて、y' 、f_i' のようにして表す。但し、状態は常に正常回路でのものと故障回路でのものとを対として扱う。これを正常–故障状態対 (あるいは単に状態対) と呼び、$\theta = (y, y')$ の様に表記する。尚、y 、$y(i)$ などの表記が混在するが、一般に前者は論理変数を表し、後者は時刻 i でのその変数の値を表す。但し、時刻が明らかな場合は、後者の意味で y と表記することもある。

外部出力の故障差関数 D_F と、次状態の故障差関数 D_G を以下のように定義する。

$D_F(x,(y,y')) = \bigvee_{i=1}^{m}[\ f_i(x,y) \oplus f_i'(x,y')\ ]$,
$D_G(x,(y,y')) = \bigvee_{i=1}^{l}[\ g_i(x,y) \oplus g_i'(x,y')\ ]$ 。

これはすなわち、故障の影響が外部出力に現れる時 (D_F) 、あるいは次状態に現れた時 (D_G) に真となる関数である。

状態対の集合 Θ に対する状態対遷移関数 $T(\Theta)$ を

$T(\Theta) = \{\ (u,u') \mid \exists x\ s.t.\ u = g(x,v),$
$\qquad u' = g'(x,v'),\ (v,v') \in \Theta\ \}$

と定義する。これは、集合 Θ に含まれる任意の状態対から遷移可能な全ての状態対の集合を与える関数である。

時刻 t における到達可能状態対集合 Ψ_t と初到達可能状態対集合 Φ_t を

$\Psi_0 = \{\ (r,r)\ \},\ \Psi_t = \Psi_{t-1} \cup T(\Psi_{t-1})\ \ (t > 0),$
$\Phi_0 = \Psi_0,\ \Phi_t = \Psi_t \cap \overline{\Psi}_{t-1}\ \ (t > 0)$

と定義する。つまり、前者は時刻 t 迄に到達可能な状態対の集合であり、後者は時刻 t で初めて到達可能な状態対の集合である。この両者には、定義から、

$\Psi_t = \Psi_{t-1} \cup T(\Phi_{t-1}) = \Psi_{t-1} \cup \Phi_t,$
$\Phi_t = T(\Phi_{t-1}) \cap \overline{\Psi_{t-1}}$

の関係が成立する。また、状態対集合の記号を小文字で書いたものは、特に断らない限り、その集合の要素の一つを表すものとする ($\phi_t \in \Phi_t$ など)。

次に特徴関数を定義する。集合 E とその部分集合 $P(P \subseteq E)$ を考える。この時、P の特徴関数 χ_P は、

$$\chi_P(x) = \begin{cases} 1 & \text{if } x \in P, \\ 0 & \text{otherwise} \end{cases}$$

と定義される。これは、集合を2値関数で表現するものである。特に E の要素が p ビットで2値符号化されている場合、x_p は p 変数論理関数となる。

$x = (x_1, x_2, \cdots, x_k)$ を変数とする論理関数 $f(x)$ を考える。この時、f の x によるスムージングは $S_x f$ と表され、

$S_x f = S_{x_1} S_{x_2} \cdots S_{x_k} f,$
$S_{x_i} f = f_{x_i} + f_{\overline{x_i}};\ i = 1, 2, \cdots, k$

と定義される。ここで、f_a は f の リテラル a による cofactor である。即ち、f_{x_i}、$f_{\overline{x_i}}$ は f の x_iにそれぞれ 1、0 を代入して得られる関数である。スムージングとはつまり、関数から変数を消去する演算である。

論理関数 f を $f : B^p \to B$ とし、c を B^p のある部分集合の特徴関数とする。論理関数 f の c による generalized cofactor [12] を f_c と表記する。f_c は、直感的には c の表す集合に含まれる入力については f と同じ値を持ち、c の表す集合に含まれない入力はドント・ケアとみなして任意の値を割り当てることにより、表現を f より小さくすることをねらった関数である。

