システムＬＳＩ設計技術　104－18
(2002. 1. 24)

AND-EXOR 論理式最小化アルゴリズム

平山　貴司†　　西谷　泰昭†

† 岩手大学 工学部
〒 020–8551 岩手県盛岡市上田 4–3–5

E-mail: †{hirayama,nisitani}@cis.iwate-u.ac.jp

あらまし　本論文では，従来よりも高速な AND-EXOR 論理式の最小化アルゴリズムを提案する．これまで，6 変数関数の AND-EXOR 論理式について現実的な計算時間で最小化することは困難とされてきたが，本手法は，任意の 6 変数関数および一部の 7 変数関数について現実的な計算時間で最小解を求めることができる．本手法の高速化のポイントは，解の探索を効率化したことにある．解の最小性を保証しつつ探索を効率化するために，枝刈りによる無駄な探索の省略や，探索範囲の動的な評価を議論する．最後に実験結果により本手法の有効性を示す．

キーワード　AND-EXOR 二段論理回路，AND-EXOR 論理式，ESOP，論理最小化アルゴリズム

An Algorithm of Minimizing AND-EXOR Expressions

Takashi HIRAYAMA† and Yasuaki NISHITANI†

† Faculty of Engineering, Iwate University
Ueda 4–3–5, Morioka-shi, Iwate, 020–8551 Japan

E-mail: †{hirayama,nisitani}@cis.iwate-u.ac.jp

Abstract　We propose a faster algorithm of minimizing AND-EXOR expressions. While it has been considered difficult to obtain the minimum AND-EXOR expression of a given function with 6 variables in a practical computing time, our algorithm can compute the minimum AND-EXOR expressions of any 6-variable and some 7-variable functions practically. In this paper, how to prune the branches in the search process and how to evaluate the search area for the minimum solutions are discussed as the key point of reduction of the computing time. Experimental results to demonstrate the efficiency are also presented.

Key words　AND-EXOR two-level circuit, AND-EXOR expression, exclusive-or sum-of-products expression, logic minimization algorithm

1. はじめに

論理回路の設計は，AND, OR, 及びNOTを基本として行われるが，排他的論理和(EXOR)ゲートを併用すると回路のゲート数を削減できるものが多い[1], [2], [4]．AND-OR回路と比較したとき，AND-EXOR回路は，算術演算回路，誤り訂正回路，通信回路などにおいて，特にゲート数が少なくてすむ[7].

EXORを用いた論理回路の基礎研究として，AND-EXOR二段論理回路およびAND-EXOR論理式が研究されている．AND-EXOR論理式にはReed-Muller論理式など多くの種類がある[10]が，最も一般化されたAND-EXOR論理式がESOP(Exclusive-or Sum Of Products)である．これは，任意の積項をEXORで結合した論理式であり，その最小積項数は，どの種類のAND-EXOR論理式よりも小さくなる．論理関数fをESOPで表現したときに，積項数が最小になるESOPを関数fの最小ESOPといい，その積項数を$\tau(f)$で表す．また，最小ESOPを求めることを最小化といい，最小性は保証されないが積項が少ないESOPを求めることを簡単化という．

近年ESOPの簡単化アルゴリズムが多数提案されており，現実的な計算時間で最小ESOPに近いESOPが得られるようになってきている[8], [11]．しかしながら，これらの多くはヒューリスティックな論理変換により論理回路を簡単化するアルゴリズムであり，簡単化された結果の最小性は保証されない．一方，ESOPの最小化アルゴリズムの研究は少なく，効率の良い最小化アルゴリズムは知られていない．最小化定理[3]に基づいた単純な最小化アルゴリズムでは，与えられた関数fの変数の個数をnとしたとき，すべての$(n-1)$変数関数gについて調べれば，fの最小ESOPを求めることができる．しかしながら，$(n-1)$変数関数の個数は$2^{2^{n-1}}$と膨大であるため，nが6以上になると現実的でない．実際，これまでの最小化アルゴリズム[3], [6], [9]では，5変数以下の関数の最小解は容易に計算できたが，6変数以上の関数については現実的な計算時間で最小ESOPを計算することは困難であった．

