設計自動化 77--8
(1995. 10. 19)

関数処理に基づく含意操作と組合せ回路の等価性検証について

松永 裕介

(株) 富士通研究所

〒211 川崎市中原区上小田中 1015

044-754-2663

yusuke@flab.fujitsu.co.jp

あらまし 本稿では、組合せ回路の内部信号線のとり得る値の集合を二分決定グラフを用いて表現することによって、等価性検証問題を解く手法について述べる。内部信号線を用いて等価性検証を行なう場合には、互いに独立な信号線の集合を用いることが望ましい。そこで、そのような集合を得るために有効な回路構造に基づくヒューリスティックを提案する。ISCAS'85 のベンチマーク結果を用いた実験では 3000 ゲート規模の回路の等価性検証を SUN-4/10 で 20 秒程度で行なっており、その有効性が示されている。

キーワード 論理検証、二分決定グラフ、含意操作

On Functional Implication and Its Application to Equivalence Checking of Combinational Circuits

Yusuke Matsunaga

Fujitsu Laboratories LTD.

1015 Kamikodanaka, Nakahara-Ku, Kasawaki 211

044-754-2664

yusuke@flab.fujitsu.co.jp

Abstract This paper describes a novel equivalence checking method of combinational circuits, which utlizes relations among internal signals represented by binary decision diagrams. To verify circuits efficiently, proper set of internal signals that are independent with each other should be chosen. A heuristic based on analysis of circuit structure is proposed to select such a set of internal signals. Experimental results using ISCAS'85 benchmarks demonstrate how the proposed method is effective. It proves equivalence of two circuits with 3000 gates in 20 seconds on SUN-4/10.

key words equivalence checking, binary decision diagrams, implication

1 はじめに

与えられた2つの組合せ論理回路が機能的に等価かどうかを検証する問題は、論理CADの重要な問題の一つである。例えば、既存の回路を人手で修正した場合に、その修正によって回路の機能が変わっていないかを調べる場合などに等価検証が用いられる。一方、組み合わせ回路の等価検証問題はco-NP完全問題であることが知られており、いかなる問題に対しても効率良く判定できるアルゴリズムは見付かっていないし、将来も見付からないであろうと予想される。そこで、実用的に意味のあるような多くの問題を効率良く判定できるような等価検証手法を開発することが必要とされている。

近年、二分決定グラフを用いた論理関数処理やテストパタン生成の技術の発展によって、今までは難しいとされて来た組み合わせ回路の検証問題のいくつかが解けるようになってきており、これらの技術を組み合わせた検証手法の有効性が確認されている。本稿ではこのような等価検証手法の現状を踏まえて、効率の良い、実用的な等価検証手法の提案を行う。以下、2章では等価検証に関する従来の研究を概観し、3章で提案するアルゴリズムについて述べる。次に、4章でベンチマーク回路を用いた実験結果を示す。

2 従来の検証手法

Bryantの提案した二分決定グラフ(BDD)[1]は論理CADのさまざまな分野に応用されているが、その中でもっとも早く応用され、その効果が実証されたのがこの等価検証であった。藤田らとMalikらが提案した手法[2, 3]は、比較を行う回路の出力の論理関数を表す二分決定グラフを作って等価性を検証するもので、回路構造を用いた入力変数順決定ヒューリスティックによって二分決定グラフの節点数を少なく抑える工夫を行っている。これにより、ISCAS85のベンチマーク回路のうちいくつかの回路に対しては外部出力の論理関数を表す二分決定グラフを作ることに成功している。Rudellの提案した動的変数順変更のヒューリスティック[4]はさらに強力で、c6288を除く全ての回路に対して「良い」変数順を求めることに成功している。しかし、二分決定グラフも万能ではなく、いかなる変数順のもとでも入力数の指数に比例した節点数を必要としてしまう論理関数も存在する。このことから、回路全体の論理関数を二分決定グラフで表現する手法は、実用的な等価検証システムの核として用いることが不適当であると言える。実際、2つの回路の等価検証問題が解けても、その回路全体の論理関数を二分決定グラフで表現できない問題はいくらでも存在する。

