回路のエラー確率を効率的に計算するための決定図

A Decision Diagram to Analyze Probabilistic Behavior of Circuits

阿部 航大 † 山下 茂 ‡
Kodai Abe Shigeru Yamashita

1. はじめに

集積回路の縮小や動作電圧の低下に伴い，放射線がゲートに与える影響は無視できないものとなっている．この影響によって一時的に発生するエラーは，ソフトエラーと呼ばれる [1]. 近年，ソフトエラーの発生が顕著になってきたことで，このエラーの発生を考慮した設計が求められている．そのため，回路の正確な信頼性を，できるだけ効率的に評価する手法が必要となっている．

ソフトエラーの発生は，必ずしも回路の外部出力に影響を与えるとは限らない．例えば，2 入力の AND ゲートに $(00)_2$ の入力を与える時，二つのうち一つの入力が反転したとしても，ソフトエラーを検出することができない．このような，ゲートの論理によってソフトエラーの伝搬を防ぐ効果を論理マスクと呼ぶ．一方で，入力が $(11)_2$ の時には，正しい出力は 1 であるため，どちらの入力も反転することは許されない．この例のように，論理マスクが発生するかどうかは，ゲートの論理とその入力によって決定される．したがって，回路の出力が最終的に 1 となる確率は，ゲートの組み合わせとその回路に与えられる入力に依存する．すなわち，回路の入力数 n に対して，2^n パターンの場合における，回路の故障確率を考える必要がある．そのため，n 入力の回路を評価する際には，2^n パターンについての信頼性を考慮しなくてはならない．このような計算は n が大きくなると困難であるため，通常はモンテカルロ法を利用したフォールト挿入テストが用いられる．あるいは解析的な手法として，Probabilistic Transfer Matrix (PTM) [2] を利用した手法が用いられる．そして，PTM を改善した手法として，数式でエラーの発生する確率を表現することにより，空間計算量を削減した手法も存在する [4].

数式で確率を表現することにより空間計算量を削減した評価手法では，PTM の最終的な評価式を変形し，各入力に対する出力が 1 になる確率を，有向非巡回グラフを用いて計算する．この手法では，回路をファンアウト・ポイントで分割し，一つの部分回路に注目し，評価式の計算を行う．このとき，評価式の計算を行う際，特殊な演算規則を適用することで，再収斂に対する問題を回避している．しかしながら，この手法が用いる数式処理では，全入力パターンに対して各ゲートの出力が 1 になる確率を一つずつ計算する．よって，大規模回路の場合，効率的な処理が実現できない．

そこで本論文では，大規模回路の効率的な評価式の計算に，適した数式処理システムを提案する．提案手法では，各入力に対する出力が 1 になる確率を，BDD(Binary Decision Diagram) と類似のデータ構造を用いて計算する [3]．このデータ構造を用いることで，大規模回路のように計算量が多い場合でも，高速に計算することができる．

実験では，数式で確率を表現することにより空間計算量を削減した評価手法と比べ，計算速度を高速化することに成功した．

以下では，まず，第 2 章で既存手法として数式で確率を表現する評価手法について解説する．次に，第 3 章で BDD を用いた数式処理システムについて提案する．その後，第 4 章で実験と考察を述べ，第 5 章で本研究のまとめと今後の課題について述べる．

2. 準備

2.1 数式で出力が 1 になる確率を評価する手法

PTM は，ゲートや回路のある入力に対し，ある出力となる確率を表す行列である．PTM を用いた評価手法は，行列同士の乗算を行うため，行列の大きさが問題となる．そこで，PTM を改善した手法では行列を用いずに PTM を用いた手法と等価な評価値を得る．また，この手法では，ゲートの出力が反転する確率とゲートの入力が反転する確率から各ゲートの出力が 1 になる確率を評価している．

2.2 正しい確率が得られないケース

ゲートの出力が 1 になる確率を評価するだけでは，回路に再収斂を含む場合において正確な結果を得ることができない．回路に再収斂が含まれている場合，計算結果には，同一のゲートに対して異なるソフトエラー発生の状態を仮定した計算が含まれてしまうからである．例えば，図 1 のようなケースにおいて，g_1 から分岐した出力は g_4 へ 2 通りの経路から入力される．