本論文では，従来よりも高速なESOP最小化アルゴリズムを提案する．本手法は，任意の6変数関数および$\tau(f) \leqq 9$程度の7変数関数について現実的な計算時間で最小ESOPを求めることができる．まず，2節で，本手法の基本となる簡単化アルゴリズム*naive-tau*$[k](f)$を示す．これは，すべての$(n-1)$変数関数について調べるかわりに，非負整数kについて$\tau(g) \leqq k$を満たす$(n-1)$変数関数gについてのみ調べるアルゴリズムである．*naive-tau*$[k](f)$が返すESOPの積項数を$\tau[k](f)$で表す．*naive-tau*$[k](f)$は，$\tau(g) \leqq k$を満たすすべてのgについて調べることから，探索プログラムと見ることができる．3節では，*naive-tau*$[k](f)$に枝刈りを追加し，探索を高速化したアルゴリズム*fast-tau*$[k](f)$を示す．この枝刈りは，解の精度に影響しないため，*fast-tau*$[k](f)$も$\tau[k](f)$を返す．*fast-tau*$[k](f)$により$\tau(f)$(最小ESOP)を求めるには，kの値を十分大きくして探索範囲を広げればよい．このとき，kの上界が評価できれば，探索の効率化につながる．そこで，4節では，$\tau(f) = \tau[k](f)$となるkの上界を示す．そのようなkについて*fast-tau*$[k](f)$を実行することで，最小ESOPを求めることができる．5節では，本手法による探索の削減効果を調べた実験結果を示す．

2. 準　　備

本節では，ESOP最小化に関する定義，定理を述べ，最小化アルゴリズムの基本概念を与える．

[定義1] n変数関数fと変数xに対して，$x = 0$と制限したときのfの部分関数を$f_{x:\{0\}}$と表し，$x = 1$と制限したときの部分関数を$f_{x:\{1\}}$と表す．更に，任意の$I \subseteqq \{0,1\}$，$J \subseteqq \{0,1\}$に対して，$I \oplus J$で集合I，Jの排他的和集合$(I \cup J) - (I \cap J)$を表すとし，$f_{x:(I \oplus J)} = f_{x:I} \oplus f_{x:J}$と定義する．すなわち，$f_{x:\{0,1\}} = f_{x:\{0\}} \oplus f_{x:\{1\}}$，$f_{x:\{\}} = 0$と定義する．論理関数$f$をESOPで表現したときに，積項数が最小になるESOPを関数fの最小ESOPといい，その積項数を$\tau(f)$で表す．

上記の表記を用いて，任意の関数fは以下のように展開できる．ここで，$I \subseteqq \{0,1\}$である．

$$f = \bar{x} f_{x:(I \oplus \{1\})} \oplus x f_{x:(I \oplus \{0\})} \oplus f_{x:(I \oplus \{0,1\})}$$

$I = \{0,1\}$の展開はシャノン展開に対応し，$I = \{0\}$，$I = \{1\}$の展開は，それぞれ負極性ダビオ展開，正極性ダビオ展開に対応する．

[定義2] n変数関数f，$(n-1)$変数関数g，添字$I \subseteqq \{0,1\}$に対して次のように関数$T_{x:I}$，Tを定義する．

$$\begin{aligned} &T_{x:I}(f,g) \\ &= \tau(f_{x:(I \oplus \{0\})} \oplus g) + \tau(f_{x:(I \oplus \{1\})} \oplus g) \\ &\quad + \tau(f_{x:(I \oplus \{0,1\})} \oplus g) \\ &T(f,g) \\ &= \min\{T_{x:\{0\}}(f,g), T_{x:\{1\}}(f,g), T_{x:\{0,1\}}(f,g)\} \end{aligned}$$

$T_{x:\{\}}(f,g)$ $(I = \emptyset)$ は以下の最小化アルゴリズムでは必要ではないため，$T(f,g)$ は $T_{x:\{\}}(f,g)$ を調べない．

[補題 1] f を n 変数関数，g を $(n-1)$ 変数関数とする．任意の添字 I，$J \subseteq \{0,1\}$ に対して，次の式が成り立つ．

$$T_{x:I}(f,g) = T_{x:(I\oplus J)}(f, f_{x:J} \oplus g)$$

(証明)

$$\begin{aligned}
&T_{x:J}(f, f_{x:(I\oplus J)} \oplus g) \\
&= \tau(f_{x:(J\oplus\{0\})} \oplus f_{x:(I\oplus J)} \oplus g) \\
&\quad + \tau(f_{x:(J\oplus\{1\})} \oplus f_{x:(I\oplus J)} \oplus g) \\
&\quad + \tau(f_{x:(J\oplus\{0,1\})} \oplus f_{x:(I\oplus J)} \oplus g) \\
&= \tau(f_{x:(I\oplus\{0\})} \oplus g) + \tau(f_{x:(I\oplus\{1\})} \oplus g) \\
&\quad + \tau(f_{x:(I\oplus\{0,1\})} \oplus g) \\
&= T_{x:I}(f,g)
\end{aligned}$$

