F-046

サブエルブラン領域計算の効率化の一手法
An Efficient Algorithm for *SHU* Computation

何 立風 (Lifeng He)† 巣 宇燕 (Yuyan Chao)†† Zhenhao Shi†††
中村 剛士 (Tsuyoshi Nakamura) ††† 伊藤 英則 (Hidenori Itoh) †††

1. まえがき

文献 3) では，自動定理証明の基礎となるエルブラン定理を改良した一手法について述べた．まず，節集合に現れる述語と関数の各引数において，その引数に対応する $\mathcal{S}$ のエルブラン領域の部分領域（サブエルブラン領域と呼ぶ）を計算するアルゴリズムを提案した．つぎに，節集合 $\mathcal{S}$ の充足不能性を調べるとき，$\mathcal{S}$ のエルブラン領域における基礎節の集合のかわりに，$\mathcal{S}$ のそれぞれの変数に，その変数に対応する $\mathcal{S}$ のエルブラン領域の部分領域の要素を代入して得られた基礎節の集合だけを考えればよいことをを証明した．さらに，サブエルブラン領域を利用してモデル生成定理証明[1),2),4)] の効率を改善できることを示した．

しかし，文献 3) に提案したサブエルブラン領域を計算するアルゴリズムでは，1 つのサブエルブラン領域を計算するために，与えれた節集合にあるすべての節を一通り処理しなければならない．そのため，計算時間が長過ぎ実用的でない場合もある．

本稿では，与えられた節集合にあるすべての節を一通り処理すれば，すべてのサブエルブラン領域を計算できる効率的なアルゴリズムを提案する．また，定理証明のベンチマーク問題を用いて提案手法の有効性を示す．

2. 前アルゴリズム

本稿では，文献 3) と同様，n-引数述語記号または関数記号 α の i 番目の引数を $\alpha^n\langle i\rangle$ で表す．また，項 τ が引数 $\alpha^n\langle i\rangle$ の値として現れることを $app(\tau, \alpha^n\langle i\rangle)$ で表す．文献 3) により，ある引数に対応するサブエルブラン領域 *SHU* (sub-Herbrand Universe) を計算するアルゴリズムは次のようになる．

アルゴリズム 2.1 （*SHU* を計算するアルゴリズム） $\mathcal{S}$ を節集合，$\alpha^n\langle i\rangle$ $(1 \le i \le n)$ を $\mathcal{S}$ に現れる引数とする．$\alpha^n\langle i\rangle$ に対応する *SHU* は次のように計算した集合 $\mathcal{H}$ である．

1. $\mathcal{H} = \phi$, $\mathcal{M} = \{\alpha^n\langle i\rangle\}$, $\mathcal{N} = \phi$ とする．
2. $\mathcal{M} = \phi$ が成立すれば，$\mathcal{H}$ は $\alpha^n\langle i\rangle$ に対応する *SHU* である．ただし，$\mathcal{H}$ に個体記号が含まれていなければ，$\mathcal{H} = \mathcal{H} \cup \{a\}$ とする．a は任意に選択した節集合 $\mathcal{S}$ の個体記号である．ただし，$\mathcal{S}$ に個体記号が含まれていないとき，a は仮想個体記号である．
3. $\mathcal{M}$ の最初の要素 $\beta^m\langle j\rangle$ を $\mathcal{N}$ に移し，$\mathcal{S}$ に現れる各 $app(\varepsilon, \beta^m\langle j\rangle)$ $(1 \le j \le m)$ にたいして，
   (1) ε が定数であれば，$\mathcal{H} = \mathcal{H} \cup \{\varepsilon\}$ とする．
   (2) ε が関数 $f(\tau_1, \ldots, \tau_t)$ であれば，$\mathcal{H} = \mathcal{H} \cup \mathcal{V}\langle f\rangle$ とする．ここでは，$\mathcal{V}\langle f\rangle$ は関数 $f(\tau_1, \ldots, \tau_t)$ の取りうる値の集合であり，その計算方法は後に示す．
   (3) ε が変数 X である場合，C を X が現れる節とし，C に現れる各 X はある u 引数述語または関数 γ の k 番目の引数の値となっている．つまり，$app(X, \gamma^u\langle k\rangle)$ が C に現れている．そのとき，$\gamma^u\langle k\rangle \notin \mathcal{M} \cup \mathcal{N}$ であれば，$\gamma^u\langle k\rangle$ を $\mathcal{M}$ に加える．
4. 2 に戻る．