2.3 順序回路のテスト生成

順序回路のテスト生成とは、順序回路 C と故障 α が与えられた時に、回路の外部入力に検査系列を与えてその故障が回路に存在しているかを外部出力のみで観測できるか否かを判断し、さらにそれが可能であった場合には、検査系列を求めることである。観測が不可能である場合、故障 α は冗長であるという。冗長故障には組合せ的冗長 (combinationally redundant) 故障、順序的冗長 (sequentially redundant) 故障という2つのクラスがある。組合せ冗長故障とは、いかなる内部状態と入力の組合せによっても故障の影響を外部出力にも次状態にも伝搬できないものであり、順序的冗長故障とは故障を外部出力に伝搬するのに必要な内部状態対に回路を設定できない、つまりリセット状態から到達できる状態対の中にはそういう状態対が存在しないものである。これを式で表すと、

組合せ的冗長
$\forall(x,y) \mid D_F(x,(y,y)) = 0\ \wedge\ D_G(x,(y,y)) = 0$
順序的冗長
$\forall\{x(k)\}_{k=0}^{N} \mid D_F(x(i),(y(i),y'(i))) = 0; i = 0, 1, \cdots, N$

但し $(y(0), y'(0)) = (r,r),\ \Phi_N \neq \{\},\ \Phi_{N+1} = \{\}$

となる。

3 STAR アルゴリズム

本手法 STAR (**S**equential **T**est p**A**ttern gene**R**ation using shared binary decision diagram) は、正常–故障状態対の非明示的な数え挙げ手法 [12] によってリセット状態から時間軸の前方向へ幅優先探索でテスト系列の探索を行なうアルゴリズムである。正常回路と故障回路の直積回路 (product machine) を考慮し、テスト系列をリセット状態から時間軸の前方向へ探索する。この際、状態対遷移関数 [12] により、各時刻において到達可能な正常–故障の状態対を非明示的にかつ完全に数え挙げ、それに対してそれに対して外部入力の組合せを探索する。全ての到達可能な状態対に対して探索を行なうので、検査入力が存在すれば必ずそれを発見できる。つまり、このアルゴリズムは完全である。また、特にテスト困難故障と冗長な故障の検出に有効である。なお探索を幅優先で行なうので、求まる系列は、その故障に対するものとしては必ず最短のものになる。また計算量の削減のため、次の時刻へ移る際に、次状態対として、遷移可能な状態対全

```
main()
{
  if (there is no (x,y) s.t. D_F(x,(y,y)) ∨ D_G(x,(y,y)) = 1)
     Combinationally Redundant ;
  else {
    Ψ_0 = {( r,r )}; Φ_0 = Ψ_0 ;
    φ = SeqTestGen ( 0, Φ_0, Ψ_0) ;
    if ( φ = null ) Sequentially Redundant ;
    else output a TP sequence v(0), v(1), ···, v(k) ;
  }
}
SeqTestGen ( t, Φ_t, Ψ_t)
{
  if ( (a,θ) exists s.t. D_F(a,θ) = 1 for θ ∈ Φ_t) {
    v(t) = a ;
    return ( θ ) ;
  } else {
    Φ_{t+1} = T(Φ_t) ∩ ¬Ψ_t;
    if ( Φ_{t+1} ≠ empty) {
      Ψ_{t+1} = Ψ_t ∪ Φ_{t+1} ;
      φ_{t+1} = SeqTestGen (t+1, Φ_{t+1}, Ψ_{t+1}) ;
      if ( φ_{t+1} ≠ null) {
        get (a,θ) that places from θ ∈ Φ_t to φ_{t+1} ;
        v(t) = a ;
        return ( θ );
      }
    }
    return ( null ) ;
  }
}
```