□

[定理 1]（最小化定理） [3], [5] 任意の n 変数関数 f に対して，$\tau(f) = \min\{T(f,g) \mid g \in \mathcal{F}^{n-1}\}$ が成り立つ．ここで，$\mathcal{F}^{n-1}$ はすべての $(n-1)$ 変数関数の集合である．

最小化定理より以下のような単純な最小化アルゴリズムを得ることができる．ここで，m 変数以下の関数の最小 ESOP(およびその積項数 $\tau(f)$) は既知としている．

(最小化アルゴリズム)

（1） $n \leqq m$ であれば，f の最小 ESOP と $\tau(f)$ を返し終了する．

（2） $T_{x:I}(f,g)$ が最小になるような $g \in \mathcal{F}^{n-1}$ と添字 $I \in \{\{0\},\{1\},\{0,1\}\}$ を見つける．$\tau(f_{x:\{0\}} \oplus g)$, $\tau(f_{x:\{1\}} \oplus g)$, $\tau(f_{x:\{0,1\}} \oplus g)$, $\tau(f_{x:\{\}} \oplus g)$ は，アルゴリズムを再帰的に適用することで計算する．

（3） 上のステップで得られた関数 g について，F_0, F_1, F_2, F_3 をそれぞれ $f_{x:\{0\}} \oplus g$, $f_{x:\{1\}} \oplus g$, $f_{x:\{0,1\}} \oplus g$, $f_{x:\{\}} \oplus g$ に再帰的にアルゴリズムを適用して得られた ESOP とする．I に従って，以下の F とその積項数 $T_{x:I}(f,g)$ を返す．

$$F = \begin{cases} \bar{x}F_2 \oplus xF_3 \oplus F_1 & (I = \{0\}) \\ \bar{x}F_3 \oplus xF_2 \oplus F_0 & (I = \{1\}) \\ \bar{x}F_0 \oplus xF_1 \oplus F_3 & (I = \{0,1\}) \end{cases}$$

定理 1は $\tau(f)$ についての定理であるが，上記の最小化アルゴリズムのようにすれば，$\tau(f)$ を求める過程で最小 ESOP も求まる．以下では，$\tau(f)$ を求めることに議論を絞っているが，上記の要領で最小 ESOP を求めることを含んでいる．

[定義 3] n 変数関数 f，非負整数 k に対して，$\tau[k](f)$ を次のように定義する．

$$\tau[k](f) = \min\{T(f,g) \mid g \in \mathcal{F}^{n-1}, \tau(g) \leqq k\}$$

積項を組合わせることによって $\tau(g) \leqq k$ であるすべての関数 g を生成することができるので，図 1 のような $\tau[k](f)$ を計算するアルゴリズム *naive-tau*$[k](f)$ が構成できる．アルゴリズム中の $\mathcal{P}^{n-1}$ は 1 個の積項で表現できるすべての $(n-1)$ 変数関数の集合を表わす．

```
function naive-tau[k](f) : integer;
{ f is an n-variable function and k is a nonnegative integer }
var s : integer;
  procedure S(g, P);
  { g ∈ F^{n-1} and P is a set of products }
  begin
    if τ(g) ≧ k or P = ∅ then return;
    p ∈ P;
    if τ(g ⊕ p) = τ(g) + 1 then begin
      s := min{s, T(f, g ⊕ p)}; S(g ⊕ p, P − {p})
    end;
    S(g, P − {p})
  end;
begin s := T(f, 0); S(0, P^{n-1}); return s end;
```

図 1 *naive-tau*$[k](f)$: $\tau[k](f)$ を求めるアルゴリズム

Fig. 1 *naive-tau*$[k](f)$: an algorithm for $\tau[k](f)$

[定理 2] *naive-tau*$[k](f) = \tau[k](f)$

(証明) アルゴリズム *naive-tau*$[k](f)$ で，$\tau(g \oplus p) \leqq k$ であるすべての関数 $h = g \oplus p$ について $T(f,h)$ を計算しているのは明らかである．また，$\tau(h) > k$ である関数 h については $T(f,h)$ を計算しない．従って，定理が成立する． □

3. 計算の高速化のための枝刈り

アルゴリズム *naive-tau*$[k](f)$ は，$\tau(g) \leqq k$ であるすべての g を生成して $T(f,g)$ の最小値を求めるものであり，ある種の探索プログラムである．この節では，有効な枝刈りによる $\tau[k](f)$ 計算の高速化を図る．まず，そのための補題を与える．