実用的な等価検証問題としては2つの回路の間に何らかの類似性が存在する場合がほとんどであり、そのような情報を有効に用いることができれば検証問題の複雑度を下げることができる。Bermanらはそのような検証手法の枠組を提案している[5]。例えば、2つの回路A,Bがそれぞれ部分回路A-1,A-2とB-1,B-2から構成されていた場合、A-1とB-1が等価でかつA-2とB-2が等価ならば、AとBも等価であることが証明できる。このように内部に等価な信号線がある場合には、そこで回路を分割することによってより小さな検証問題に置き換える手法が考えられる。ところが、これだけでは完全な検証は行えない。図 1の2つの回路を考えて見ると、ともに入力側の部分回路は全く等価である(s_1とs_2、t_1とt_2がそれぞれ等価)が、出力側が一方は論理和$(s_1 + t_1)$であるのに対して他方は排他的論理和$(s_2 \oplus t_2)$であり異なっている。それではこの2つの回路は全体として異なっているかというとそうではなくて、機能的には等価である。このように、等価な内部信号線で回路を分割してしまった場合には、個々の検証では等価でないと判定されても全体では等価となる場合があるので注意が必要である。このようなケースはフォールス・ネガティブ(false negative)と呼ばれる。

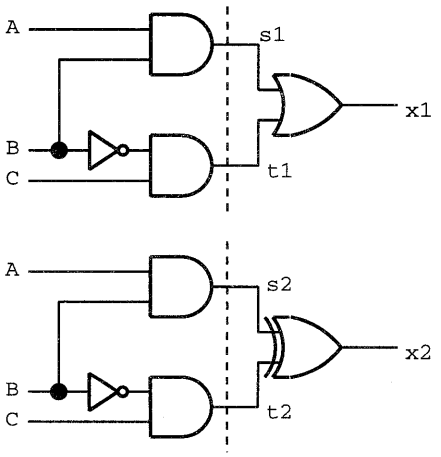

図 1: フォールス・ネガティブの例

文献[5]ではフォールス・ネガティブの問題についての指摘はあるものの、その具体的な方法に関する記述はない。また、基本的な検証の方法として全パタンのシミュ

レーションを用いているため、この手法をそのまま用いることはできないが、内部の等価点の情報を用いるという枠組は有効と思われる。

Kunz は recursive learning と呼ばれる間接的な含意操作を用いて等価検証を行う手法を提案した [6]。Reddy らはこの recursive learning と二分決定グラフを組み合わせた手法を提案した [7] が、基本的には Kunz の手法と同様に ATPG の正当化処理 (justification) を用いている。彼らの論文によれば、フォールス・ネガティブ問題を効率的に解いている、と書いてあるが、実験結果を見る限り 1 分以内に検証問題を解いている例はすべてフォールス・ネガティブ問題を含まないものであり、フォールス・ネガティブ問題を含んだ例の場合には 1 分から 1 ～ 2 時間を要している。さらに、この手法では、再帰レベルの制限された recursive learning 処理のみで内部信号線間の等価関係を求めているため、実際には等価であるにもかかわらず、それが発見できない場合が起こり得る。最終的に、内部信号線間の等価関係を発見し損ねた場合には、外部出力の等価検証を行う際に利用できる情報が減ることになるので、検証が終らなかったり、より多くの計算時間を必要としてしまうなどの無駄を生じる。Jain らの提案した手法 [8] は functional learning と呼ばれる一種の含意操作を用いて内部信号線間の等価関係を求め、それを利用して外部出力の間の等価検証を行う、というものである。この手法も Reddy らの手法と同じで、内部信号線間の等価関係は不完全であり、functional learning のレベルの制限によって等価と判定できない場合がある。例えば、図 2に示した 2 つの回路を検証する際に、内部信号線 $X_1, X_2, \ldots$ と $Y_1, Y_2, \ldots$ が等価だと判定されていれば、これらの信号線の出力側の部分回路は同じ構造を持っているので、組合せ回路 2 に含まれるすべての信号線が等価と判定される。ところが、recursive learning や functional learning の等価判定能力の制限によって $X_1, X_2, \ldots$ と $Y_1, Y_2, \ldots$ の等価判定が行なえなかったとしたら、組合せ回路 2 の信号線は等価と判定されなくなるため、回路全体の等価判定も容易には行なえなくなる。もしも、組合せ回路 2 が乗算器のように、その論理関数を二分決定グラフでは効率的に表せない回路であった場合には、回路全体の等価判定を現実的な時間内には行なえない可能性もある。