この時，ゲート g_4 では，ゲート g_1 でエラーが発生する仮定と発生しない仮定の 2 つの矛盾した仮定を同時

† 立命館大学大学院, Graduate School of Information Science and Engineering, Ritsumeikan University
‡ 立命館大学, Ritsumeikan University

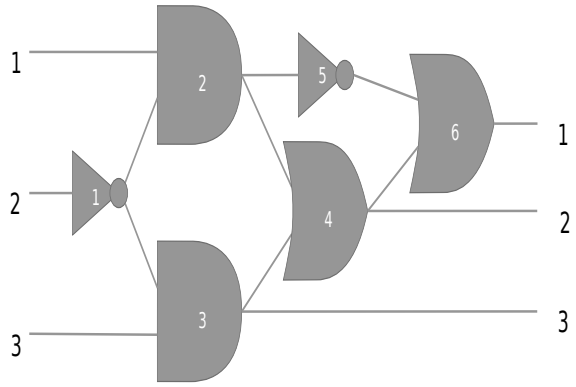

図 1: 再収斂によって正確な結果が得られない例

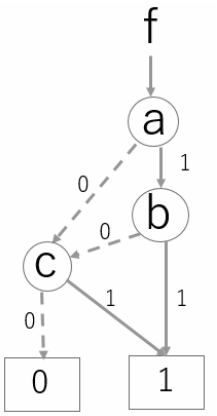

図 2: 関数 $f = ab + c$ を表す BDD

に用いることがあるため，誤った値を計算する可能性がある．したがって，g_4 での計算は，g_2 の計算で g_1 にエラーが発生しないと仮定したにもかかわらず，g_4 の計算で g_1 にエラーが発生すると仮定して計算した場合を含む．ゲート g_2 の分岐でも，同様に矛盾した仮定を用いて計算する場合がある．

2.3 Binary Decision Diagram

ここでは，第 3 章の提案手法で参考とする BDD(Binary Decision Diagram) について説明する．BDD とは，論理関数を表現するデータ構造の一種であり，大きな特徴を 2 つもつ．1 つは，カノニカル性を持つことである．カノニカルとは，同一の論理関数を表す BDD は唯一であることを意味する．もう 1 つは，組み合わせ爆発を起こすような論理関数の集合を，効率よく圧縮し表現できることである．以下では，具体的なデータ構造や構築方法などについて説明する．

2.3.1 基本的なデータ構造

BDD の簡単な例を，図 2 に示す．図 2 は関数 $f = ab+c$ を表しており，ある入力パターンに対して，決められた出力をすることを表現している．例えば，$a = 0$ のとき，a のノードにおける，0 の枝 (以下 F 枝) を辿り次のノードの値を参照する．この場合，関数 f の出力に影響を与えるノードは c のみなので，ノード b の値は参照しなくてよい．つまり，$a = 0$ のとき，$c = 0$ であるならば関数 f の出力は 0 になり，$c = 1$ であるならば関数 f の出力は 1 になる．また，$a = 1$ のとき，a における 1 の枝 (以下 T 枝) を辿る．このとき，b は関数 f の出力に影響を与えるので，この場合はノード b の値を参照しなくてはならない．このように，各ノードの値によって枝を辿り，関数の最終的な出力が 0 か 1 に定まるデータ構造となっている．

2.3.2 BDD の構築方法

BDD を構築するには，変数の順序付けを行い，論理式から生成する方法がよく用いられる．具体的には，2 つの BDD の間で 2 項演算を繰り返して任意の BDD を生成する．図 2 のような BDD を生成するには，まず初めに，$a \prec b \prec c$ と順序付けを行う．次に，BDD(a) と BDD(b) の AND 演算をし，BDD(ab) を生成する．最後に，(c) との OR 演算を行うことで，BDD(ab+c) を生成する．2 項演算の具体的なアルゴリズムは式 (1) のようになる．

$$f < op > g = v(f_v < op > g_v) + v'(f'_v < op > g'_v) \quad (1)$$

$< op >$ は AND 演算や OR 演算を表す．また，v は変数を表し，変数の順序付けに基づいて再帰的に展開していく．図 2 のような，順序付けが行われた BDD を順序付き BDD と呼ぶ．BDD は変数の順序付けによってノード数が変化する場合があるため，順序の考慮が重要である．