[補題 2] f, g をそれぞれ n 変数関数，$(n-1)$ 変数関数とする．このとき，$\tau(g) + \tau(h) = \tau(g \oplus h)$ であ

る任意の $(n-1)$ 変数関数 h に対して，$T(f, g \oplus h) \geq T(f,g) - \tau(h)$ が成り立つ．

(証明) 補題の条件を満たす h に対して，$T(f, g \oplus h) = T_{x:\{0,1\}}(f, g \oplus h) = \tau(f_{x:\{1\}} \oplus g \oplus h) + \tau(f_{x:\{0\}} \oplus g \oplus h) + \tau(h \oplus g)$ と仮定する．この仮定は以下の議論で一般性を失うことはない．このとき，$T_{x:\{0,1\}}(f, g \oplus h)$ の各項 $\tau(f_{x:\{1\}} \oplus g \oplus h)$, $\tau(f_{x:\{0\}} \oplus g \oplus h)$, $\tau(g \oplus h)$ について次の式が成り立つ．

$$\tau(f_{x:\{1\}} \oplus g \oplus h) \geq \tau(f_{x:\{1\}} \oplus g) - \tau(h)$$
$$\tau(f_{x:\{0\}} \oplus g \oplus h) \geq \tau(f_{x:\{0\}} \oplus g) - \tau(h)$$
$$\tau(g \oplus h) = \tau(g) + \tau(h)$$

従って，上記の 3 式より次の式が成り立つ．

$$T(f, g \oplus h) = T_{x:\{0,1\}}(f, g \oplus h)$$
$$\geq T_{x:\{0,1\}}(f,g) - \tau(h) \geq T(f,g) - \tau(h)$$

□

上記補題より，$T(f,g) - (k - \tau(g)) \geq s$ であれば，$\tau(g \oplus h) = \tau(g) + \tau(h) \leq k$ であるどのような h についても，$T(f, g \oplus h) \geq s$ であることがわかる．ここで，アルゴリズム *naive-tau*$[k](f)$ の手続き $S(g, \mathcal{P})$ について考えると，$S(g, \mathcal{P})$ は，$\tau(g \oplus h) = \tau(g) + \tau(h) \leq k$ である h について $T(f, g \oplus h)$ を計算している．従って，$T(f,g) - (k - \tau(g)) \geq s$ であれば，$S(g, \mathcal{P})$ を実行しても s より小さい解（積項数）を得られないことがわかる．

以上の議論から，図 2のアルゴリズム *fast-tau*$[k]$ を得る．ここではアルゴリズム記述の簡潔さのために省略したが，実際のプログラムでは，$\tau(g)$, $T(f,g)$ を再計算しないため，手続き S にその値を引数として渡している．アルゴリズム *fast-tau*$[k]$ 中の下線部は，*naive-tau*$[k]$ と異なる部分である．

[定理 3] *fast-tau*$[k](f) = \tau[k](f)$

(証明) アルゴリズム *naive-tau*$[k]$ と *fast-tau*$[k]$ の違いは，手続き S の最初の if 文の条件が異なるだけである．*fast-tau*$[k]$ の条件 $T(f,g) - (k - \tau(g)) \geq s$ は $\tau(g) \geq k - (T(f,g) - s)$ と書けて，これは，$T(f,g) \geq s$ であること考慮すると *naive-tau*$[k]$ の条件 $\tau(g) \geq k$ より弱い命題である．従って，明らかに *naive-tau*$[k](f) \leq$ *fast-tau*$[k](f)$ である．

不等号の原因は，$\tau(g) < k$ であるにもかかわらず，*fast-tau*$[k]$ では，$\tau(g) \geq k - (T(f,g) - s)$ であるため S の本体である $T(f, g \oplus p)$ の計算およびそれに続く $S(g \oplus p, \mathcal{P} - \{p\})$, $S(g, \mathcal{P} - \{p\})$ の呼出しを行わないことにある．しかしながら，補題 2により，$\tau(g) + \tau(h) = \tau(g \oplus h) \leq k$ であるどのような h に対しても $T(f, g \oplus h) \geq T(f,g) - \tau(h) \geq T(f,g) - (k - \tau(g))) \geq s$ であることが保証されているので，*naive-tau*$[k](f)$ = *fast-tau*$[k](f)$ を得る．従って，定理 2より *naive-tau*$[k](f) = \tau[k](f)$ であるので *fast-tau*$[k](f) = \tau[k](f)$ が成り立つ． □

```
function fast-tau[k](f) : integer;
{ f is an n-variable function and k is a nonnegative integer }
var s : integer;
  procedure S(g, P);
  { g ∈ F^{n-1} and P is a set of products }
  begin
    if T(f,g) - (k - τ(g)) ≧ s or P = ∅ then return;
    p ∈ P;
    if τ(g ⊕ p) = τ(g) + 1 then begin
      s := min{s, T(f, g ⊕ p)};
      S(g ⊕ p, P - {p})
    end;
    S(g, P - {p})
  end;
begin s := T(f,0); S(0, P^{n-1}); return s end;
```

