ステム含意ノードを用いた
組合せテスト生成アルゴリズムの効率化

日大生産工(院)　○大森　悠翔　　日大生産工　細川　利典
FLEETS　　吉村　正義　　明大　　山崎　浩二

1　はじめに

回路の高集積化が進むにつれて，テスト生成1)の対象となる回路の規模がますます大きくなる傾向にあり，それとともにテスト生成に要する計算費用が膨大なものとなってきている．多くのテスト生成アルゴリズムは，解の探索のために決定木1）を用い，解が見つかるか探索空間が空になるまで探索を行う．決定木には決定点1）と呼ばれる信号線1）が積まれ，０か１の値を割当てる選択が枝になる．もし，決定処理で悪い決定点を選択すると，バックトラック1）が頻繁に発生してしまう．バックトラックが頻繁に発生すると，テスト生成に時間がかかるという問題がある．　この問題を解決するために含意ノードと呼ばれる決定点の探索手段が提案されている．ここで我々は，含意ノードの発見アルゴリズムによってテスト生成結果1)に変化が起こることに気づいた．

本稿では，幅優先探索による含意ノードの発見アルゴリズムを提案し，その効果を実験で示す．また，新たにより多くの含意を引き起こすようなステム含意ノードを提案する．

2　決定点

決定点は図1で示すように，次の2つのタイプに分けることができる．

Type-A:
　決定点が外部入力のみで構成される．この決定点によって前方含意が発生する．

Type-B:
　外部入力を含むすべての信号線が決定点の候補となる．前方含意だけでなく後方含意も発生する．

多くのテスト生成アルゴリズムでType-Aの決定点が利用されている．Type-AはType-Bに比べて探索空間が小さくなる．Type-Bは容易に冗長故障を見つけることができる.しかし，内部信号線が決定点になるため探索空間が増加する．すると，バックトラック数が増大する危険性がある．

テスト生成問題は，含意操作1）によって値が割り当てられる信号線が多いほど探索空間の枝刈りにつながる性質がある．

つまり，より多くの含意を引き起こす決定点を見つけることがテスト生成の効率化につながるということになる．

もし，複数のType-Aの決定点を被覆するような決定点を見つけることができれば，前方含意と後方含意が発生しより多くの含意が得られる．その上Type-Aの決定点よりも探索空間が少なくなる．そのような決定点を含意ノード2)と呼ぶ.

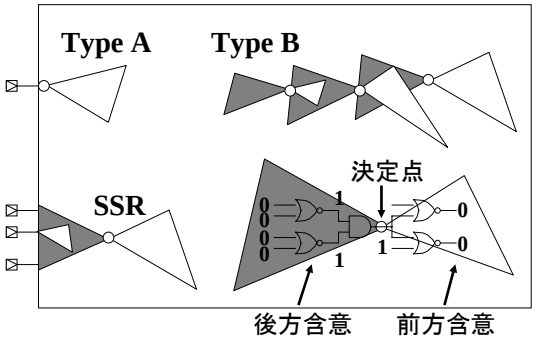

図1　決定点

3　含意ノード

3.1.　含意ノード定義

次に含意ノードの定義を示す.

- 含意ノードの信号線は故障1) の影響を受けない
- 少なくとも1つの外部入力によって含意ノードの割当目標がみたされる

図2に含意ノードの例を示す．図2の信号線Ａ，Ｂは外部入力である．図2中の灰色の領域は回路が存在しているとする．この例では故障の影響を受けない信号線Ｇに論理値１を割当てる目標に対する後方追跡を考える．ただし，信号線Ｄにはすでに論理値０が割り当てられているとする．もし信号線Ｇに値１を割り当てると含意操作により外部入力線Ａ，Ｂの値が一意的に論理値０に決まる．

On The Acceleration of Test Generation Using Fanout Stem Implication Node
Yuushou OOMORI, Toshinori HOSOKAWA,
Masayosi YOSHIMURA, Kouji YAMAZAKI

もし信号線Gに値１を割り当てると含意操作により外部入力線A，Bの値が一意的に論理値０に決まる．ここで定義より，信号線Gに１を割当てる目標は外部入力A，Bに０を割り当てることによってみたされるため信号線Gが含意ノードとなる．

