設 計 自 動 化 83－3
（1997. 2. 14）

出力変数の分割による大規模回路の特徴関数生成手法

田治米 純二† 新井 浩志‡ 深澤 良彰†

†早稲田大学理工学部 ‡千葉工業大学工学部

概要

本稿では組合せ回路の出力変数をいくつかのグループに分割することにより、出力変数がとりうる論理値の集合を表す特徴関数を高速に生成するための手法を提案する。特徴関数は有限集合を表現する手段の一つであり、集合の要素を2進符号化して論理関数で表現したものである。近年、二分決定グラフ(以下BDD)を用いた技術の発展により論理関数処理を効率的に行うことができ、大規模な組合せ回路に対する特徴関数を生成することが可能となってきている。しかし、依然として扱える回路規模には限界がある。本手法では、論理関数の出力変数を相互に関連の強いものから構成されるグループに分割して特徴関数を生成する。これにより、必要となるBDDのノード数は少なくなり、従来よりも大規模な回路の特徴関数を生成することができる。この方法で得られた特徴関数は真の解を持つ特徴関数とは異なる。しかし、論理検証などにおいて周辺回路から入力の制約を求める場合など、必ずしも真の解を求める必要はない。本手法により、真の解に近い解を持つ特徴関数を大規模な回路から取り出すことが可能となる。

A Method to Generate a Characteristic Function for Large Scale Circuit by Splitting Output Variables

Junji TAJIME † Hiroshi ARAI ‡ Yoshiaki FUKAZAWA †

†School of Science and Engineering, Waseda University
‡Faculty of Engineering, Chiba Institute of Technology

Abstract

This paper presents a method to generate a characteristic function for large scale circuit. Recently, a characteristic function for practical size of logic circuit can be obtained relatively easy, however, it is still difficult for the proposed method to generate a characteristic function for large scale circuit. By our method, output functions are splitted into some groups and characteristic functions for each groups are generated . As a result, we can get the characteristic function with shorter computational time and smaller BDD nodes size. Though the generated function is redundant, such characteristic function is enough to get input constraints from surrounding circuits and so on. In our method, relativeness of output functions is determined by the number of common primary inputs, and output functions are splitted into groups based on this relativeness.

1 はじめに

半導体技術の進歩により、設計される回路規模は増大してきている。そのため、設計された回路の正当性を検証することが論理CADの重要な問題の一つとなっている。近年、Bryantによって提案された二分決定グラフ(Binary Decision Diagram、以下BDDと略す)[1]は変数順序決定アルゴリズムなどの様々な技術の発展によって、組合せ回路の検証において今までは困難とされてきた問題を解くために利用することが可能となってきている。順序回路においてもBDDを用いた等価性検証手法[2],[3]が提案され、従来よりも大規模な回路が扱えるようになった。しかし、依然として扱える回路の規模には限界がある。本研究では、検証対象回路とその周辺回路を考える。大規模回路の設計では、全ての機能をまとめて実現することはなく、機能ごとに設計、実現していく。このため、ある回路を考えた時、その回路の入出力と接続された周辺部分の回路が存在する(図1)。もし、検証対象回路の入力と接続されている周辺回路から入力制約が取り出せるならば、その入力制約を検証に用いることにより、確かめるべき検証項目を削減でき、検証を効率的に行なえる。ここでいう入力制約とは、周辺回路がとりうる出力値の集合のことである。そこで、本稿では検証対象回路の周辺回路の出力値集合(検証対象回路の入力値集合)を求めるための手法について提案する。組合せ回路の出力値集合を表現する方法としては、集合の各要素を2進符号化して集合の要素を論理関数であらわす特徴関数(characteristic function)を用いる。本手法では大規模回路の特徴関数を求めるために、出力変数をいくつかのグループに分割する。そしてグループごとに特徴関数を求めることにより、特徴関数の生成過程で必要なBDDのノード数を減らし、より大規模な回路からも特徴関数を生成することができる。出力変数を分割することによって扱える回路規模は大きくなるが、得られる特徴関数は冗長な解となってしまう。しかし、本研究の目的から必ずしも正確な特徴関数を得る必要はなく、より真の集合数に近い特徴関数を求めることが目的となる。このような特徴関数を得るために、本稿では、出力に影響を与える入力集合から、関連の強い出力変数同士をまとめる。この結果、要素数の少ない解集合を持つ特徴関数を得ることができる。