図 3: Algorithm

てではなく、その時刻で初めて到達可能となるもの (初到達可能状態対) だけを選ぶ様にしている。

アルゴリズムを擬似コード表現したものを図3に示し、以下その動作について述べる。

main() では、まず最初に、外部出力の故障差関数または次状態出力の故障差関数を真とする入力 x と状態 y の組合せが存在するかどうか調べる。そのような (x,y) が存在しなければ、対象としている故障 (α) は組合せ的冗長である。(x,y) が存在する時には、時刻0での到達可能状態対集合Ψ_0 と初到達可能状態対集合 Φ_0 を初期化 (その要素は共にリセット状態対のみ) し、**SeqTestGen** を呼び出す。**SeqTestGen** は時刻 t と初到達可能状態対集合 Φ_t と到達可能状態対集合 Ψ_t を引数として、Φ_t および Φ_t から到着できる状態対で、故障の影響を外部出力へ伝えるものがあるかどうか調べ、もしあればその状態対へ到達可能な状態対 ϕ を返すものである。ϕ が空 (null) であれば、故障 α は順序的冗長である。そうでなければ検査系列 $v(0), v(1), \cdots, v(k)$ が得られており、それを出力して終了する。

SeqTestGen() は、まずその時刻 t において与えられた初到達可能状態対集合 Φ_t に対して外部出力の故障差関数 D_F を真とする入力と状態対の組が存在するか調べ、存在すれば、その入力をその時刻における検査入力 $v(t)$ とし、その状態対 (ϕ_t) を返り値として戻る。存在しなければ、Φ_t と Ψ_t から Φ_{t+1} とΨ_{t+1} を求める。ここで Φ_{t+1} が空集合となった場合には、故障を外部出力に伝搬するのに必要な内部状態対に回路を設定できないので、その故障は順序的に冗長であるとして、検査入力なしで戻る。そうでなければ時刻を1だけ増やし、Φ_{t+1} を初到達可能状態対集合、Ψ_{t+1} を到達可能状態対集合として再帰的に **SeqTestGen** を呼び出す。そして次の時刻で状態対 ϕ_{t+1} (と検査入力) を得て戻ってきたら、Φ_t に属する状態対の中から ϕ_{t+1} に遷移することのできる状態対の一つ ϕ_t とその時の入力 $v(t)$ (時刻 t での検査入力) を求め、ϕ_t を返す。

4 STAR アルゴリズムの実現法

4.1 論理関数処理による STAR の実現

STAR アルゴリズムは集合演算や論理関数演算を数多く必要とする。これを如何にして現実的な時間と記憶量で行なえるようにするかが、成功の鍵となる。我々は、これらの演算を共有二分決定グラフ (Shared Binary Decision Diagrams; SBDD's) [11] に基づく高速な論理関数処理によって実現する。SBDD は、二分決定グラフ (Binary Decision Diagrams; BDD's)[13] から、さらに複数の二分決定グラフ間で共通な部分グラフの共有化を図ったものであり、複数の関数を、少ない記憶量で同時に表せる。SBDD で論理関数を表現した場合、多くの部分を正常時と故障時の論理関数の間で共有できる可能性があり、記憶量および計算時間の削減が期待できる。

4.2 状態対の集合の表現と状態対遷移関数の計算法

本手法では正常-故障状態対集合 Θ と到達可能状態対集合 Ψ_t と初到達可能状態対集合 Φ_t などの全ての集合を、特徴関数を用いてSBDD で表現する。以下では混乱のない限り集合といえば、その集合の特徴関数を意味するものとする。

現正常-故障状態対から次正常-故障状態対を求める状態対遷移関数 $T(\Theta)$ は、generalized cofactor を利用して計算する。これは、[12] の手法と同様に正常回路と故障回路の直積回路に対して、現在の状態対と次状態対の関係を表す transition relation を計算し、これから現状態対を表す変数を消去して次状態対を計算するものである。Θ に含まれる状態対の遷移先だけを正確に計算すれば良いため、generalized cofactor を用いて表現の縮約を図る [12]。論理式で表せば、transition relation とは、(y,y') から入力 x によって (Y,Y') に遷移できるような (x,y,y',Y,Y') の集合を論理関数として表現したものであり、

$$\{\bigwedge_{i=1}^{l}(Y_i \equiv (g_i(x,(y,y')))) \cdot (Y_i' \equiv (g_i'(x,(y,y'))))\}$$