図 2 *fast-tau*$[k]$: $\tau[k](f)$ を求めるアルゴリズム

Fig. 2 *fast-tau*$[k]$: an algorithm for $\tau[k](f)$

4. 最小性を保証するための k の上界

前節では $\tau[k](f)$ を計算するアルゴリズムを与えた．我々の目標は $\tau(f)$ を計算する高速なアルゴリズムの開発である．$\tau[0](f) \geq \tau[1](f) \geq \cdots \geq \tau(f)$ であるので，k を順次大きくして $\tau[k](f)$ を計算すればいつかは $\tau(f)$ を計算できる．従って，計算すべき k の上界を評価できればよい．この節では，$\tau(f) = \tau[k](f)$ となる k の上界を求める．

[定義 4] $\tau[k](f) = \tau(f)$ となる最小の k を $\kappa(f)$ で表わす．すなわち，次のように $\kappa(f)$ を定義する．

$$\kappa(f) = \min\{k \mid \tau[k](f) = \tau(f)\}$$

$\kappa(f)$, $\tau[k](f)$ の定義より明らかに次の性質が成り立つ．

[性質 1]

- $k \geq \kappa(f)$ であれば，$\tau(f) = \tau[k](f)$ である．
- $\tau(f) = T_{x:I}(f,g)$ であれば，$\kappa(f) \leq \tau(g)$.

$\kappa(f)$ の上界について 2 つの補題を示す．

[補題 3] $\kappa(f) \leq \lfloor \tau(f)/3 \rfloor$

(証明) $\tau(f)$ の定義より，ある $I = \{0\}, \{1\}, \{0,1\}$ に対し，$\tau(f)$ は次のように表わされる．

$$\tau(f) = T_{x:I}(f,g)$$
$$= \tau(f_{x:(I\oplus\{0\})} \oplus g) + \tau(f_{x:(I\oplus\{1\})} \oplus g)$$
$$+\tau(f_{x:(I\oplus\{0,1\})} \oplus g)$$

ここで，3つの項 $\tau(f_{x:(I\oplus\{0\})} \oplus g)$, $\tau(f_{x:(I\oplus\{1\})} \oplus g)$, $\tau(f_{x:(I\oplus\{0,1\})} \oplus g)$ のうち最小のものを $\tau(f_{x:(I\oplus J)} \oplus g)$ $(J = \{0\},\{1\},\{0,1\})$ とすると，$\tau(f_{x:(I\oplus J)} \oplus g) \leqq \lfloor \tau(f)/3 \rfloor$ である．また，このとき，補題1より，$T_{x:I}(f,g) = T_{x:J}(f, f_{x:(I\oplus J)} \oplus g)$ であるので，$\tau(f) = T_{x:J}(f, f_{x:(I\oplus J)} \oplus g)$ である．従って，$\kappa(f) \leqq \tau(f_{x:(I\oplus J)} \oplus g) \leqq \lfloor \tau(f)/3 \rfloor$ が成り立つ． □

[定義 5]
$\gamma(f) = \max\{\tau(f_{x:J}) \mid J = \{0\},\{1\},\{0,1\}\}$

[補題 4] $\kappa(f) \leqq \tau(f) - \gamma(f)$

(証明) $\tau(f) = T_{x:I}(f,g)$ $(I = \{0\},\{1\},\{0,1\})$，$\gamma(f) = \tau(f_{x:J})$ $(J = \{0\},\{1\},\{0,1\})$ として，次の不等式を示す．

$$\tau(f_{x:(I\oplus J)} \oplus g) \leqq \tau(f) - \tau(f_{x:J})$$

上式が成り立てば，補題1より $\tau(f) = T_{x:I}(f,g) = T_{x:J}(f, f_{x:(I\oplus J)} \oplus g)$ であるので，$\kappa(f) \leqq \tau(f_{x:(I\oplus J)} \oplus g) \leqq \tau(f) - \tau(f_{x:J}) = \tau(f) - \gamma(f)$ が成り立つ．以下では，上の不等式が成り立つことを示す．