2 集合の論理表現

2.1 定義

以下では集合$\{0,1\}$をBで表す。

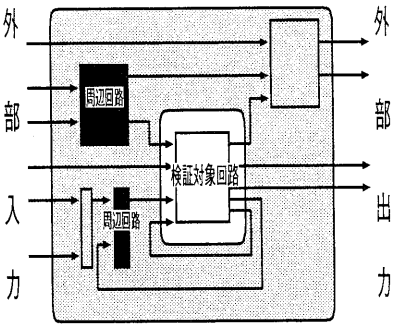

図1: 周辺回路の特徴関数を用いた検証

関数ベクトル 論理関数ベクトル$F=[f_1(\vec{x})...f_m(\vec{x})]$は$F : B^n \rightarrow B^m$なる関数である。ここで$\vec{x}=[x_1...x_n]$は論理関数$f_1...f_m$の入力変数ベクトルである。

$Img(F)$はFの像を表し、$Img(F) =_{\text{def}} \{\vec{y} \in B^m | \vec{y} = F(\vec{x}), \vec{x} \in B^n\}$で定義する。ここで、$\vec{y}=[y_1...y_m]$は$y_i \equiv f_i$なる出力変数ベクトルである。

特徴関数 論理関数ベクトルFが与えられた時、Fの像$S=Img(F)$に対して、Sの特徴関数χ_Sは式(1)のような$\chi_S : B^m \rightarrow B$の関数として定義される。

$$\chi_S(\vec{y}) = \begin{cases} 1 & \vec{y} \in S \\ 0 & \vec{y} \notin S \end{cases} \quad (1)$$

例えば、集合$\{[0\ 0],[1\ 1],[0\ 1]\}$の特徴関数は$\overline{y_1}+y_2$となる。逆に、B^mのいかなる部分集合Sに対しても特徴関数χ_Sが存在する。

コファクタ演算 関数ベクトルから特徴関数を求める時に用いるコファクタ演算について定義する。論理関数ベクトルFと、B^nの部分集合S'の特徴関数cが与えられた時、Fのcに対するコファクタ$F_c = [(f_1)_c...(f_n)_c]$とは、$B^n$から$B^m$への関数であり、定義域が$S'$のもとでは$F$と等しい値域を持つ関数である。$F$と$c$から、この様な関数$F_c$を求める計算をコファクタ演算と呼ぶ[3]。その計算方法は図2のようになる。ここで、F_x, c_xと$F_{\overline{x}}, c_{\overline{x}}$は$F$と$c$で用いられている任意の変数$x$にそれぞれ1と0を代入して得られる論理関数を表す。

```
function cofactor(F, c){
  if(c = 1 or is_constant(F)) return F;
  else if(c_x̄ = 0) return cofactor(F_x, c_x);
  else if(c_x = 0) return cofactor (F_x̄, c_x̄);
  else return x· cofactor(F_x, c_x)
              +x̄· cofactor(F_x̄, c_x̄);
}
```

図 2: コファクタ演算

```
function vec_to_char(F_k, c){
  F'_k=cofactor(F_k,c);
  if(k ≠ m){
    if(f'_k = 1) return y_k· vec_to_char(F'_{k+1},c);
    else if(f'_k = 0)
            return ȳ_k· vec_to_char(F'_{k+1},c);
    else return y_k· vec_to_char(F'_{k+1},f'_k)
                +ȳ_k· vec_to_char(F'_{k+1},f̄'_k);
  }
  if(k = m){
    if(f'_m = 1) return y_m;
    else if(f'_m = 0) return ȳ_m;
    else return 1;
}}
```

図 3: 特徴関数の生成

2.2 関数ベクトルから特徴関数への変換

関数ベクトルから、特徴関数を求める方法を図3に示す。ここで、F_k は F に対して、F_k : $B^n \rightarrow B^{m-k+1}$ なる関数ベクトル $F_k =_{\text{def}} [f_k \ f_{k+1} \cdots f_m]$ と定義する。関数ベクトル $F = [f_1 \cdots f_m]$ の特徴関数 $\chi_S(\vec{y})$ は χ_S=vec_to_char(F,1) により求めることができる。