2.3.3 BDD 処理系で用いられる技法

- BDD の共有化
  BDD の共有化とは，複数の BDD で部分的にノードを共有する技法である．例えば，BDD(ab+c) と BDD(ab+d) の二つの BDD が存在する場合，ab の部分を共有することができる．このとき，2 つの BDD を合わせた場合のノード数に比べて，ノード数は減少している．これは，BDD のノード数が増加するほど，共有できるノードが増えるので，有効な技法である．

- 否定枝
  否定枝とは，BDD の枝に否定の属性を持たせることで，否定の関係にある BDD を共有化する技法で

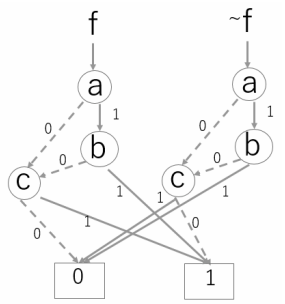

図 3: 関数 $f = ab + c$ とその否定を表す BDD

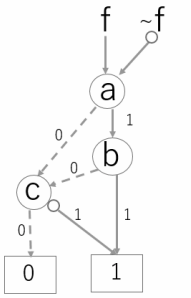

図 4: 否定枝を適用した BDD

ある．また，カノニカル性を保つために，否定を持たせる枝は T 枝のみといったルールや，1 の定数ノードは使用しないといったルールが存在する．例として，否定枝を適用する前の BDD を図 3 とする．NOT を表す BDD の生成方法は，BDD を複製して 0 と 1 の定数ノードを交換するだけで良い．図 3 に対して否定枝を適用し，共有化した BDD は図 4 のようになる．枝に否定の属性を持たせることで，グラフが圧縮されたのがわかる．また，NOT 演算はグラフの根の枝の否定枝を切り替えるだけで良いので，定数時間で実行することが可能である．

3. 決定図を用いた数式処理システム

3.1 Binary Decision Diagram for Probabilities

本研究では，回路やゲートの出力が 1 となる確率を計算するための，BDD と類似のデータ構造を提案する，これを Binary Decision Diagram for Probabilities (以下 BDDP) と呼ぶ．BDDP は BDD と似たデータ構造であるが，異なる点がいくつかある．1 つは，BDDP は回路やゲートの出力が 1 となる確率の式を表すという点である．もう 1 つは，終端ノードが 1 しか存在しない点である．BDDP を F とし，その F の定義式を式 (2) に示す．また，その図を図 7 に示す．

$$F = P_F f_F + P_T f_T x \quad (2)$$

ここで P_F は F 枝の値，P_T は T 枝の値である．また，f_F は F 枝が指す BDDP，f_T は T 枝が指す BDDP である．BDDP の例を，図 5 に示す．図 5 の BDDP は，$f = 0.6a + 0.2$ という数式を表現する．このとき，$f = 0.6a + 0.2$ は 1 を出力する確率を表す数式となる．変数 a の係数をノード a の T 枝で保持し，変数 a がない項はノード a の F 枝で保持する．このように，枝に値を持たせることで，数式を表現する．また，BDDP における否定枝は，図 6 のように ◯ で表し，否定枝の持つ意味を式 (3) に示す．これは BDDP における NOT 演算に相当する．

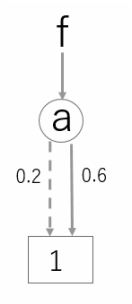

図 5: 式 $f = 0.6a + 0.2$ を表す BDDP

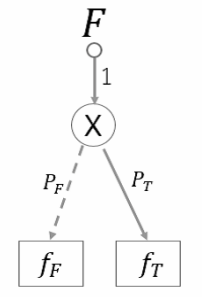

図 6: 否定枝を持つ BDDP

$$F = 1 - (P_F f_F + P_T f_T x) \quad (3)$$

3.2 BDDP の正規化

ここでは，BDDP における演算を行う上での正規化ルールについて説明する．正規化を行う理由は，2.2 節で説明したカノニカル性を保つためである．1 つ目の正規化ルールは，F 枝の値を 1 とすることである．F 枝の値を 1 とするために，式 (2) の右辺を P_F で括り，f_F の係数を 1 にする．その結果，正規化された式は式 (4) となり，図 8 のようになる．

$$\begin{aligned} F &= P_F f_F + P_T f_T x \\ &= P_F(f_F + \frac{P_T}{P_F} f_T x) \end{aligned} \quad (4)$$