このように、等価検証に関してはいくつかの有効なアイデアや枠組は提案されているが、それが実用的な検証手法として完成されているとは言いがたい。

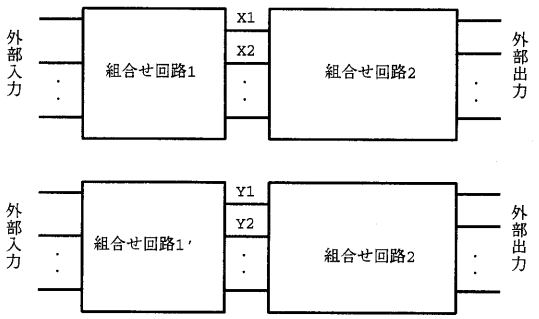

図 2: 検証の例

3 等価検証のアルゴリズム

3.1 用語の定義

ブーリアン・ネットワーク (Boolean network) とは組み合わせ論理回路をテクノロジに依存しないレベルで表現する非巡回有向グラフ $G = (V, E)$ で、入力に対応する節点以外の節点 v はその機能を表す論理関数 F_v を持つ。節点 v の論理関数の入力変数に対応する節点 u を節点 v のファンイン (fan-in) と呼ぶ。また、節点 v を節点 u のファンアウト (fan-out) と呼ぶ。このようなファンインとファンアウトの関係のある節点 u、v 間にのみ有向枝 (u, v) が存在する。節点 v のファンインの集合を FI_v で表す。節点 v のファンアウトの集合を FO_v で表す。以下のように再帰的に定義される節点の集合 TFI_v、TFO_v を節点 v の**推移的ファンイン** (transitive fan-in)、および、**推移的ファンアウト** (transitive fan-out) と呼ぶ。

$$TFI_v = FI_v \cup (\bigcup_{u \in FI_v} TFI_u)$$

$$TFO_v = FO_v \cup (\bigcup_{u \in FO_v} TFO_u)$$

節点 v の論理関数を外部入力の関数として表したものを節点 v の**大域的関数** (global function) と呼び、G_v で表す。G_v は次のように求めることができる[1]。

$$G_v = \exists u \in FI_v, F_v \wedge (\bigwedge_{u \in FI_v} (u \equiv G_u))$$

節点 v と節点の u の大域的関数が等しい時、つまり、$G_v \equiv G_u$ となる時、節点 v と節点 u は**等価**であると言う。互いに等価な節点 v と u の対を**等価節点対**と呼ぶ。

[1]以後、このように節点名をそのまま、その節点に対応した論理変数として用いる。

節点 v に対して、その推移的ファンインの部分集合 T を考える。もしも、外部入力から節点 v に至るいかなる経路も T の要素の節点を一つ以上含んでいる時、T を節点 v の**基底**と呼ぶ。節点 v の論理関数を基底 T の節点を入力として表した関数を、節点 v の T のもとでの**局所的関数** (local function) と呼び、L_v^T で表す[2]。L_v^T も G_v と同じように計算できる。また、外部入力の節点の集合を PI とした時に、任意の節点 v に対して $L_v^{PI} \equiv G_v$ が成り立つ。

等価検証の対象の2つのブーリアン・ネットワークの対応する入力を一つにまとめたネットワークを**コンポジット・ネットワーク** (composite network) と呼ぶ (図 3)。等価検証問題とは、コンポジット・ネットワーク上の対応する出力対 (v, v') が等価かどうかを調べる問題である。