反例はあるが、多くの場合に F よりも F_c のBDDのノード数の方が小さくなるため、コファクタ演算を用いた特徴関数の生成方法は有効であることが知られている。しかし、このアルゴリズムを用いても、特徴関数の生成には指数に比例する計算時間を必要とする。このため、大規模な回路の特徴関数を求めるためには、回路全体の論理を直接扱うのではなく、分割して特徴関数を生成する必要がある。以下では出力変数の分割による特徴関数の生成について述べる。

3 提案手法

3.1 アルゴリズムの概要

アルゴリズムは、大きく分けて、入力変数の順序付け、出力変数のグループ化、特徴関数の合成の3段階からなる。まず、論理回路をBDDで表現し、さらに出力変数をグループ化する処理のために、与えられた論理回路の入力変数の順序と出力変数の優先度を決定する。次に出力変数の分割では、出力変数の評価値をもとに出力変数のグループ化による分割を行なう。最後に、出力変数のグループごとに図3の手法を用いて、各々のグループに対する特徴関数を求め、それらを合成することで最終的な特徴関数を得る。

3.2 入力変数の順序付け

論理関数をBDDで表現した場合、入力変数の順序によってBDDの形が異なり、処理時間が大きくかわる。このため、BDD処理において、入力変数の順序付けは重要である。本手法は、動的重み付け法 [4] と出力の優先度による方法 [5] の組合せにより入力変数順序を決定する。

動的重み付け法 BDDの入力変数順序付けにおいては、以下の2つの経験則を用いることにより、BDDのノード数を削減できることが知られている。

1. 出力を制御する入力変数を上位にする
2. 局所計算性のある入力変数の組を近い順にする

この方法では、入力に対して重みを決定し、重みが最大の入力を一番制御力の強い入力として選択する。そして、選択した入力を削除したものとし、新たに重み付けを動的に行なうことにより、上記2の経験則を満たすようにしている。重み付けの方法としては、出力に重み1を与え、ゲートのファンイン数で均等に重みを分割して、重みをファンインに伝達していく。ファンインが枝分かれしている場合は、それぞれの重みの和とする。この操作を入力まで適用することで入力変数の重みを求めることができる。その様子を図4に示す。

出力の優先度による方法 上記の変数順序決定法は1出力関数について考慮したものであり、多出力関数に関しては考慮されておらず、段数による優先度で並べているだけである。このため、優先度の高い出力のノード数を減らすことはできる

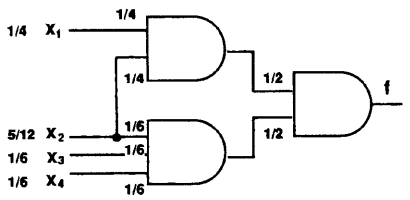

図 4: 重み付け法

が、多出力回路全体のノード数が減るとは限らない。一般に回路が多出力で構成されていることを考えると、多出力回路全体のノード数を減らすことを考慮した方法が必要となる。出力関数の優先度による方法では出力関数の優先度が高いものから入力変数の順序付けを行なう。優先度が低い出力変数から求めた入力変数順序にまだ順序が決まっていない変数が含まれていた場合には、これを、既に決定している入力変数順序に挿入していくことで、多出力回路全体のノード数を減らすようにしている (図 5)。出力関数の優先度としては、回路全体を制御する出力は、最も多くの入力変数に関係しているものと考え、入力変数の数が多いものから優先度をつける。入力変数の数が同じ場合には、より多くのゲートに関連している出力関数の優先度を高くしている。

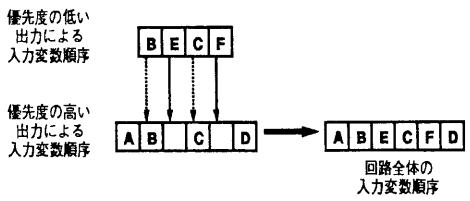

図 5: 変数の挿入

3.3 出力変数のグループ化

関数ベクトル $F = [f_1 ... f_m]$ に対して $Img(F) = Img(SF_1) \times ... \times Img(SF_p)$ なる部分関数ベクトル $SF_1, ..., SF_p$ を求めることができれば、分割によって特徴関数 χ_S を求めることができる。しかし、一般に、部分関数ベクトル $SF_1, ..., SF_p$ を求めることが出来るとは限らない。そこで、出力変数の集合を、決められた要素数のグループに分割することを考える。この方法だと、分割で求められた $SF_1^*, ..., SF_q^*$ は $Img(F) \subseteq Img(SF^*) = Img(SF_1^*) \times ... \times Img(SF_q^*)$ となり $Img(F)$ に等しい出力論理値の集合は得られない。しかし、第 1 章で述べた背景より我々は $Img(F) \subseteq Img(SF^*)$ を満たす集合の要素数 $|Img(SF^*)|$ がより $|Img(F)|$ に近い集合を求めることを目的とする。そのために、関連の強い出力変数を 1 つのグループにまとめることが重要となる。