次に，2 つ目の正規化ルールについて説明する，2 つ目の正規化ルールは，T 枝の値を正とすることである．式 (5) は，F における T 枝の値が負の場合を表し，図 9 のように表せる．また，式 (6) は，式 (5) を正規化した式であ

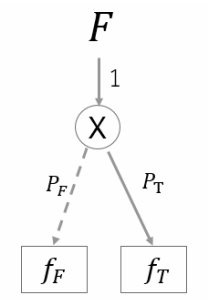

図 7: F 枝について正規化する前の BDDP

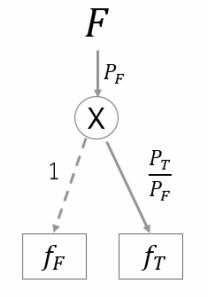

図 8: F 枝について正規化された BDDP

り，否定枝を用いて図 10 のように表せる

$$F = P_F(-\frac{P_T}{P_F}f_T x + f_F) \quad (5)$$

$$= \left(1 - P_F\left(\frac{P_T}{P_F}f_T x + (1 - f_F)\right)\right) \quad (6)$$

3.3 BDDP の演算

以下では，式 (4) のように正規化された BDDP を F と G とし，それぞれを用いて BDDP の演算を説明する．

$$F = P_F(\alpha f_{F1}x + f_{F0})$$

$$G = P_G(\beta f_{G1}x + f_{G0})$$

3.3.1 AND 演算

BDDP における演算の一つである AND 演算を F と G を用いて説明する．まず初めに，式 (7) のように，それぞれの要素を掛け合わせる．

$$AND(F,G) = P_F P_G\left(\alpha f_{F1}x + f_{F0}\right)\left(\beta f_{G1}x + f_{G0}\right) \quad (7)$$

次に，式 (7) を展開し，x の係数について考える．x の係数を A とおき，式 (8) に示す．

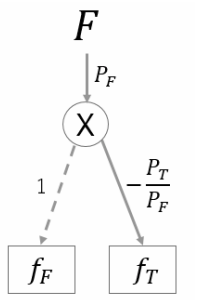

図 9: T 枝の値について正規化する前の BDDP

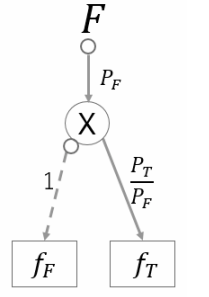

図 10: T 枝の値について正規化された BDDP

$$\begin{aligned} A &= \alpha\beta \cdot AND(f_{F1}, f_{G1}) + \alpha \cdot AND(f_{F1}, f_{G0}) \\ &\quad + \beta \cdot AND(f_{F0}, f_{G1}) \\ &= OR(OR(\alpha\beta \cdot AND\left(f_{F1}, f_{G1}\right), \alpha \cdot AND\left(f_{F1}, f_{G0}\right)), \\ &\quad \beta \cdot AND\left(f_{F0}, f_{G1}\right)) \end{aligned} \quad (8)$$

x を含まない項は B とおき，式 (9) に示す．

$$B = AND(f_{F0}, f_{G0}) \quad (9)$$

このとき，A と B の係数をそれぞれ γ, δ とすると，BDDP における AND 演算は，式 (10) となる．AND 演算の実行結果を図 11 に示す．

$$\begin{aligned} & AND(F,G) \\ &= P_F P_G(\gamma A x + \delta B) \\ &= \delta P_F P_G\left(\frac{\gamma}{\delta}Ax + B)\right) \end{aligned} \quad (10)$$

3.3.2 OR 演算

BDDP における OR 演算を，F と G を用いて説明する．前述した AND 演算では，出力が 1 となる確率における数式同士の積を計算するだけで，正規化された BDDP