上述のアルゴリズムが停止したとき，$\mathcal{N}$ にあるすべての引数が同ドメイン (同じサブエルブラン領域をもつ) 引数である．

アルゴリズム 2.2 （すべての引数の *SHU* を計算するアルゴリズム） $\mathcal{S}$ を節集合とする．$\mathcal{S}$ に現れるすべての述語記号や関数記号の引数に対応する *SHU* は次のように求められる．

1. $\mathcal{T}$ を $\mathcal{S}$ に現れるすべての述語記号や関数記号の引数の集合とする．また，$j = 0$ とする．
2. $\mathcal{T}$ が空であれば，$\mathcal{H}_1, \ldots \mathcal{H}_j$ が導出した *SHU* である．$\mathcal{N}_k$ $(1 \le k \le j)$ に現れるすべての引数は同ドメイン引数であり，同一 *SHU* $\mathcal{H}_k$ をもつ．
3. $\alpha^n\langle i\rangle$ を $\mathcal{T}$ の一番目の要素とし，$j = j + 1$ とする．定義 2.1 によって，$\alpha^n\langle i\rangle$ の *SHU* $\mathcal{H}_j$ と $\alpha^n\langle i\rangle$ の同ドメイン引数集合 $\mathcal{N}_j$ を計算する．そこで，$\mathcal{T}$ から $\mathcal{N}_j$ に属する要素があれば除去する．
4. 2 に戻る．

アルゴリズム 2.3 （エルブラン領域の形の *SHU* を計算するアルゴリズム） $\mathcal{S}$ を節集合，$f_1, \ldots, f_m$ を $\mathcal{S}$ に現れる関数記号とする．また，$\mathcal{H}_1, \ldots, \mathcal{H}_n$ を定義 2.2 によって導出した *SHU* とし，$\mathcal{V}\langle f_j\rangle$ $(1 \le j \le m)$ を $\mathcal{H}_1, \ldots, \mathcal{H}_n$ に現れる関数記号の取りうる値の集合とする．

$\mathcal{H}_i^*(0)$ $(1 \le i \le n)$ を $\mathcal{H}_i$ に現れる個体記号の集合として，$\mathcal{V}^*(f_j, 0) = \phi$ $(1 \le j \le m)$ とする．

f_j $(1 \le j \le m)$ は h_j 引数の関数記号，$f_j^{h_j}(t)$ $(1 \le t \le h_j)$

☆ 本研究は，豊秋奨学会の研究助成を受けている．
† 愛知県立大学，情報科学研究科
†† 名古屋産業大学，環境情報マネジメント研究科
††† 名古屋工業大学，情報工学研究科

に対応する SHU は $\mathcal{H}_{u_j}$ $(1 \leq u_j \leq n)$ とする．また，$k = 0, 1, 2, \ldots$ にたいして，

$$\mathcal{V}^*(f_j, k+1) = \{f_j(\alpha_1, \ldots, \alpha_{h_j}) \mid \alpha_1 \in \mathcal{H}^*_{u_1}(k), \ldots, \alpha_{h_j} \in \mathcal{H}^*_{u_{h_j}}(k)\}$$

さらに，$1 \leq i \leq n$ の各 i にたいして，

$$\mathcal{H}^*_i(k+1) = \{\mathcal{H}^*_i(k) \cup \mathcal{V}^*(f_v, k+1) \mid \mathcal{V}\langle f_v\rangle \in \mathcal{H}_i\}$$