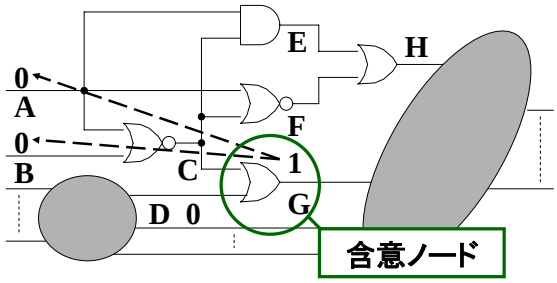

図2　含意ノード例

3.2.　探索空間の減少

定義より含意ノードは少なくとも１つの外部入力を含意することになる.図2の例では含意ノードGに論理値１を割当てることによってA＝０，B＝０の含意が発生する．ここで図2を例にとって決定木に積まれる決定点を考える.図3の(1)は含意ノードGを決定木に積む場合を示している．図3の(2)はGに値１を割当てる目標に対してType-Aの決定点A，Bを決定木に積む場合を示している.図2の例と図3からわかるように決定木に含意ノードGを積むことと決定点A，Bを決定木に積むことは効果が同じである．しかし，含意ノードを用いた場合は探索空間が小さくなる．

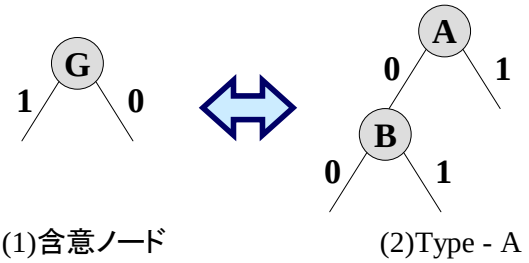

図3　探索空間の減少

3.3.　早期の矛盾発見

含意ノードを用いることによって早期に矛盾1)を発見することができる.図4に早期の矛盾発見の例を示す．すでにH＝１，B＝０，D＝０が割り当てられているとする，ここで信号線Gに論理値１を割り当てる目標の後方追跡1)を考える．定義より信号線Gは含意ノードとなる．G＝１から含意操作を行うと，A＝０，C＝１，E＝０，F＝０，H＝０となる．ところが，すでにH＝１が割り当てられているため信号線Hで矛盾が発生する．

テスト生成ではこの時点で矛盾を発見し，信号線Gに値０を割当てるためにバックトラックする．例ではH＝１の正当化をせずに矛盾を発見できている．

このことから決定木に積む決定点の数が減ることがわかる．

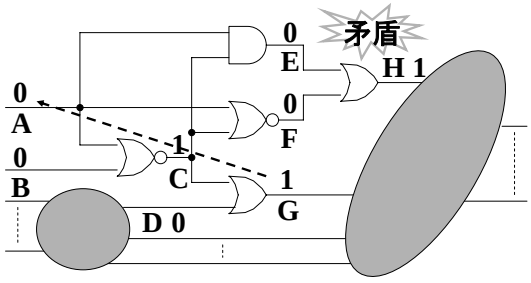

図4　早期の矛盾発見

3.4.　含意される信号線の増加

含意ノードを決定点とすることによって１つの決定点あたりの含意される信号線が増えることを図5の例で示す.図5の信号線Aは外部入力とする.含意ノードDに論理値１を割り当てる状況を考える．D＝１から含意操作を行うと，後方含意操作によりB＝０，C＝０，A＝１となる．B＝０,C＝０の含意よりさらに前方含意が発生する．

もし,従来どおりに含意ノードを利用せずにD＝１からの後方追跡でA＝１の決定点を得て,含意操作をおこなうとA＝１しか含意されない．

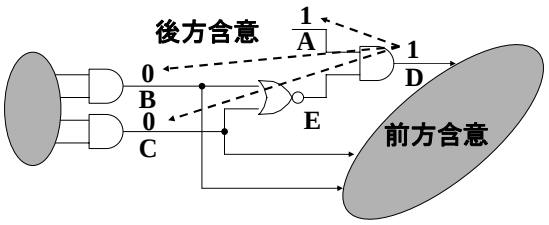

図5　含意される信号線の増加

3.5.　含意ノード発見アルゴリズム

```
while(objectives > 0){
  select objective line K that is the deepest level
        from PI
  if(K cannot be affected by the injected fault){
        if(requested_0(K) > requested_1(K)){
                Trace_unique_path(K,0);
        }
        else{
                Trace_unique_path(K,1);
        }
  }
  else{
        Multiple_backtrace(K);
  }
}
```

```
Trace_unique_path(K,L){
  if(K is primary input){
    return(find_implying_node);
  }
  if(controlling level(V) is required to
        complete K=L){
    if(there is only a line(N) whose value is
        unknown){
      compute requested_0(N) and
        requested_1(N);
      Trace_unique_path(N,V);
    }
    else{
      select a input(N) whose value is
        unknown;
      add N to objective stack;
      compute requested_0(N) and
        requested_1(N)
      return(continue);
    }
  }
  else{
    for every line(N) whose value in
        unknown compute requested_0(N)
        and requested_1(N){
      V = non controlling level
      Trace_unique_path(N,V);
    }
  }
}
```

図6　含意ノード発見アルゴリズム

図6に含意ノード発見アルゴリズムを示す. 含意ノード発見処理は後方追跡の一部の処理として考えられる．含意ノードの候補となる信号線が故障の影響を受ける場合は従来の後方追跡を行う．探索処理は深さ優先探索によって進み，最初に見つけた外部入力を含意するような信号線が含意ノードとなる.