$J = \{0\}$ とする．$J = \{1\},\{0,1\}$ の場合も同様の議論ができる．$\tau(f) = T_{x:I}(f,g) = \tau(f_{x:(I\oplus\{0\})} \oplus g) + \tau(f_{x:(I\oplus\{1\})} \oplus g) + \tau(f_{x:(I\oplus\{0,1\})} \oplus g)$ であるので，

$$\tau(f_{x:(I\oplus J)} \oplus g)$$
$$= \tau(f_{x:(I\oplus\{0\})} \oplus g)$$
$$= \tau(f) - (\tau(f_{x:(I\oplus\{1\})} \oplus g) + \tau(f_{x:(I\oplus\{0,1\})} \oplus g))$$

と表わせる．ここで，$(f_{x:(I\oplus\{1\})} \oplus g) \oplus (f_{x:(I\oplus\{0,1\})} \oplus g) = f_{x:\{0\}} = f_{x:J}$ より $\tau(f_{x:(I\oplus\{1\})} \oplus g) + \tau(f_{x:(I\oplus\{0,1\})} \oplus g) \geqq \tau(f_{x:\{0\}}) = \tau(f_{x:J})$ が成り立つので，上式より次の不等式を得る．

$$\tau(f_{x:(I\oplus J)} \oplus g) = \tau(f_{x:(I\oplus\{0\})} \oplus g)$$
$$\leqq \tau(f) - \tau(f_{x:\{0\}}) = \tau(f) - \tau(f_{x:J})$$

□

上記の2つの補題と，$\tau(f) \leqq T(f,g)$ より次の系を得る．

[系 1] f, g をそれぞれ n 変数関数，$(n-1)$ 変数関数とすると，$\kappa(f) \leqq \min\{\lfloor T(f,g)/3 \rfloor, T(f,g) - \gamma(f)\}$ が成り立つ．

上記の系により，$T(f,0)$ と $\gamma(f)$ を最初に計算し，$k = \min\{\lfloor T(f,0)/3 \rfloor, T(f,0) - \gamma(f)\}$ としてアルゴリズム $fast\text{-}tau[k](f)$ を実行すれば $\tau(f)$ を計算できる．さらに，k の値をアルゴリズム実行中に得られた s $(= T(f,g))$ により，動的に減らすことにより，その探索範囲を減らすこともできる．そのアルゴリズム $tau1$ を図3 に示す（下線部が k の値を s によって更新している部分である）．

```
function tau1(f) : integer;
{ f is an n-variable function }
var s, k : integer;
  procedure S(g, P);
  { g ∈ F^{n-1} and P is a set of products }
  begin
    if T(f,g) - (k - τ(g)) ≧ s or P = ∅ then
    return;
    p ∈ P;
    if τ(g ⊕ p) = τ(g) + 1 then begin
      s := min{s, T(f, g ⊕ p)};
      k := min{⌊s/3⌋, s - γ(f)};
      S(g ⊕ p, P - {p})
    end;
    S(g, P - {p})
  end;
begin
  s := T(f,0);
  k := min{⌊s/3⌋, s - γ(f)};
  S(0, P^{n-1});
  return s
end;
```

図3 $tau1$: $\tau(f)$ を求めるアルゴリズム
Fig. 3 $tau1$: an algorithm for $\tau(f)$

[定理 4] $tau1(f) = \tau(f)$

(証明) $tau1(f) \geqq \tau(f)$ は明らかである．以下では $tau1(f) \leqq \tau(f)$ を示す．アルゴリズム $tau1(f)$ のリターン値を s_{min} とする．このとき，$k = \min\{\lfloor s_{min}/3 \rfloor, s_{min} - \gamma(f)\}$ としてアルゴリズム $fast\text{-}tau[k](f)$ を実行すると，アルゴリズム $fast\text{-}tau[k](f)$ で計算する $T(f, g\oplus p)$ はすべて $tau1(f)$ でも計算されるので，$fast\text{-}tau[k](f) \geqq tau1(f)$ が成り立つ．従って，定理3より $\tau[k](f) = fast\text{-}tau[k](f) \geqq tau1(f)$ である．系1より $\kappa(f) \leqq \min\{\lfloor s_{min}/3 \rfloor, s_{min} - \gamma(f)\} = k$ であるので，$\tau(f) = \tau[k](f) \geqq tau1(f)$ が成り立つ． □

k の値を減らすことができれば，計算時間をさらに減らすことができる．そのための補題を与える．

[補題 5] $\tau(f) < s$ であれば，$\kappa(f) \leqq \min\{\lfloor(s-1)/3\rfloor, s-1-\gamma(f)\}$ である．