3.3.1 変数の重複

関数 f、g の入力変数集合 I_f、I_g に対して $I_f \cap I_g = \emptyset$ が成り立つならば、関数 f、g は共通な入力変数を持たない。そのため、お互いの関数同士はその出力値に影響を及ぼすことはなく、各々の像は独立となる。関数ベクトル $[f \quad g]$ の像は $Img([f\ g]) = Img(f) \times Img(g)$ により求められ、f、g を分割しても $Img([f\ g])$ を求めることが出来る。

しかし、一般には $Img([f \quad g]) \subseteq Img(f) \times Img(g)$ となる。このことから、関数ベクトルを決められた要素数に分割する場合、出力変数の重複を許すことにより、より真の出力集合値集合に近い集合が得ることができる場合がある。

例えば以下のような関数ベクトル F を考える。

$F = [\ f_1\ f_2\ f_3\ ]$

但し、 $f_1 = x_1 \cdot x_2 \cdot x_3$,

$f_2 = x_1 + x_2 + x_3$, $f_3 = x_1 \oplus x_2 \oplus x_3$

この関数ベクトルに対して以下の二つの分割を適用するものとする。ここで、分割 A は変数の重複がない場合、分割 B は変数の重複を許す場合である。

分割A : $SF_1^* = \{f_1,\ f_2\}$, $SF_2^* = \{f_3\}$

分割B : $SF_1^* = \{f_1,\ f_2\}$, $SF_2^* = \{f_2, f_3\}$

これらの分割によって得られる特徴関数に基づいた出力論理値集合は以下の通りとなる。

真の論理値集合:

$Img(F) = \{[000], [010], [011], [111]\}$

分割 A による論理値集合:

$Img(SF_1^*) \times Img(SF_2^*)$

$= \{[00], [01], [11]\} \times \{[0], [1]\}$

$= \{[000], [001], [010], [011], [110], [111]\}$

分割 B による論理値集合:

$Img(SF_1^*) \times Img(SF_2^*)$

$= \{[00], [01], [11]\} \times \{[000], [010], [011], [110], [111]\}$

$= \{[000], [010], [011], [110], [111]\}$

本手法では、出力変数の重複を許すことにより、

より真の解に近い解を持つ特徴関数を生成する。

3.3.2 出力変数の関連

分割して求めた特徴関数が真の特徴関数に近い解を持つためには、関連の強い出力変数を1つのグループにまとめる必要がある。ここで、ある出力関数 f、g、h と、それに対応する出力変数 y_f、y_g、y_h があった場合に、「出力変数 y_g よりも y_h の方が、y_f との関連が強い」とは、以下の式(2)を満たすものと定義する。

$$|Img([f\ g])| < |Img([f\ h])| \qquad (2)$$

関連の強い出力変数をまとめるための基準としては共有している入力変数とそうでない入力変数の2つを考える。3.3.1節で述べたように共通な入力変数を持たない出力関数はお互いの出力値に影響を与えない。このことから、共有している入力変数の数が多いもの程、関連の強い出力変数となる可能性が高くなる。これに対して共有しない入力変数が多い出力変数は、別の出力変数との関連が多くなる。これらのことから、なるべく多くの入力変数を共有し、共有しない入力変数が少ない出力変数を一つのグループにまとめることを考える。我々は出力関数 f と出力関数の集合 SF との評価値 $evaluate(f, SF)$ を式(3)で定義し、この評価値を用いてグループ化を行なう（/ は差集合をあらわす）。