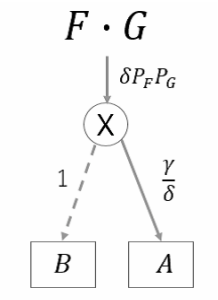

図 11: BDDP における AND 演算の実行結果

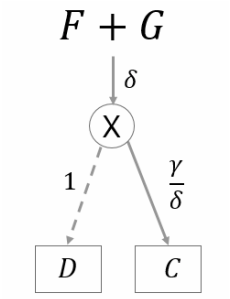

図 12: BDDP における OR 演算の実行結果

を求めている．しかし，OR 演算において出力が 1 となる確率を積だけで求めることはできない．まず初めに，出力が 1 になるパターンについて考える．F と G の 2 つの BDDP に対して，OR 演算を行うと，出力が 1 となる確率は式 (11) となる．また，0 を出力する確率を表す BDDP は，それぞれ $1-F$，$1-G$ となる．

$$\begin{aligned} & F\cdot G+(1-F)\cdot G+F\cdot(1-G) \\ = \;& F+G-F\cdot G \end{aligned} \tag{11}$$

つまり，出力が 1 となる確率は，BDDP 同士の和から積を減算することで求まるため，式 (12) となる．

$$OR(F,G)-AND(F,G) \tag{12}$$

式 (12) の OR 演算の部分を式 (13) に示し，AND 演算の部分を式 (14) に示す．

$$\left(P_F\alpha f_{F1}+P_G\beta f_{G1}\right)x+\left(P_F f_{F0}+P_G f_{G0}\right) \tag{13}$$

$$-P_FP_G((\alpha f_{F1}\beta f_{G1}+\alpha f_{F1}f_{G0}+f_{F0}\beta f_{G1})\,x+f_{F0}f_{G0}) \tag{14}$$

x の係数を C とおき，式 (13) と式 (14) における x の係数部分をまとめ，式 (15) に示す．

$$\begin{aligned} C \;=\;& P_F\alpha f_{F1}+P_G\beta f_{G1}-P_FP_G\alpha f_{F1}\beta f_{G1} \\ & -P_FP_G\alpha f_{F1}f_{G0}-P_FP_Gf_{F0}\beta f_{G1} \\ =\;& OR\left(P_F\alpha f_{F1},P_G\beta f_{G1}\right)+AND\left(-P_F\alpha f_{F1},P_G\beta f_{G1}\right) \\ & +AND\left(-P_F\alpha f_{F1},P_Gf_{G0}\right)+AND\left(P_FF_{F0},-P_G\beta f_{G1}\right) \\ =\;& OR(OR(OR(OR(P_F\alpha f_{F1},P_G\beta f_{G1}), \\ & AND(-P_F\alpha f_{F1},P_G\beta f_{G1})), \\ & AND(-P_F\alpha f_{F1},P_Gf_{G0})), \\ & AND(P_FF_{F0},-P_G\beta f_{G1})) \end{aligned} \tag{15}$$

x を含まない項を D とおき，式 (16) に示す．

$$\begin{aligned} D \;=\;& P_Ff_{F0}+P_Gf_{G0}-P_FP_Gf_{F0}f_{G0} \\ =\;& OR(P_Ff_{F0},P_Gf_{G0})+AND(-P_Ff_{F0},P_Gf_{G0}) \end{aligned} \tag{16}$$

このとき，C と D の係数をそれぞれ γ,δ とすると，BDDP における OR 演算は，式 (17) となる．OR 演算の実行結果を図 12 に示す．

$$\begin{aligned} & OR(F,G) \\ =\;& \gamma\cdot Cx+\delta\cdot D \\ =\;& \delta\left(\frac{\gamma}{\delta}Cx+D)\right) \end{aligned} \tag{17}$$

4. 実験と考察

4.1 実験結果

第 3 章で提案した BDDP について，プログラムを実装し，その評価実験を行った．表 1 に，数式で出力が 1 になる確率を評価する手法と計算速度を比較した結果を示す．表の各列は，以下のような意味を持つ．

回路 ベンチマーク回路のファイル名
入力 外部入力の数 $|PI|$
時間 計算時間（秒）
既存 数式で出力が 1 になる確率を評価する手法の結果
提案 BDDP を用いて出力が 1 になる確率を評価する手法の結果
比率 (既存手法 の結果)/(提案手法の結果)