(証明) 補題 3，4より $\kappa(f) \leqq \min\{\lfloor\tau(f)/3\rfloor, \tau(f)-\gamma(f)\}$ であり，補題の仮定より $\tau(f) \leqq s-1$ であるので，明らかに補題は成り立つ． □

上記補題により，アルゴリズム実行中のある時点までに積項数 $s = T(f,g)$ が求まっているとき，以降の探索で s より小さい値を得るには，$k = \min\{\lfloor(s-1)/3\rfloor, s-1-\gamma(f)\}$ として探索をすればよいことがわかる．そのアルゴリズム *tau2* を図 4に示す（下線部が *tau1* を修正した部分である）．

```
function tau2(f) : integer;
{ f is an n-variable function }
var s, k : integer;
  procedure S(g, P);
  { g ∈ F^{n-1} and P is a set of products }
  begin
    if T(f,g) − (k − τ(g)) ≧ s or P = ∅ then
    return;
    p ∈ P;
    if τ(g ⊕ p) = τ(g) + 1 then begin
      s := min{s, T(f, g ⊕ p)};
      k := min{⌊(s − 1)/3⌋, s − 1 − γ(f)};
      S(g ⊕ p, P − {p})
    end;
    S(g, P − {p})
  end;
begin
  s := T(f, 0);
  k := min{⌊(s − 1)/3⌋, s − 1 − γ(f)};
  S(0, P^{n-1});
  return s
end;
```

図 4 *tau2*: $\tau(f)$ を求めるアルゴリズム
Fig. 4 *tau2*: an algorithm for $\tau(f)$

[定理 5] $tau2(f) = \tau(f)$

(証明) アルゴリズム $tau2(f)$ のリターン値を s_{min} とし，$k_{min} = \min\{\lfloor(s_{min}-1)/3\rfloor, s_{min}-1-\gamma(f)\}$とすると，定理 4の証明と同様の議論により，$\tau[k_{min}](f) = fast\text{-}tau[k_{min}](f) \geqq tau2(f) = s_{min}$ が成り立つ．ここで，$\tau(f) < s_{min}$ と仮定すると，補題 5より $\kappa(f) \leqq k_{min}$ であり，$\tau(f) = \tau[k_{min}](f) \geqq tau2(f) = s_{min}$ を得る．これは仮定 $\tau(f) < s_{min}$ に矛盾する． □

5. 実験結果

本論文では，最終的なアルゴリズム *tau2* を得るために，探索範囲を制限する次の 2 つの手法を用いた．

（1） $T(f, g \oplus h)$ の下界の利用 (*naive-tau*[k], *fast-tau*[k])

（2） $\kappa(f)$ の上界の利用 (*tau1*, *tau2*)

それぞれの手法の効果を調べるために，上記の 2 つの手法のすべての組合せに対応する 12 個のアルゴリズムを実装し，$T(f, g \oplus p)$ の呼出し回数を数えた．実験対象とした関数は 5 変数関数のすべての LP 同値類代表関数（6936 個）である．表 1にそれぞれのアルゴリズムに対する $T(f,g)$ の呼出し回数の 1 関数当りの平均を示す．表中の *tau1*, *tau2* は $\kappa(f)$ の上界を，それぞれ $\kappa(f) \leqq \min\{\lfloor s/3\rfloor, s-\gamma(f)\}$, $\kappa(f) \leqq \min\{\lfloor(s-1)/3\rfloor, s-1-\gamma(f)\}$ としたアルゴリズムに対応する．また，*naive*, *fast* は，$T(f, g \oplus h)$ の下界を使わない，使うに対応する．すなわち，$S(g, \mathcal{P})$ の最初の if 文の条件を $\tau(g) \geqq k$ とするか $T(f,g) - (k - \tau(g)) \geqq s$ とするかに対応する．表より，$T(f, g \oplus h)$ の下界を用いて *naive* を *fast* に変えたときの削減効果が大きいことがわかる．また，$\kappa(f)$ の上界については，$\lfloor(s-1)/3\rfloor$ と $s-1-\gamma(f)$ のどちらかの一方の上界が不必要ということはなく，2 つの上界を使うことによって，最終的なアルゴリズムの高速性が達成されている．

表 1 $\kappa(f)$ の上界の違いによる枝刈り効果
Table 1 Effect of pruning methods for upper bounds of $\kappa(f)$