$$evaluate(f, SF) = |I_f \cap I_{SF}| - |I_f / I_{SF}| \qquad (3)$$

3.3.3 分割アルゴリズム

出力変数のグループ化の手順は以下のようになる。ここで、1グループあたりの出力変数の数を分割サイズと呼び、その数を d で表す。

ステップ0 分割される出力関数ベクトルに含まれる出力関数の集合を SF とし、$i = 1$、$Select = \emptyset$ とする。

ステップ1 出力関数の中で他のどの出力関数とも共有する入力変数を持たないもの出力関数の集合を SF_e とする。SF_e は次式(4)を満足する。

$$SF_e = \{f_e |\ I_{f_e} \cap I_f = \emptyset,\ f_e \in SF\} \qquad (4)$$

但し、$f \in SF/SF_e$

$SF \leftarrow SF/SF_e$ とし、次のステップに進む。

ステップ2 SF に含まれる出力関数 f の中で $|I_f|$ が最大のものを f_{piv} とする。すなわち、f_{piv} は次式(5)を満足するものとする。

$$|I_{f_{piv}}| \geq |I_f|,\ f \in SF \qquad (5)$$

このような出力関数が二つ以上存在する場合は、それぞれの関数を構成するゲート数が最大のものを f_{piv} とする。

$SF_i = \{f_{piv}\}, Select \leftarrow Select \cup \{f_{piv}\}$, $SF \leftarrow SF/\{f_{piv}\}$ とし次のステップに進む。

ステップ3 f_{piv} と入力変数を共有する出力関数の集合 $Relate$ を求める。

$$Relate = \{f |\ I_{f_{piv}} \cap I_f \neq \emptyset, f \in (SF \cup Select)\} \qquad (6)$$

ステップ4 SF_i との評価値を計算し、f_{max} を求める。f_{max} は評価値が最大の出力関数である。

- $|SF_i Select|$ が $[d/2]$ より小さい場合
  $evaluate(f_{max}, SF_i) \geq evaluate(f, SF_i)$,
  $f \in Relate$;

- $|SF_i Select|$ が $[d/2]$ 以上の場合
  $evaluate(f_{max}, SF_i) \geq evaluate(f, SF_i)$,
  $f \in (Relate/Select)$

$SF_i \leftarrow SF_i \cup \{f_{max}\}$、$Relate \leftarrow Relate/\{f_{max}\}$ とし、もし $f_{max} \in Select$ ならば、$SF_i Select \leftarrow SF_i Select \cup \{f_{max}\}$ $f_{max} \notin Select$ ならば、$Select \leftarrow Select \cup \{f_{max}\}$、$SF \leftarrow SF/\{f_{max}\}$ とする。

ステップ5 もし、$SF = \emptyset$ ならば分割処理は終了。$Relate = \emptyset$ または $SF_i = d$ ならば、$i \leftarrow i + 1$ として、ステップ2に戻る。どちらでもない場合は、f_{max} を計算するためにステップ4へ戻る。

ステップ1の式(4)を満たす出力変数は、どの出力変数とも独立な出力変数となるので、特徴関数は恒真となる。このため、計算する必要がないので分割グループには含めない。

ステップ3では f_{piv} と独立した出力関数を評価の出力関数から除いている。従来の我々の方法[6]では、この条件は満たしていなかったために、

f_{piv} と全く独立であっても、それ以外の出力関数と関連が強いものが選択されていた。このため、評価値の高い出力関数が選択されると、毎回同じ出力関数が選択されることがあった。この条件により、f_{piv} を基準とした関連の強い出力関数が選択される。

ステップ4では重複する出力変数の数に制限がないと、最悪の場合非常に多くの出力変数を選択してしまう。このため、1グループあたり重複する出力変数の数の制約として $[d/2]$ を与えている ([x] は x をこえない最大の整数)。

3.4 特徴関数の合成

上述の方法により分割された出力関数グループ $SF_1^*, ..., SF_q^*$ の各々の特徴関数 $\chi_{S_1}, ..., \chi_{S_q}$ を 2.2 節で示した従来手法で求める。求める出力値集合は $S = S_1 \times ... \times S_q$ の集合演算により得られる。ここで、特徴関数で表現された集合の直積は、特徴関数の論理積により求めることができるので、各々の特徴関数の論理積 $\chi_S = \chi_{S_1} \cdot ... \cdot \chi_{S_q}$ により特徴関数を求め、最終的な出力値集合を得ることができる。