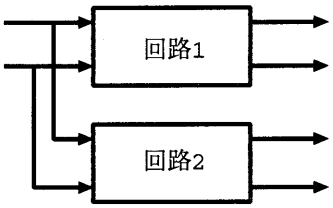

図 3: コンポジット・ネットワーク

3.2 検証手法の枠組

検証の大きな枠組としては、Berman の手法 [5] と同様の手法を用いる。図 4にそのフローチャートを示す。

まず、最初にコンポジット・ネットワークに対して乱数パタンを用いたシミュレーションを行い、等価節点対の候補リスト (CEP リストと呼ぶ) を作成する。次に CEP リストから入力からの幅優先探索順で節点対 (v_1, v_2) を取り出し、その節点対が等価かどうかの検証を行う。もしも等価と分かった場合には、節点対 (v_1, v_2) を検証された等価節点対のリスト (EP リスト) に加える。また、v_2 を含む節点対を CEP リストから取り除く[3]。さらに、図 5に示すように v_2 のファンアウトの節点のファンインの接続を変更し、v_1 を新たなファンインとする。このような処理を CEP リストの全ての節点対に対して行い、最終的に対応する外部出力の対が EP リストに含まれているかのチェックを行えば回路自体の等価検証が行える。

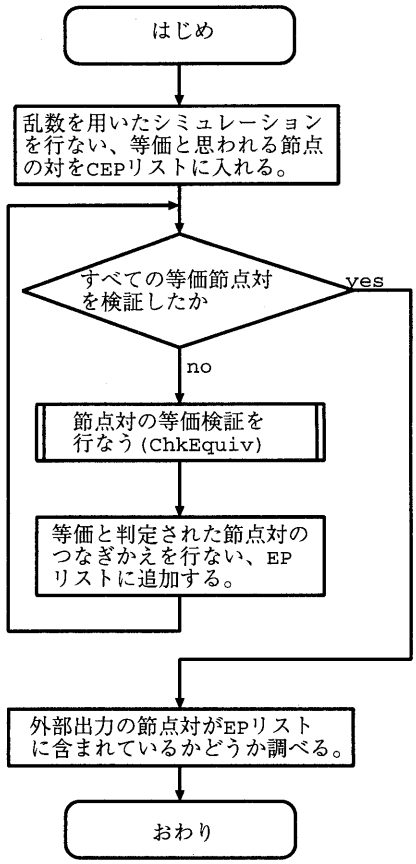

図 4: 検証のフローチャート

3.3 関数的含意を用いた節点対の検証

与えられた節点対 (v_1, v_2) が等価かどうかを検証するために、関数的含意 (functional implication) と呼ぶ手法を用いる。これは Jain らの提案した functional learning[8] と同様に二分決定グラフを用いて節点間の論理的な関係を求めるものであるが、その適用方法や目的が異なっている[4]。

ここでは、ブーリアン・ネットワーク中の節点集合 $S = \{s_1, s_2, \ldots\}$ の値の組の集合から他の節点集合 $T = \{t_1, t_2, \ldots\}$

[2] 基底の要素は必ずしも節点 v の推移的なファンインである必要はないが、TFI_v に含まれない要素は局所的関数を表す時には用いられない

[3] これは例えば、(v_2, v_3) という様な節点対が CEP リストに含まれていた場合、同様に (v_1, v_3) という節点対も CEP リストに含まれているはずなので、検証は v_1 と v_3 が等価かどうかを試すだけでよいからである。この等価関係に関しては推移則が成り立つので、v_1 と v_2 が等価であり、v_1 と v_3 が等価ならば v_2 と v_3 も等価である。

[4] 元来、learning という言葉は、後で含意操作を行うときのための情報を記録する、という意味で用いられていたが、検証で用いられているなかにはそのような意味にそぐわない例も見受けられる。