このとき，$\mathcal{V}\langle f_j\rangle = \mathcal{V}^*(f_j, \infty)$ （つまり，関数 f_j の取りうる値の集合である），$\mathcal{H}^*_i(\infty)$ は $\mathcal{H}_i$ のエルブラン領域の形である．

3. 効率的なアルゴリズム

上述のサブエルブラン計算法は効率的なものではない．定義 2.1 により，ある引数に対応するサブエルブラン領域を計算するとき，節集合にあるすべての節を一通り処理しなければならない．そのとき，処理中の引数については，節集合にあるすべての引数と照合しなければならない．そのため，計算時間が長過ぎて実用的でない場合もある．例えば，TPTP library にサブエルブラン計算だけで 300 秒 (定理証明コンテストの制限時間，http://www.cs.miami.edu/~ tptp/CASC/ をご参照) を越えた問題は多数ある (実験結果参照)．

本稿では，節集合にあるすべての節を一通り処理するだけですべてのサブエルブラン領域を計算できるアルゴリズムを提案する．

定義 3.1 (同ドメイン引数と同ドメイン引数集合) 同じサブエルブラン領域をもつ引数を**同ドメイン引数**，同ドメイン引数の集合を**同ドメイン引数集合**と呼ぶ．

定義 3.2 (変数に対応する引数) X を節 C に現れる変数とする．節 C に現れる $app(X, \alpha^n\langle i\rangle)$ のような引数 $\alpha^n\langle i\rangle$ を変数 X に対応する引数と呼ぶ．

補題 3.1 ある変数に対応するすべての引数は同ドメイン引数である．

ある変数に対応するすべての引数は推論においていつも必ず同じ値をとるため，補題 3.1 は明らかである．

補題 3.2 $\mathcal{D}_1$ と $\mathcal{D}_2$ を同ドメイン引数集合とする．もし $\mathcal{D}_1$ と $\mathcal{D}_2$ が共通の引数一つ以上をもっていれば，$\mathcal{D}_1$ と $\mathcal{D}_2$ に属するすべての引数は同ドメイン引数である

補題 3.2 は定義 3.1 により明らかである．補題 3.2 により，共通の引数をもつ同ドメイン引数集合を併合することができる．

アルゴリズム 3.1 （節における同ドメイン引数集合の計算アルゴリズム) 節 C における同ドメイン引数集合は次のように計算できる．

1) $X_1, \cdots, X_n$ を節 C に現れる変数とする．X_1 から X_n まで各 X_i において，$\mathcal{E}$ を変数 X_i に対応する同ドメイン引数集合，$\mathcal{D}_1, \cdots, \mathcal{D}_m$ をこれまで求めた同ドメイン引数集合候補 (最初は存在しない) とする．$\mathcal{D}_1, \cdots, \mathcal{D}_m$ に $\mathcal{E}$ と同じ引数を 1 つ以上もつもの $\mathcal{D}_{p_1}, \cdots, \mathcal{D}_{p_l}$ $(1 \leq p_i \leq m, 1 \leq i \leq l)$ とする．$\mathcal{D}_{p1}, \cdots, \mathcal{D}_{pl}$ と $\mathcal{E}$ を併合して新たな同ドメイン引数集合候補とする．一方，そのような同ドメイン引数集合がなければ，$\mathcal{E}$ を新たな同ドメイン引数集合候補として加える．
2) 節 C に現れる各 $app(c, \alpha^r\langle i\rangle)$ (c は定数) にたいして，もし引数 $\alpha^r\langle i\rangle$ がこれまで求められたある同ドメイン引数集合候補に属してあれば，なにもしない．そうでなければ，$\{\alpha^n\langle i\rangle\}$ を新たな同ドメイン引数集合候補として加える．
3) 節 C に現れる各 $app(f(t_1, \cdots, t_u), \beta^v\langle j\rangle)$ にたいして，もし引数 $\beta^v\langle j\rangle$ がこれまで求められたある同ドメイン引数集合候補に属してあれば，なにもしない．そうでなければ，$\{\beta^v\langle j\rangle\}$ を新たな同ドメイン引数集合として加える．