4　含意ノードの幅優先探索

図6で示した含意ノードの発見アルゴリズムは再帰的に外部入力に到達するまで含意ノードの探索を行う.

探索アルゴリズムは様々な方法が考えられるが，今回幅優先探索による含意ノードの発見アルゴリズムを提案する.

後方追跡のアルゴリズムの構造は多重後方追跡3)と似ている．まず初期目標群3)に対して，多重後方追跡を行う．もし，現在目標の信号線が分岐信号線だった場合ステム目標群3)に追加する．初期目標群が空になったらステム目標群から目標を取り出す．取り出したステム目標は現在目標群に追加し，後方追跡処理を継続する．以上の操作を繰り返し，最初にみつけた外部入力を含意するような信号線が含意ノードとなる.

5　ステム含意ノード

5.1.　ステム含意ノード定義

含意ノードは外部入力を含意するような，決定点であるが，ある部分回路の出力が外部入力とみなせるような分岐信号線をステム含意ノードと定義する.

ここでは，部分回路の出力が外部入力とみなせるような信号線とは先頭信号線 3)や基礎ノード4)とする.

5.2.　先頭信号線

図7に先頭信号線3)を示す．図7はある回路の部分回路を表していて，入力は全て外部入力とする.図7のような樹枝状回路では分岐がないため，樹枝状回路の出力とみなせる信号線の値が決まればバックトラック無しで外部入力の値が決まる．そのような信号線を先頭信号線という.

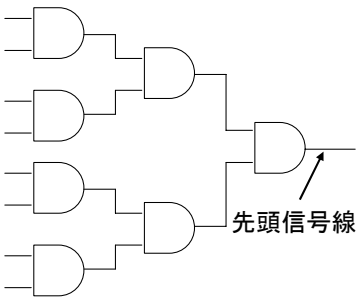

図7　先頭信号線

5.3.　基礎ノード

図8に基礎ノード4)を示す.図8はある回路の部分回路を表していて，入力は全て外部入力とする．図8の部分回路では分岐が存在しているが，この部分回路の出力とみなせる信号線の値が決まれば外部入力の値もバックトラック無しで決まる.このように回路の構造的に先頭信号線と同じ効果がある信号線を基礎ノードという.

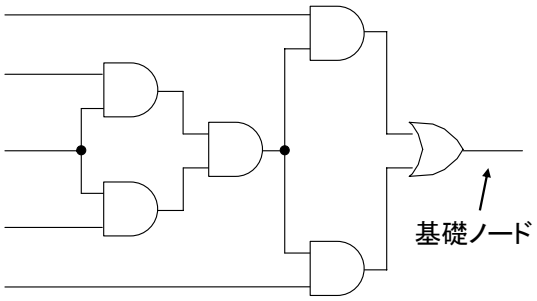

図8　基礎ノード

5.4.　探索空間の減少

先頭信号線,基礎ノードともにその信号線の値が決まれば外部入力の値がきまるという性質を持っている．さらに，それらを含意するステム含意ノードを決定点とすることによって含意ノードを用いるよりもさらに探索空間を小さくすることができる.

6 実験結果

SPOPアルゴリズム5)とFANアルゴリズム3)の実装を行った．後方追跡のアルゴリズムとして，多重後方追跡,深さ優先探索の含意ノードを用いた後方追跡,幅優先探索の含意ノードを用いた後方追跡を実装した．実験はISCAS85ベンチマーク回路に対して行った.バックトラックリミットは100回とする．実装はＣ言語で行った．マシンのＯＳはWindowsXP，CPUは2.0Ghz，メモリは2GBである.

表1にSPOPアルゴリズムの実験結果を示す.表1は各回路に対して上から多重後方追跡,含意ノードの深さ優先探索,含意ノードの幅優先探索を適用した場合のテスト生成結果を示している.