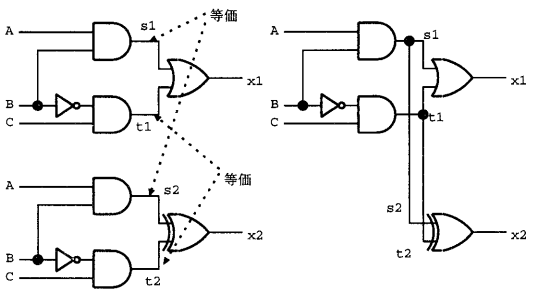

図 5: 等価節点の検証後の接続の変更

の値の組の集合を求める処理のことを**関数的含意**と定義する。例えば、図 6の回路の例で、$S = \{a, b\}$ とし、この節点上の値の組 $\{(a = 1, b = 0), (a = 0, b = 1)\}$ に対する $T = \{c, d, e\}$ 上の値の取り得る組は、$\{(c = 1, d = 1, e = 0), (c = 0, d = 1, e = 1)\}$ である。

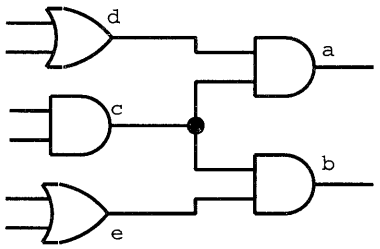

図 6: 関数的含意の例

Kunz の recursive learning ではこのような含意を一つ一つの値に対して行ない、そのすべての含意に共通に現われる割り当てのみを含意の結果として保存する。つまり、上の例では $d = 1$ のみが含意の結果となる。この点が関数的含意と大きく異なるところである。

二分決定グラフを用いて関数的含意を行うには、節点集合 S の各々の要素 s に対して T のもとでの局所関数 L_s^T を計算し、S 上の値の集合 (を表す特性関数)$F_S(S)$ の変数 s をその局所関数 L_s^T で置き換え (文献 [1] でいうところの compose 演算) ればよい。つまり、$S = \{s_1, s_2, \ldots\}$ として、

$$F_T(T) = \exists s \in S, (F_S(S) \wedge (\bigwedge_{s \in S} (s \equiv L_s^T)))$$

のようにして T 上の値の集合 $F_T(T)$ が計算できる。この操作自体は目新しいものではなく、一般には像計算 (image computation) と呼ばれるものである。テスト生成で用いられる含意操作 (learning を用いた間接的含意操作を含む) は、この関数的含意の特殊な例で、値の集合でなく、$a = 1 \Rightarrow b = 0$ といった一組の値を対象に行われるものである。つまり、適切に節点集合 S および T が選ばれていれば、この関数的含意操作のみで従来のテスト生成ベースの手法が行っていた含意操作を全て行うことができる。逆に、関数的含意では判明する節点の値の関係のうち、$a = 1 \Rightarrow b = 0$ といった形の関係では表現できないものも存在する[5]。

この関数的含意を用いた検証アルゴリズムを図 7に示す。

```
boole ChkEquiv(v1, v2) {
    S ← {v1, v2};
    F_S ← v1 ⊕ v2;
    while (F_S ≠ φ) {
        if (S が外部入力節点の集合) return FALSE;
        /* S より次の節点集合 T を求める。 */
        T ← GetT(S);
        /* 関数的含意を行なう。 */
        F_T ← FuncImp(F_S);
        S ← T;
        F_S ← F_T;
    }
    return TRUE;
}
```

図 7: 関数的含意を用いた検証アルゴリズム

まず、最初に検証の対象の 2 つの節点 v_1, v_2 を S として、$F_S(S) = v_1 \oplus v_2$ であるような値の集合を考える。検証のゴールとしては $F_T(T) = \phi$ となるような節点集合 T を求めることである。その場合に 2 つの節点 v_1, v_2 は等価であることが証明される。そのような T が一回の含意操作で求められればよいが、必ずしも最適な節点集合を見付けることができるとは限らないので、何回かの含意操作を経て最終的に $F_T(T) = \phi$ になるような節点集合 T を求めることを目標とする。もしも、そのような節点集合を見付けられずに $v_1 \oplus v_2$ から含意される外部入力節点の値の集合を求めることができた場合には、この 2 つの節点は等価でないことが証明される。