例 3.1 アルゴリズム 3.1 により，節 $\neg p(X, c) \vee \neg p(X, f(Y)) \vee q(X, Y) \vee q(Y, X)$ における同ドメイン引数集合の計算は次のようになる．

変数 X に対応する同ドメイン引数集合候補 $\mathcal{D}_x$ は $\{p^2\langle 1\rangle, q^2\langle 1\rangle, q^2\langle 2\rangle\}$ である．変数 Y に対応する同ドメイン引数集合候補 $\mathcal{D}_y$ は $\{f^1\langle 1\rangle, q^2\langle 2\rangle, q^2\langle 1\rangle\}$ である．

しかし，$q^2\langle 1\rangle \in \mathcal{D}_x$ かつ $q^2\langle 1\rangle \in \mathcal{D}_y$ であるから，$\mathcal{D}_x$ と $\mathcal{D}_y$ を併合して新たな同ドメイン引数集合候補 $\mathcal{D}_1 = \{p^2\langle 1\rangle, q^2\langle 1\rangle, q^2\langle 2\rangle, f^1\langle 1\rangle\}$ となる．

定数 c において，$app(c, p^2\langle 2\rangle)$ がある．$p^2\langle 2\rangle \notin \mathcal{D}_1$ のため，$\mathcal{D}_2 = \{p^2\langle 2\rangle\}$ は新たな同ドメイン引数集合候補として加えられる．

関数 $f(Y)$ において，$app(f(Y), p^2\langle 2\rangle)$ がある．$p^2\langle 2\rangle \in \mathcal{D}_2$ のため，なにもしない．

以上により，与えられた節における同ドメイン引数集合は 2 つ，つまり，$\mathcal{D}_1 = \{p^2\langle 1\rangle, q^2\langle 1\rangle, q^2\langle 2\rangle, f^1\langle 1\rangle\}$ と $\mathcal{D}_2 = \{p^2\langle 2\rangle\}$ が計算できる．

アルゴリズム 3.2 （節集合における同ドメイン引数集合の計算アルゴリズム) $\mathcal{S}$ を節集合，$C_1, \ldots, C_n$ を $\mathcal{S}$ に現れる節とする．$\mathcal{S}$ における同ドメイン引数集合は節 C_1 から C_n まで各 C_i に対して次のように処理して計算できる．

$\mathcal{D}_1, \cdots, \mathcal{D}_m$ を同ドメイン引数集合候補 (最初は存在しない) とする．$\mathcal{E}_1, \ldots, \mathcal{E}_t$ を定義 3.1 による節 C_i から求められた同ドメイン引数集合候補とする．$\mathcal{E}_1$ から $\mathcal{E}_t$ まで各 $\mathcal{E}_j$ にたいして，$\mathcal{D}_{p_1}, \ldots, \mathcal{D}_{p_u}$ $(1 \leq p_k \leq m, 1 \leq k \leq u)$ をそれぞれ $\mathcal{E}_j$ と同じ引数を 1 つ以上もつ同ドメイン引数集合候補とする．$\mathcal{D}_{p_1}, \ldots, \mathcal{D}_{p_u}$ を $\mathcal{E}_j$ と併合して新たな同ドメイン引数集合候補とする．ただし，このような同ドメイン引数集合候補は存在しない場合，$\mathcal{E}_j$ を新たな同ドメイン引数集合候補として加える．

例 3.2 $\mathcal{S}$ を次の節集合とする．

$$\neg p_1(a)$$
$$\neg p_2(f(b))$$
$$p_1(X) \vee p_2(Y) \vee p_3(Y)$$

アルゴリズム 3.1 により，節 $\neg p_1(a)$ における同ドメイン引数集合は 1 つ，つまり，$\{p_1^1\langle 1\rangle\}$ が求められる．また，節 $\neg p_2(f(b))$ における同ドメイン引数集合は 2 つ，つまり，$\{p_2^1\langle 1\rangle\}$ と $\{f^1\langle 1\rangle\}$ が求められる．さらに，節 $p_1(X) \vee p_2(Y) \vee$