4 実験結果

以上のアルゴリズムを C 言語で実装し、実験を行なった。使用計算機は Sun の SS20(hyperSPARC 150MHz、メモリ 256MB) で、BDD 用パッケージとしては、文献 [4] の SBDD パッケージ ver6.0 を使用した。BDD ノード数の最大値は 1,000,000 としている。評価対象回路としては KUE-CHIP2[7] のコントローラ部 (controller)，命令レジスタ & デコーダ部 (ir_idc)，そして ISCAS'85 のベンチマークデータを使用した (表 1)。c2670、c6288、c7552 に関しては BDD のノード数が最大値を越えしまったため、また c1355 は c499 と論理が等しいため除いてある。

4.1 最大ノード数、計算時間

表 2 に分割処理中の BDD の最大ノード数と特徴関数を求めるのに要した時間を示す。時間の計測は time 関数を用いて行なった。今回の実験は、最大ノード数が 1,000,000 ノードのため、950,000 ノードを越えるごとにガーベージコレクションを行なっており、G.C. で表している。N.F. は 24 時間計算を行なっても処理が終了しなかったことを表しており、$(> 1M)$ は BDD の最大ノード数を越えてしまったことを表している。その結果、計測不可能な部分には – が記入されている。

回路名	入力数	出力数	分割サイズ	ノード数
controler	50	56	20	917
ir_idc	9	24	10	25
c432	36	7	4	25657
c499	41	32	10	29683
c880	60	26	10	19703
c1908	33	25	10	7865
c3540	50	22	10	214943
c5315	178	123	20	6197

表 1: 評価対象回路

分割なしの場合と分割ありの場合について、最大のノード数と計算時間を比較する。まず最大ノード数については，分割なしの場合より分割ありの場合の方が，より少ないノード数で特徴関数を求めることができる。時間に関しても、分割なしの場合より分割ありの場合の方が，より短い時間で特徴関数を求めることができている。規模の小さい回路に対して，多くの時間がかかっているものがあるが，分割によって求めたものは 1 グループの特徴関数を生成するたびにガーベージコレクションをおこなっているためと思われる (後述)。

次に出力変数の重複の有無について比較する。重複のある場合は重複なしの場合に比べて重複の分だけ関数の呼びだし回数が多くなるため、多くの計算時間を必要とする．そして，最大ノード数も多くなっている。しかし、4.2 節で後述する様に，重複ありの場合の方が，真の解に近い特徴関数が得られており、真の解に近い解の特徴関数はその論理が複雑になり，グラフの冗長部分が少なくなる傾向にあるためグラフのノード数が多くなるものと考えられる。

図 6に、特徴関数生成処理中における BDD のノード数の変化と再帰呼び出し回数の変化を示す。この例では一時的に確保された BDD を削除するため、再帰関数呼び出し 10 回ごとにガーベージコレクションを行なっている。このアルゴリズムでは、出力関数の数の指数に比例した呼び出しが行なわれる。

ノード数に関して、まず、出力関数ベクトルを分割して特徴関数を求めた場合と、分割を行なわずに求めた場合のノード数を比較する。分割した場合には特徴関数に冗長な部分ができ簡単になってしまうため、ノード数を単純に比較することはできないが、分割を行なわない場合にはノー

回路名	最大ノード数			計算時間 [1/60sec]		
	分割なし	分割あり		分割なし	分割あり	
		重複なし	重複あり		重複なし	重複あり
controler	80583	1025	31342	142320	227	373
ir_idc	1725	153	687	31	172	237
c432	42752	32199	37907	71	92	154
c499	-	617232	620157	N.F.	3242	3351
c880	-	G.C.	G.C.	N.F.	5250	5929
c1908	G.C.	105586	186666	371660	580	956
c3540	($> 1M$)	G.C.	G.C.	-	9151	22404
c5315	-	($> 1M$)	-	N.F.	-	N.F.