[5] この点に関しては文献 [8] にもその指摘があるが、そのような関係をどのように用いるかに関する具体的な記述はない。

図 7のアルゴリズムが効率良く動くかどうかの鍵は節点集合Tの求め方にある。極端な話、常にTとして外部入力の節点集合を返すのであれば、この検証アルゴリズムは大域関数を二分決定グラフで表す検証手法となんら変わることはない。そこで、この節点集合Tの求め方に関して役立つと思われる性質について以下のような考察を行った。

性質1 TはSの各要素sの基底となっていなければならない。これは関数的含意の計算方法より明らか。

性質2 その推移的ファンアウトTFO_vの中にSの要素が1つ以下しか含まれていない様な節点vのみからなる節点集合Tでは、関数含意の結果$F_T(T) = \phi$となることはない。$\Rightarrow$ TはSの複数の節点に関係しているような節点を少なくとも一つは(たぶんできるだけ多く)含む必要がある。

性質3 フォールス・ネガティブ問題は、充足性から生じる関係を持った節点があたかも独立変数のように扱われることに起因する。$\Rightarrow$ Tの節点は互いに独立性が高いほうが望ましい。

このような考察をもとに、与えられた節点集合Sから次の節点集合Tを求めるヒューリスティックを開発した(図 8)。

まず、Sの各々の要素から推移的ファンインにラベル付けを行うことで、各節点vがいくつの要素に関係しているかを調べる。この値をl_vとする。次に、各節点vに対して次のような再帰手続きによってm_vを計算する。

$$m_v = \max(l_v, (\max_{u \in FI_v} m_u))$$

この値は自分、および自分の推移的ファンインのなかのl_vの値の最大値を表している。基本的に、Sの各要素sからファンイン方向に深さ優先探索を行って、$m_v \equiv l_v$であるような節点vをTの候補とする。これは上の性質2にしたがった選択である。外部入力では必ず$m_v \equiv l_v$が成り立つので、sから外部入力の間のどこかの節点vで必ず$m_v \equiv l_v$が成り立つ。このことより、この方法で得られたTがsの基底となっていることは明らかであり、性質1は満たされる。次に、性質3を考慮するための補正を行う。単純に考えると2つの節点uとvの間に推移的ファンイン、推移的ファンアウトの関係がある場合には、この2つの節点は独立ではないと言える。そこで、推移的ファンアウト側の節点をTから除いてしまう処理が考えられるが、Tは基底でなければならないために代わりに0個以上の節点がTに

```
節点集合 GetT(S) {
    foreach s ∈ S {
        s の推移的ファンインにラベルをつける。
    }
    foreach v ∈ ラベル付けされた節点 {
        ラベル付けされた回数を l_v に設定する。
        m_v の値を計算し、設定する。
    }
    do {
        すべての節点の印を消す。
        foreach s ∈ S {
            s から DFS を行ない、
            l_v ≡ m_v であるような節点 v に印を付ける。
        }
        chg ← FALSE;
        foreach v ∈ 印の付いた節点 {
            n ← 印の付いていないファンインの数;
            if (n ≤ 1) {
                l_v ← 0;
                chg ← TRUE;
            }
        }
    } while (chg = TRUE);
    return 印の付いた節点;
}
```

図 8: 節点集合を求めるヒューリスティック

加えられることになる。この処理によってTの要素数が増えることは好ましくないので、Tの他の要素の推移的ファンアウトであるような節点で、かつ、その節点を削除することでTの要素数が増えない(自分の代わりに多くとも1つしか追加されない)ようなものに対しては削除を行う。具体的にはそのような節点のl_vの値を0にして、もう一度Tを求める処理を行う。このようなループを変化が起こらなくなるまで繰り返す。