$p_3(Y)$ における同ドメイン引数集合は 2 つ，つまり，$\{p_1^1\langle 1\rangle\}$ と $\{p_2^1\langle 1\rangle, p_3^1\langle 1\rangle\}$. が求められる．
アルゴリズム 3.2 により，節 $\neg p_1(a)$ と $\neg p_2(f(b))$ を処理するとき，同ドメイン引数集合の併合は必要ない．節 $p_1(X) \vee p_2(Y) \vee p_3(Y)$ を処理するとき，$\{p_1^1\langle 1\rangle\}$ を $\{p_1^1\langle 1\rangle\}$ と，$\{p_2^1\langle 1\rangle, p_3^1\langle 1\rangle\}$ を $\{p_2^1\langle 1\rangle\}$ と併合できる．そのため，節集合 $\mathcal{S}$ から三つの同ドメイン引数集合，つまり，$\mathcal{D}_1$=$\{p_1^1\langle 1\rangle\}$, $\mathcal{D}_2$= $\{f^1\langle 1\rangle\}$ と $\mathcal{D}_3$= $\{p_2^1\langle 1\rangle, p_3^1\langle 1\rangle\}$ が計算できる．

定義 3.3 (同ドメイン引数集合に対応する定数集合と関数記号集合) $\mathcal{S}$ を節集合，$\mathcal{D}$ を $\mathcal{S}$ における同ドメイン引数集合の 1 つとする．$\mathcal{D}$ に対応する定数集合 $\mathcal{C}$ (関数記号集合 $\mathcal{F}$) は $app(c, \alpha^n\langle i\rangle))$ $(app(f, \alpha^n\langle i\rangle)))$ かつ $\alpha^n\langle i\rangle \in \mathcal{D}$ のような定数 c (関数 f) の集合である．ただし，$\mathcal{C}$ が空集合であれば，$\mathcal{C} = \{a\}$ とする．そこの a は $\mathcal{S}$ のエルブラン領域に現れる任意の定数である．

例 3.3 例 3.2 の $\mathcal{D}_1$ に対応する定数集合 $\mathcal{C}_1$ と関数記号集合 $\mathcal{F}_1$ はそれぞれ $\{a\}$ と $\emptyset$ である．$\mathcal{D}_2$ に対応する定数集合 $\mathcal{C}_2$ と関数記号集合 $\mathcal{F}_2$ はそれぞれ $\{b\}$ と $\emptyset$ である．また，$\mathcal{D}_3$ に対応する定数集合 $\mathcal{C}_3$ と関数記号集合 $\mathcal{F}_3$ はそれぞれ $\{a\}$ (または $\{b\}$) と $\{f\}$ である．

アルゴリズム 3.3 (節集合の SHUs を計算するアルゴリズム). $\mathcal{S}$ を節集合，$\mathcal{D}_1, \ldots, \mathcal{D}_n$ を $\mathcal{S}$ における同ドメイン引数集合，$\mathcal{C}_1, \ldots, \mathcal{C}_n$ および $\mathcal{F}_1, \ldots, \mathcal{F}_n$ をそれぞれ対応する定数集合と関数記号集合とする．
各 m-引数関数 $f(\alpha_1, \ldots, \alpha_m)$ において，ここでは，$f^m\langle j\rangle \in \mathcal{D}_{t_j}$ $(1 \leq j \leq m$ とする．また，$1 \leq t_j \leq n)$，$\mathcal{V}(f, 0) = \emptyset$, $\mathcal{H}_i(0) = \mathcal{C}_i$ $(1 \leq i \leq n)$ とする．さらに，各 $k = 0, 1, 2, \ldots$ にたいして，$\mathcal{V}(f, k+1) = \{f(\alpha_1, \ldots, \alpha_m) | \alpha_j \in \mathcal{H}_{t_j}(k)\}$, $\mathcal{H}_i(k+1) = \mathcal{H}_i(k) \cup \{\mathcal{V}(f, k+1) | f \in \mathcal{F}_i\}$ とする．
$\mathcal{H}_1(\infty)$, ..., $\mathcal{H}_n(\infty)$ はそれぞれ $\mathcal{D}_1$,..., $\mathcal{D}_n$ に対応する SHU である．