N.F.: Not finished, G.C.: Garbage collection, ($> 1M$): Node overflow

表 2: 最大ノード数、計算時間

ド数はほぼ単調増加となる。これに対して、分割による方法では、1グループの特徴関数を求めるごとに不必要な出力関数を削除することができるため、ノード数の増加を少なく抑えることができる。ただし、グループ間で出力関数を重複させている場合は、重複している出力関数は特徴関数生成に使わなくなるまで削除することはできないため、重複のないものよりも削除できるサイズが少ない。今回のデータでは、特徴関数を生成する回路の規模が小さいので問題にならないが、出力関数ベクトルのノード数が多い場合には、有効なことである。

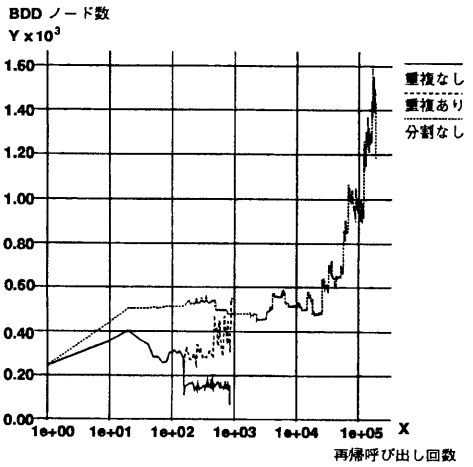

図 6: ノード数の変化

4.2 集合数の比較

4.1 に示したように、出力関数ベクトルをいくつかのグループにして分割して別々に特徴関数を求め，この特徴関数を合成して，元の出力関数ベクトルの特徴関数を求めることにより、最大ノード数は減り、計算時間を短縮できる。しかし、本手法により、実際の集合数を，どれだけ真の解集合に近くすることができるかが重要である。表 3 には分割を行なわない場合、分割を行なうが重複のない場合、重複を許す場合 (本提案手法) の比較を示している。ただし、特徴関数生成の結果、全ての出力組合せがある場合には特徴関数の集合数の比較は行なえないため取り除いている。この結果を見てみると、出力値の集合数に制約がある場合に関して、本手法で得られた集合数が、より少なくなっていることがわかる。

回路名	集合数		
	分割なし	分割あり	
		重複なし	重複あり
controler	12228952	13355286528	629996000
ir_idc	480	63840	1760
c880	4915200	24313856	9175040
c1908	8778752	33554432	20971520
c3540	–	1179520	1131968

表 3: 集合数の比較

5 まとめ

今回は、出力変数の分割によって組合せ回路からる特徴関数を生成する手法について提案した。この方法により、真の特徴関数に近い解を持つ特徴関数を求めることが可能となる。さらに真の解に近い特徴関数を少ないBDDのノード数で生成するためには、以下の2点を考慮する必要があると考えている。

- 内部論理による分割
  正確な解を求めるためのグループ分割の基準として、現在は入力変数の共有度だけで決めているが、評価値が等しくてもその論理により関連の強さが変わってくる。このためより要素数の少ない解を得るためには、回路の論理を考慮しなくてはならない。

- 検査対象からの出力変数順序付け
  特徴関数の生成において出力変数に対しては、入力変数のような順序付けを行なっていない。しかし、最適な出力変数の順序は存在する。ノード数を小さくするためには出力変数の順序についても考えなければならない。我々は周辺回路を特徴関数生成の対象としているため、周辺回路の出力の順序は検査対象となる回路の入力から決定でき、その結果ノード数を少なくすることができると思われる。

そして、本手法で生成された特徴関数を論理検証に適用してゆくことが今後の課題となる。

参考文献

[1] R. E. Bryant :"Graph-Based Algorithms for Boolean Function Manipulation",*IEEE Trans. Comput.*, Vol. C-35, No.8, pp.677-691 (1986).

[2] O. Coudert, C. Berthet and J. C. Madre : "Verification of Synchronous Sequential Machines Based on Symbolic Execution", *LNCS 407*, Springer-Verlag (1989).

[3] H. Touati, H. Savoj, B. Lin, R. K. Brayton and A. Sangiovanni-Vincentelli : "Implicit State Enumeration of Finite State Machines using BDD's", In *Proc.ICCAD*, pp.130-133(1990).

[4] 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, No.1, pp.77-85. Jan. (1991).

[5] H. Fujii, G. Ootomo and C. Hori :"Interleaving Based Variable Ordering Methods for Ordered Binary Decision Diagrams", In *Proc. of ICCAD*,pp.38-41,Nov. (1993).

[6] 田治米, 新井, 深澤 : "周辺回路を考慮した論理検証のための特徴関数の生成について", 情報処理学会第53回全国大会, 2B-5 (1996).

[7] 越智, 澤田, 岡田, 上嶋, 神原, 濱口, 安浦 : "KUE-CHIP2 設計ドキュメント", 京都高度技術研究所 (1992).