表1　テスト生成結果(SPOP)

回路名	故障検出効率[%]	打切り故障数	冗長故障数	バックトラック数	CPU時間[sec.]	後方追跡
c1355	91.49	134	8	16352	18.25	多重後方追跡
	99.87	2	8	2380	3.91	含意ノード(深)
	100.00	0	8	199	5.53	含意ノード(幅)
c1908	99.89	2	9	403	1.98	多重後方追跡
	100.00	0	9	20	1.62	含意ノード(深)
	100.00	0	9	64	1.92	含意ノード(幅)
c2670	100.00	0	117	255	1.94	多重後方追跡
	100.00	0	117	264	1.98	含意ノード(深)
	99.85	4	113	963	2.72	含意ノード(幅)
c3540	100.00	0	137	205	4.95	多重後方追跡
	100.00	0	137	250	5.11	含意ノード(深)
	100.00	0	137	210	5.30	含意ノード(幅)
c5315	100.00	0	59	238	3.41	多重後方追跡
	99.94	3	59	754	3.91	含意ノード(深)
	99.96	2	59	465	3.58	含意ノード(幅)
c7552	99.99	1	131	1393	12.41	多重後方追跡
	99.99	1	131	1012	10.83	含意ノード(深)
	99.96	2	131	1741	14.12	含意ノード(幅)

結果について考察してみるとc1355, c1908の回路で多重後方追跡よりも含意ノードを用いた後方追跡で打切り故障数が少ない結果となっている．このことから多重後方追跡よりも含意ノードを用いた方が良い決定点を選択していることがわかる．

特にc1355では多重後方追跡の打切り故障数が134個なのに対し，深さ優先探索の含意ノードを用いた後方追跡では打切り故障数が2個，幅優先探索の含意ノードを用いた後方追跡では打切り故障数が0個だった．故障検出効率にすると多重後方追跡よりも含意ノードを用いた後方追跡方が10%近くも良くなっている．バックトラック数を見てみると幅優先探索の含意ノードを用いた後方追跡のバックトラックがかなり少ないことがわかる．このことからc1355は内部信号線を決定点にするよりも外部入力に近い信号線を決定点にする方がテストパターンを見つけやすいことがわかる．

c5315では多重後方追跡の打切り故障数が少ない結果となっているが，内部信号線を決定点とする方がテストパターンを見つけやすい性質ではないかと推測する．

SPOPアルゴリズムで，含意ノードの効果が最も良かったc1355の回路に対してFANアルゴリズムでも実験を行った．表2にc1355のFANアルゴリズムの実験結果を示す.FANアルゴリズムもSPOPアルゴリズム同様多重後方追跡よりも含意ノードを用いた後方追跡で打切り故障数が少ない結果となっている．

しかし，深さ優先探索の含意ノードを用いた後方追跡と幅優先探索の含意ノードを用いた後方追跡では結果が異なっている．この原因は現状ではわからない．

表2　c1355テスト生成結果(FAN)

回路名	故障検出効率[%]	打切り故障数	冗長故障数	バックトラック数	CPU時間[sec.]	後方追跡
c1355	94.66	84	8	8837	8.64	多重後方追跡
	100.00	0	8	2682	3.39	含意ノード(深)
	99.43	9	8	2170	5.01	含意ノード(幅)

6 おわりに

本稿では，新たに幅優先探索を用いた含意ノードの発見アルゴリズムを示し，その効果を検証した.結果よりc1355で大きな効果を得たが，回路によっては良い結果が得られなかった．このことより決定点とする信号線によってテスト生成結果に影響があることがわかる．

今後，ステム含意ノードを用いた後方追跡の実装を行いその効果を検証する．さらにc1355のSPOPアルゴリズムとFANアルゴリズムで含意ノードの探索方法の違いがテスト生成にどのような影響を及ぼすのかを解析する．

「参考文献」

1) MIRON Abramovici , Melvin A.Breuer , Arthur D.Friedman “DIGITAL SYSTEMS TESTING AND TESTABLE DESIGN”,1995.

2) Mituo Teramoto, ”A Method for Reducing the Search Space in Test Pattern Generation “ INTERNATIONAL TEST CONFERENCE,pp.429-435,1993.

3) Fujiwara “on the Acceleration of Test Generation Algorithms ” , IEEE Trans,pp.1137-1144,1983.

4) T.Kirkland M.R.Mercer,”A Topological Search Algorithm for ATPG,” in Proc. 24th Design Automation Conf.June,pp.502-508,1987.

5) M.Henftling H.C.Wittmann K.J.Antreich “A Single-Path-Oriented Fault-Effect Propagation in Digital Circuits Considering Multiple-Path Sensitization”,ICCAD’95,pp.304-309,1995 .