例 3.4 $\mathcal{S}$ を例 3.2 に示した節集合，$\mathcal{D}_1$, $\mathcal{D}_2$ と $\mathcal{D}_3$ を例 3.2 に計算した $\mathcal{S}$ における同ドメイン引数集合，$\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$, $\mathcal{F}_1$, $\mathcal{F}_2$ と $\mathcal{F}_3$ を それぞれ例 3.3 に求められた $\mathcal{D}_1$, $\mathcal{D}_2$ と $\mathcal{D}_3$ に対応する定数集合と関数記号集合とする．アルゴリズム 3.3 により，
$\mathcal{V}(f, 0) = \emptyset$, $\mathcal{H}_1(0) = \mathcal{C}_1 = \{a\}$, $\mathcal{H}_2(0) = \mathcal{C}_2 = \{b\}$, $\mathcal{H}_3(0) = \mathcal{C}_3 = \{a\}$.

$\mathcal{V}(f, 1) = \{f(b)\}$, $\mathcal{H}_1(1) = \{a\}$, $\mathcal{H}_2(1) = \{b\}$, $\mathcal{H}_3(1) = \{a, f(b)\}$.

$\mathcal{V}(f, 2) = \{f(b)\}$, $\mathcal{H}_1(2) = \{a\}$, $\mathcal{H}_2(2) = \{b\}$, $\mathcal{H}_3(2) = \{a, f(b)\}$.

.....

$\mathcal{V}(f, \infty) = \{f(b)\}$, $\mathcal{H}_1(\infty) = \{a\}$, $\mathcal{H}_2(\infty) = \{b\}$, $\mathcal{H}_3(\infty) = \{a, f(b)\}$.
つまり，$\mathcal{D}_1$, $\mathcal{D}_2$ と $\mathcal{D}_3$ に対応する SHU は それぞれ $\{a\}$, $\{b\}$ と $\{a, f(b)\}$ である．

提案したアルゴリズムの証明も前アルゴリズムと同様に行なえる．スペース制限のため，ここで省略する．

4. 実験結果

定理証明のベンチマーク問題集 TPTP library[☆] version 3.1.1 には，8013 問の問題がある．その中の 4365 問は非領域限定かつ非ホーン節を含む問題である．本稿で提案した手法が適用できる問題，すなわち，2 つ以上の SHU が導出される問題の数は 709 である．

前アルゴリズムと本稿で提案したアルゴリズムをそれぞれ SICS Prolog 言語で実装した． 計算機 (Intel PentiumIII/980MHZ, 512MB, Solaris OS) 上で両手法を適用可能な 709 の問題にたいして，制限時間を 2 秒，5 秒，10 秒，50 秒，100 秒，200 秒，300 秒としたそれぞれの場合，変換できた問題数を表 1 に示す． なお，すべての問題において両手法の結果は完全に一致する．

両アルゴリズムにおける節集合に含まれる引数の数と平均実行時間を表 2 に示す．ただし，時間制限は 10000 秒とする．また，節集合にある引数の数が 50000 越える問題において，前アルゴリズムは設定の制限時間内に終了しなくなる．例えば，問題 SYN826-1(節数 2004，引数総数は 1542050，分割された SHU の数は 938 である) では，提案したアルゴリズムは 287 秒で変換できたのに対して，前アルゴリズムは 1 日かかっても終了しなかった．

本手法を組み込んだモデル生成推論システム R-SATCHMO[2] を 2006 年 8 月自動推論に関する国際会議 IJCAR[☆☆] で行われた定理証明システムコンテスト CASC-J3[☆☆☆] に用いられた問題にチャレンジしたが，全体からみれば，結果が良かったとは言えない．その理由として以下の点を挙げることができる：