と表せる。これと Θ との論理積をとり、x,y,y' をスムージングすれば、Θ から遷移できる状態対の集合 $\Gamma(\Theta)$ が得られる。しかし、上式の g_i, g_i' のかわりに、それらの Θ による generalized cofactor $(g_i)_\Theta, (g_i')_\Theta$ を用いれば、Θ

との論理積演算なしに同じ結果が得られる。すなわち、

$$\Gamma(\Theta) = S_x S_y S_{y'}[\{\wedge_{i=1}^{l}(Y_i \equiv (g_i(x,(y,y')))_\Theta)\} \cdot \{\wedge_{i=1}^{l}(Y_i' \equiv (g_i'(x,(y,y')))_\Theta)\}] \circ$$

である。generalized cofactor の性質から後者の演算法では、前者に比べて中間計算で必要な SBDD のノード数が大幅に減少すると考えられる。また、次のようにして (Y,Y') を (y,y') と変換することにより、状態対遷移関数 $T(\Theta)$ が得られる。

$$T(\Theta) = S_Y S_{Y'}[\Gamma(\Theta)\cdot\{\wedge_{i=1}^{l}(y_i \equiv Y_i)\}\cdot\{\wedge_{i=1}^{l}(y_i' \equiv Y_i')\}].$$

これは、変数シフトエッジを用いた BDD[11] では簡単に計算できる。

4.3 故障差関数

故障の組合せ的冗長性の判定の際に必要な故障差関数 $D_F(x,(y,y))$ と $D_G(x,(y,y))$ は記号単一故障伝搬法 (Symbolic Single Fault Propagation method: SSFP)[14] を利用して求める。つまり、最初に正常回路の外部出力、次状態出力をはじめとする全信号線の正常時の論理関数を、外部入力変数 x と現状態変数 y の論理関数として求める。次に全信号線の故障時の論理関数を求めるのであるが、この際故障点から外部出力と次状態出力に向かって故障の影響を計算していく。途中で故障時の論理関数が正常時の論理関数と等しくなった経路は処理を打ち切る。正常時の論理関数と故障時の論理関数の排他的論理和から故障差関数 D_F, D_g を求める。なお、正常時の論理関数は不変なので一度求めるだけで良い。また検査可能性の判定ができるとすぐに処理を終了するため、すべての信号線の論理関数を求める必要はない。特に冗長な故障の場合は故障の影響がすぐに消えることが多く、効率が良い。

一方、現状態対集合で故障の影響が伝搬するかどうかを探索する故障差関数 $D_F(a,\theta)$ は、$f'(x,y)$ で現状態変数 y を y' に変換して、$f(x,y)$ と $f'(x,y')$ から求める。

4.4 変数の順序づけ

論理関数を表す SBDD の大きさを左右するのは、変数の順序付けである。ここでは、動的重み付け法 [11] に基づく順序付け法を用いる。まず、組合せ回路部に動的重み付け法を適用し、外部入力変数、状態変数を区別せず $x_1, x_2, \cdots, x_n, y_1, y_2, \cdots, y_l$ を順序付ける。次に、外部入力変数内、状態変数内での順序関係を保存しながら、全ての外部入力変数が全ての状態変数より上位に来るように順序を付け直す。これは、外部入力変数が状態変数より大きな影響を出力に及ぼすと考えられるからである。故障回路の状態変数である y_i' は y_i のすぐ下位に、また、遷移演算の際に必要となる次状態変数 Y_j 、Y_k' は現状態変数の順序を守りながら現状態変数の下位に来るように順序を付ける。例えば、動的重み付け法によって、高位側から順に