このヒューリスティックに従って節点集合を求める例を図 9に示す。

まず、a、bの推移的ファンインにラベル付けを行ない、l_vの値を計算する。この例では、節点c、eがaのみの推移的ファンインになっているほかはa、bの両方の推移的ファンインとなっている。つぎに、自分、および自分の推移的ファンインのl_vの値を最大値を計算し、それをm_vに

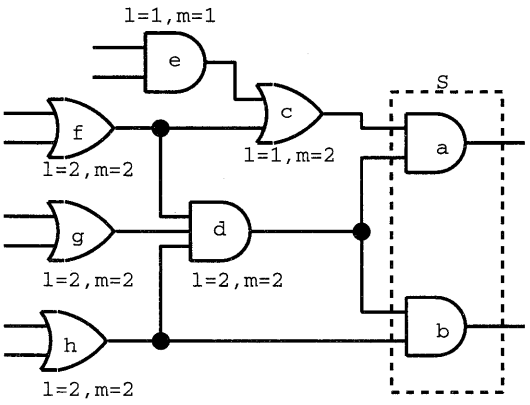

図 9: 節点集合を求める例

設定する。節点 c は自分の l_c は 1 であるが、そのファンインである節点 f のラベル値 l_f が 2 なので、$m_c = 2$ となる。それ以外の節点は $l_v = m_v$ となっている。この状態で、T の候補を求めると、$\{e, f, d, h\}$ となる。しかし、節点 d に注目してみるとその 3 つのファンインのうち 2 つは T の候補となっている。そこで、この d は T の候補に含めるのをやめ、$l_d = 0$ としてもう一度 T の候補を求めると、$\{e, f, g, h\}$ を得る。この時点で、T の候補の間に推移的ファンイン、推移的ファンアウトの関係はないので処理が終る。実際、節点 d は、節点 f や節点 h が 0 になると自分も 0 になるのでこれらの節点に対して独立でない。そこで、このような節点 d は除いておいたほうが、フォールス・ネガティブの問題を回避できる可能性がある。

4 実験結果

以上のような検証手法を実装し、実験を行った。ただし、効率化のために以下のような実装上の工夫を行なっている。

- 図 7のアルゴリズムは比較的遅いので、最初は、対象となる節点対にもっとも近い等価節点を基底として局所的関数を求め、それを比較する。その局所的関数が一致すればさらなる検証は必要ない。そうでない場合には図 7のアルゴリズムを起動する。
- 等価な節点対だけでなく、互いに否定の関係にある節点対も検証する。検証はほとんど同様に行なえる。検証された後の接続のつなぎ替えでは極性を反転させる。
- 二分決定グラフの変数順は図 8で印を付ける順序をそのまま用いている。動的変数順変更などは一切、行なっていない。

等価検証のベンチマークとして ISCAS'85 の回路のオリジナルと冗長故障を取り除いたもの (例えば、c432 と c432nr) を用いた。使用計算機は SUN-4/10 である。

表 1: 等価検証の実験結果

回路名	入力数	出力数	CPU 時間 (秒)	
			文献 [7]	本手法
c432	36	7	2.2	0.64
c499	41	32	2.17	0.88
c1355	41	32	6.73	2.61
c1908	33	25	14.54	4.63
c2670	233	139	159.3	3.38
c3540	50	22	67.64	18.99
c5315	178	123	372.8	8.36
c6288	32	32	32.74	7.09
c7552	207	108	5583.3	13.73

文献 [7] でも同様の実験を行なっているので比較を行なった。しかし、使用計算機についての記述がないため厳密な比較は行なえないが、計算時間で数百倍の差がついている場合もあり、使用計算機に関わらず本稿で提案する手法の有効性は明らかであると思われる。また、文献 [8] についても FJ1、FJ2 と名付けられた回路を用いて比較を行なった。これらの回路はもとは一つの回路を出力ごとに切り出したものである。文献 [8] では SUN-4/10 を用いて FJ1 の検証に 590 秒、FJ2 の検証に 6 時間半を要している。切り出す前の回路 (FJ1 + FJ2 + α) を本稿の手法で検証を行なったところ、SUN-4/10 で 11 秒であった。このように有意な差がついた理由は、2章で述べたように、文献 [7] や文献 [8] の手法が内部の等価節点対を完全には列挙していないこと、および、フォールス・ネガティブ問題を回避するような回路分割をうまく行なえていないことに起因するものと思われる。