現時点では，再収斂の問題に対応するまで至らなかったので，再収斂を無視して計算している．なお，0.000001 に満たない値は，0.000000 として表記する．また，比率の計算において，結果が 0.000000 の場合は，0.000001 として表している．本実験のベンチマーク回路には VERILOG で記述された組み合わせ回路を用いており， 全てのゲートを NAND ゲートとして計算している．そして，各ゲー

表 1: 既存手法との計算時間の比較

回路	入力	時間 (秒) 既存	提案	比率
C17	5	0.000000	0.000000	1.0
decod	5	0.0004	0.00074	0.5
z4ml	7	0.006	0.005021	1.1
9symml	9	0.035	0.021363	1.6
x2	10	0.006	0.004729	1.2
cu	14	0.195	0.07805	2.5
parity	16	0.309	0.169371	1.8
pcle	19	4.411	0.847329	5.2
cc	21	18.326	5.083124	3.6
mux	21	22.765	4.382017	5.1

表 2: 実験環境

CPU	Intel® Core™ i5-3570 CPU (3.40GHz)
メモリ	8 GB
OS	Linux (Fedora 20)
言語	C++
コンパイラ	g++ 4.8.3

トが反転する確率は，$err(g) = 0.05$ を設定した．提案手法の実験環境は，表 2 の通りである． 加えて，表 3 は，BDDP を用いた手法の正確性を検証した結果である．擬似乱数を用いて，100 回のシミュレーションを行った結果と，BDDP を用いて計算した確率を比較している．表 3 の値は，各入力パターンにおける，比較結果の平均を計算したものである．

4.2 考察

表 1 より，回路の規模が大きくなるほど，BDDP は高速に計算できていることがわかる．しかし，表 3 では，全体的に既存より誤差が大きくなっていることがわかる．既存手法ではほとんどのケースで 10%以内の誤差で計算できているのに対し，BDDP を用いた手法では 10%の誤差を越えるケースが多く見られる．その要因の 1 つとして，正規化の方法に乗算と除算を採用していることが考えられる．そのため，加算と減算を使う方法に変えると，誤差を小さくできると思われる．

表 3: 正確性の検証

回路	既存研究	BDDP
C17	0.034	0.057
decod	0.011	0.029
z4ml	0.006	0.048
9symml	0.010	0.086
x2	0.033	0.130
cu	0.036	0.118
parity	0.043	0.185
pcle	0.021	0.171

5. おわりに

本稿では，決定図を用いた確率計算のための数式処理システムについて提案した．提案手法では， Binary Decision Diagram (BDD) と類似のデータ構造である Binary Decision Diagram for Probabilities (BDDP) を用いて，計算時間の面で効率化を行った．既存手法で認識されていた，再収斂による矛盾した確率の計算にも対応している．

実験した結果，BDDP を用いて確率の計算を行う提案手法は，数式で確率を表現することにより空間計算量を削減した評価手法に比べて，計算時間を高速化することに成功した．再収斂に対応するまでには至らなかったが，BDDP が大規模回路の評価に適していることは示すことができた．

提案手法では，BDD と類似のデータ構造である BDDP といった確率計算に対応した決定図を用いることで，高速な信頼性の計算に成功した．しかし，誤差が既存手法に比べて大きくなってしまった．その原因の 1 つとして，正規化の方法が関係していると考えられる．乗算と除算で正規化してしまうと，正規化を崩した時に誤差が発生してしまう．これが繰り返されることにより，誤差が大きくなる．正規化の方法を加算と減算に変えることで，誤差を小さくすることができるのではないかと考えられる．

参考文献

[1] Robert Baumann. Soft errors in advanced computer systems. *IEEE Des. Test*, 22(3):258–266, May 2005.

[2] Smita Krishnaswamy, George F. Viamontes, Igor L. Markov, and John P. Hayes. Probabilistic transfer matrices in symbolic reliability analysis of logic circuits. *ACM Trans. Des. Autom. Electron. Syst.*, 13(1):8:1–8:35, February 2008.

[3] S. Minato. Implicit manipulation of polynomials using zero-suppressed bdds. In *Proceedings of the 1995 European Conference on Design and Test*, EDTC '95, pages 449–, Washington, DC, USA, 1995. IEEE Computer Society.

[4] Masatoshi Tsushima. An Efficient Calculation Method for Reliability Analysis of Logic Ciruits. Master's thesis, Ritsumeikan Univ, 2015.