$y_2,\ x_1,\ x_3,\ y_1,\ x_2$

という順序が得られた場合、本手法での順序は

表 1: ISCAS89 ベンチマーク回路

Circuit	*#PI*	*#PO*	*#DFF*	*#Gates*	*#Faults*
s27	4	1	3	10	32
s208	11	2	8	96	215
s298	3	6	14	119	308
s382	3	6	21	158	399
s386	7	7	6	159	384
s400	3	6	21	162	424
s444	3	6	21	181	474
s510	19	7	6	211	564
s526n	3	6	21	194	553
s526	3	6	21	193	555
s641	35	24	19	379	467
s713	35	23	19	393	581
s820	18	19	5	289	850
s832	18	19	5	287	870
s953	16	23	29	395	1079
s1196	14	14	18	529	1242
s1238	14	14	18	508	1355
s1488	8	19	6	653	1486
s1494	8	19	6	647	1506

$x_1,\ x_3,\ x_2,\ y_2,\ y_2',\ y_1,\ y_1',\ Y_2,\ Y_2',\ Y_1,\ Y_1'$

となる。

5 実験結果

前章で述べた順序回路向け検査系列生成手法を、C 言語を用いて SPARCstation2 上に実現し、幾つかの ISCAS89 ベンチマーク回路 [15] に対して実験を行なった。表 1に実験を行なった ISCAS89 ベンチマーク回路の諸元を示す。各欄は、外部入力数 $(\#PI)$、外部出力数 $(\#PO)$、フリップフロップ数 $(\#DFF)$、ゲート数 $(\#Gates)$、故障数 $(\#Faults)$ を表す。ISCAS89 ベンチマーク回路にはリセット状態が明記されていないため、全ての FF が 0 の状態をリセット状態と仮定した。回路中の全ての故障について STAR を実行したものであり、故障シミュレーションは行なっていない。また実行時、SBDD の最大ノード数は $2^{20} \simeq 10^6$ とした。実験結果を表 2に示す。$\#CRF$ は組合せ的冗長な故障の数、$\#SRF$ は順序的冗長な故障の数、$\#N_{max}$ はテスト生成時に作成された BDD の最大のノード数、L_{max} は最大のテスト系列の長さ、$Time$ はテスト生成時間、FC は故障検出率を表す。但し、故障検出率とは、$(\#CRF + \#SRF + \text{検出故障数})/\#Faults \times 100(\%)$ である。

回路規模が 30 フリップフロップ以下、600 ゲート程度の回路に対しては、現実的な時間内で検出率 100%のテスト生成ができた。組合せ的冗長な故障と順序的冗長な故障の判定もできた。結果で示してない回路は SBDD のノード数が溢れて 100% の検出率が得られなかったものである。

6 結論

本稿では、論理関数処理に基づいた完全な順序検査生成アルゴリズム STAR を提案した。検査系列生成システムの実現の際には、論理関数及び集合の表現に共有二分決定グラフを使用し、実行の高速化と使用記憶量の削減を図った。このアルゴリズムは、テスト生成の完全性を最優先に考えたものであったが、十分現実的な時間で検

表 2: 実験結果

Circuit	*#CRF*	*#SRF*	N_{max}	L_{max}	*Time**	*FC*
s27	0	0	36	2	0.1s	100%
s208	0	65	48	18	15.1s	100%
s298	0	35	2799	20	22.1s	100%
s382	0	20	376481	133	0.92h	100%
s386	0	70	85	9	4.5s	100%
s400	8	21	435935	133	0.96h	100%
s444	14	21	254079	133	1.13h	100%
s510	0	0	40	48	1.89h	100%
s526n	0	87	273698	133	1.45h	100%
s526	1	88	273532	133	1.33h	100%
s641	0	59	19758	5	0.87h	100%
s713	38	63	19760	5	0.99h	100%
s820	0	35	167	12	58.9s	100%
s832	14	37	167	12	59.0s	100%
s953	0	10	59100	12	0.37h	100%
s1196	0	3	102408	3	896s	100%
s1238	69	3	115401	3	918s	100%
s1488	0	40	114	23	76.5s	100%
s1494	12	39	114	23	77.8s	100%