$\kappa(f)$	*naive*	*fast*
$\kappa(f) \leqq \lfloor s/3\rfloor$	2537.2	254.2
$\kappa(f) \leqq s - \gamma(f)$	11187.0	676.7
tau1	2211.5	240.6
$\kappa(f) \leqq \lfloor(s-1)/3\rfloor$	1030.0	174.2
$\kappa(f) \leqq s - 1 - \gamma(f)$	579.0	128.5
tau2	526.5	120.1

最後に実際の計算時間について述べる．我々は LISP（CMU-CL）を用いてプログラムを記述し，AMD Athron 1.4GHz, FreeBSD 4.3-R の計算機で実行した．上記の 6936 個の LP 同値類 5 変数代表関数を，*tau2* を用いて計算するのに要した時間は 1.25 秒であり，1 関数当りわずか 0.18×10^{-3} 秒である．さらに，すべての 6 変数対称関数（128 個）についても実行してみた．その結果，1 関数当りの計算時間は 116 秒（4時間8分/128個 = 116）であり，最長計算時間は 672.86 秒であった．これは，これまでに知られている最小化アルゴリズムとしては格段に高速であることが確認できた．なお，この実験では，対称関数を用いたが，本アルゴリズムは任意の関数に適用

できる．また，7変数関数についても，$\tau(f) \leq 9$ 程度の関数については，現実的な計算時間で最小化できることを確認した．

今回の実験に用いたプログラミング言語はLISPであるので，その計算速度はあまり期待できないはずであるが，十分に高速である．C言語など，より動作が高速なプログラミング言語で実装すればさらに高速化できることが期待できる．

6. むすび

本論文では，従来よりも高速なESOP最小化アルゴリズムを示した．本アルゴリズムでは，解の探索における枝刈りおよび計算すべき k の上界の評価の2つの手法により高速化を達成した．実験結果より，2つの手法を組み合わせることで効果的に解の探索を減らせることが確かめられた．また，6変数以下の任意の関数と一部の7変数関数について，現実的な計算時間で最小化できることを確かめられた．

文　献

[1] Chattopadhyay, S., Roy, S., and Chaudhuri, P. P., "KGPMIN: an efficient multilevel multioutput AND-OR-XOR minimizer," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 3, pp. 257–265, Mar. 1997.

[2] Debnath, D. and Sasao, T., "Minimization of AND-OR-EXOR three-level networks with AND gate sharing," *IEICE Trans. Information and Systems*, vol. E80-D, no. 10, pp. 1001–1008, Oct. 1997.

[3] 平山貴司, 西谷泰昭, "論理関数のあるクラスについて最小性を保証するAND-EXOR論理式の簡単化アルゴリズム", 電子情報通信学会論文誌 D-I, vol. J78-D-I, no. 4, pp. 409–415, Apr. 1995.

[4] Luccio, F. and Pagli, L., "On a new boolean function with applications," *IEEE Trans. Comput.*, vol. 48, no. 3, pp. 296–310, Mar. 1999.

[5] Nishitani, Y. and Shimizu, K., "Lower bounds on size of periodic functions in exclusive-OR sum-of-products expressions," *IEICE Trans. Fundamentals of Electronics, Information and Communication Engineers*, vol. E77-A, no. 3, pp. 475–482, Mar. 1994.

[6] Perkowski, M. and Chrzanowska-Jeske, M., "An exact algorithm to minimize mixed-radix exclusive sums of products for incompletely specified Boolean functions," *Proc. of International Symposium on Circuits and Systems '90*, pp. 1652–1655, 1990.

[7] Sasao, T. and Besslich, P., "On the complexity of mod-2 sum PLA's," *IEEE Trans. Comput.*, vol. 39, no. 2, pp. 262–266, Feb. 1990.

[8] Sasao, T., "EXMIN2: A simplification algorithm for exclusive-OR-sum-of-products expressions for multiple-valued input two-valued output functions," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 5, pp. 621–632, May 1993.

[9] Sasao, T., "An exact minimization of AND-EXOR expressions using BDD's," IFIP WG 10.5 Reed-Muller'93, pp. 91–98, Germany, 1993.

[10] Sasao, T., "Representations of logic functions using EXOR operators," in (Sasao and Fujita eds.) *Representations of Discrete Functions*, pp. 29–54, Kluwer Academic Publishers, 1996.

[11] Song, N. and Perkowski, M. A., "Minimization of exclusive sum-of-products expressions for multiple-valued input, incompletely specified functions," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 4, pp. 385–395, Apr. 1996.