次に節点集合を求めるヒューリスティックの性能を評価するために、図 7のアルゴリズム中のループが何回、回ったかを計測した。図 8のヒューリスティックがうまく働いていれば、このループ回数が低くなると期待される。表 1の実験の際に、等価検証ルーティン (ChkEquiv) が起動された回数は 242 回 (検証を行なった等価節点対の

総数は 17154 組) で、一回の検証の際の平均ループ回数は 1.56、最大ループ回数は 6 であった。ChkEquiv 中で最低一回はループを回るので、この値は節点集合の選択のヒューリスティックが効果的であることを示していると言える。

5 おわりに

以上のように、関数的含意を用いて組合せ回路の等価性検証を行なう手法および、関数的含意の際に用いられる節点集合の選択方法を提案した。実験結果が示すように、本手法は実用的な例に対して効率良く処理を行なっている。一般に、co-NP 問題に対しては本手法のように二分決定グラフを用いた網羅的な探索が効果的と思われる。一方、等価ではないのに CEP リストに入ってしまったような節点対を等価でないと判定する問題は、テスト生成と同様に NP 完全問題であり、本手法の関数的含意よりは従来のテスト生成の手法のほうが効果的と思われる。今回の実験では 1000 パタン程度の乱数シミュレーションを行なうことで、ほとんどすべての等価でない節点対を区別できているが、より実用的に堅牢なシステムとするためには、テスト生成の手法と組み合わせる必要があると思われる。

また、この検証手法で用いた関数含意はいわゆる論理合成で言うところの、充足性のドントケア (satisfiability don't cares: SDCs) を表していると言うことができる。従来、大域的関数を二分決定グラフで表すことが難しいような規模の組合せ回路の簡単化に関しては、テスト生成手法をベースにしたものが有効と言われてきたが、これは冗長故障の判定 — いわゆる co-NP 問題 — を扱っているわけで、果たしてそれが本当に最良の方法なのかは疑問である。等価検証と同じように二分決定グラフをベースにしたより効率的な手法の開発が今後の課題である。

参考文献

[1] R.E. Bryant, "Graph-based algorithms for boolean function manipulation", *IEEE Transactions on Computer*, C-35(12), 1986.

[2] M. Fujita, H. Fujisawa, and N. Kawato, "Evaluation and Imporovements of Boolean Comparison Method Based on Binary Decison Diagrams", In *Proc. of ICCAD*, pp. 2-5, Nov. 1988.

[3] S. Malik, A. Wang, R. Brayton, and A. Sangiovanni-Vincentelli, "Logic Verification Using Binary Decision Diagrams in a Logic Synthesis Environment", In *Proc. of ICCAD*, pp. 6-9, Nov. 1988.

[4] R. Rudell, "Dynamic Variable Ordering for Ordered Binary Decision Diagrams", In *Proc. of ICCAD*, pp. 42-47, Nov. 1993.

[5] C.L. Berman and L.H. Trevillyan, "Functional Comparison of Logic Designs for VLSI Circuits", In *Proc. of ICCAD*, pp. 456-459, Nov. 1989.

[6] W. Kunz, "Hannibal: An Efficient Tool for Logic Verification Based on Recursive Learning", In *Proc. of ICCAD*, pp. 538-543, Nov. 1993.

[7] S.M. Reddy, W. Kunz, and D.K. Pradhan, "Novel Verification Framework Combining Structural and OBDD Methods in a Synthesis Environment", In *Proc. of 32nd DAC*, pp. 414-419, Jun. 1995.

[8] J. Jain, R. Mukherjee, and M. Fujita, "Advanced Verification Techniques Based on Learning", In *Proc. of 32nd DAC*, pp. 420-426, Jun. 1995.