* s: sec., h: hours

査系列の生成が行なえ、その有効性を実証することができた。

今後の課題としては、より大規模な回路への対応と、高速化の問題がある。その解決のための方策として、探索を完全に幅優先で行なわずに、状態対集合を複数に分割しつつ、その結果得られた部分集合に対して深さ優先探索を行なう方法が考えられる。あるいは、更に記憶効率の良い（共有度の高い）SBDD 上での変数の順序付け法を求めることも重要な課題である。

謝辞

本研究を行なうにあたり、有益な御助言を頂いた松下電器産業（株）の本原章氏と SBDD に基づく論理関数処理プログラムを提供して頂いた湊真一氏（現在 NTT(株)）及び京都大学矢島研究室の諸氏と大阪大学白川研究室の諸氏に心から感謝致します。

参考文献

[1] J. P. Roth, "Diagnosis of Automata Failures:A Calculus and a Method," *IBM J. of Research and Developement*, Vol. 10, pp. 278–291 , July 1966.

[2] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," *IEEE Trans. Computers*, Vol. C-30, pp. 215–222, March 1981.

[3] H. Fujiwara and T. Shimono, "On the Acceleration of Test Generation Algorithm," *IEEE Trans. Computers*, Vol. C-32, pp. 1137–1144, Dec. 1983.

[4] M. H. Schultz, E. Trischler, and T. M. Sarfert, "SOCRATES: A Highly Efficient Automatic Test Pattern Generation System," *IEEE Trans. Computer-Aided Design*, Vol. CAD-7, pp. 126–137, Jan. 1988.

[5] E. B. Eichelberger and T. W. William, "A Logic Design Structure for LSI Testability," *Proc. 14th. Design Automation Conference*, pp. 462–468, June 1977.

[6] S. Mallela and S. Wu, "A Sequential Circuit Test Generation System," *Proc. Int. Test Conf.*, pp. 57–61, Nov. 1985.

[7] R. Marlett, "An Effective Test Generation System for Seqential Circuits," *Proc. 23rd Design Automation Conference*, pp. 250–256, June 1986.

[8] W. T. Cheng and T. J. Chakraborty, "Gentest: An Automatic Test-Generation System for Sequential Circuits," *IEEE Computer*, Vol. 22, pp. 43–49, April 1989.

[9] H-K. T. Ma, S. Devadas, A. R. Newton, and A. Sangiovanni-Vincentelli, "Test Generation for Sequential Circuits," *IEEE Trans. Computer-Aided Design*. Vol. CAD-7, pp. 1081–1093, Oct. 1988.

[10] A. Ghosh, S. Devadas, and A. R. Newton, "Test Generation and Verification for Highly Sequential Circuits," *IEEE Trans. Computer-Aided Design*. Vol. CAD-10, pp. 652–667 , May 1991.

[11] 湊, 石浦, 矢島, "論理関数の共有二分決定グラフによる表現とその効率的処理手法," 情報処理学会論文誌, Vol. 32, pp. 77–85, Jan. 1991.

[12] H. J. Touati, H. Savoj and B. Lin,et al., "Implicit State Enumeration of Finite State Machines using BDD's," *IEEE Intenational Conf. on Computer-Aided Design*, pp. 130– 133,1990.

[13] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Trans. Computer*. Vol. 35, pp. 677–691, Aug. 1986.

[14] 井置, 石浦, 矢島, "共有二分決定図を用いた組合せ論理回路のテスト生成," 情報処理学会第 38 会全国大会, 2S-5, pp. 1317–1318,1989.

[15] F. Brglez, D. Bryan, and K. Kozminski, "Combination Profiles of Sequential Benchmark circuits," *Proc. Int. Symp. on Circuits and Systems*, June 1989.