(1) 文献 5) において指摘されたように，定理証明システムコンテストに用いられた TPTP 問題集の問題の殆んどは反駁手続きシステム向きの問題である；

(2) R-SATCHMO は不等式処理ための手続きを組み込んでいない；

(3) R-SATCHMO が Prolog によって実装されたの対して，ほかの推論システムは C 言語または C++言語で実装されたものである．文献 6) により，同じ機能のプログラムにおいて，C 言語で実装したものは Prolog 言語で実装したものより 100 倍以上速くなる．

(4) 定理証明システムコンテスト CASC-J3 に用いられたコンピュータは AMD Athlon XP 2200+/1797MHz, 512MB, Linux OS であり，本論文に用いられたコンピュータより 2 倍弱速い．

モデル生成システムに向きの充足可能問題 EPS 部門に対して，R-SATCHMO は良い性能を示した (R-SATCHMO は

☆ http://www.cs.miami.edu/~tptp/
☆☆ http://www.easychair.org/FLoC-06/IJCAR.html
☆☆☆ http://www.cs.miami.edu/ tptp/CASC/J3/

表 1 実験結果 (I)

実行時間 (秒)	≤ 2	≤ 5	≤ 10	≤ 50	≤ 100	≤ 200	≤ 300
本アルゴリズム	637	663	681	691	694	707	709
前アルゴリズム	404	579	609	634	642	646	654

表 2 実験結果 (II)

節集合の引数の総数	問題数	本アルゴリズム	前アルゴリズム
0 ～ 10000	642	1.43 秒	3.66 秒
10001 ～ 20000	8	2.07 秒	21.45 秒
20001 ～ 30000	5	2.43 秒	455.64 秒
30001 ～ 40000	0	該当なし	該当なし
40001 ～ 50000	9	4.27 秒	4518 秒
50001 ～ 60000	1	6.56 秒	>10000 秒
60001 ～ 70000	10	7.35 秒	>10000 秒
70001 ～ 80000	3	8.42 秒	>10000 秒
80001 ～ 200000	5	13.07 秒	>10000 秒
200001 ～ 500000	9	31.18 秒	>10000 秒
500001 ～ 1000000	15	147.46 秒	>10000 秒
1000001 ～ 1600000	2	244.34 秒	>10000 秒

表 3 実験結果 (III)

Problems	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	*No.*	R
PUZ018-2	0.0	0.1	0.1	0.0	3.4	0.0	T.o.	T.o.	G.u.	2	0.1
SYN307-1	0.0	0.0	0.0	0.0	0.0	0.0	T.o.	T.o.	G.u.	2	0.1
SYN317-1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	G.u.	2	0.1
SYN419-1	0.1	1.1	0.1	0.6	59.1	T.o.	1.1	T.o.	G.u	2	0.4
SYN423-1	0.1	1.9	11.2	1.1	T.o.	T.o.	T.o	T.o.	G.u.	2	0.6
SYN521-1	0.0	0.1	0.0	0.1	0.5	0.0	0.0	0.0	G.u.	4	0.1
SYN534-1	0.0	0.1	0.0	0.1	0.1	0.0	0.0	0.0	G.u.	2	0.1
SYN539-1	0.0	0.2	0.0	0.2	0.5	0.0	1.2	1.5	G.u.	2	0.8
SYN541-1	0.0	0.1	0.0	0.2	0.5	0.0	T.o.	0.2	G.u.	2	0.5
SYN823-1	0.5	1.6	0.7	15.0	T.o.	284.4	0.2	0.4	G.u.	128	14.2
SYN872-1	0.5	0.6	1.2	179.8	T.o.	T.o.	0.3	0.5	G.u.	62	67.5
SYN888-1	0.8	1.8	1.9	281.7	T.o.	T.o.	0.4	0.6	G.u.	64	104.6

Prolog 言語で実装されていることに留意). 結果を表 3 に示す (実行時間は秒である). R-SATCHMO 以外のデータは CASC-J3 によるものである. また, P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9 と R はそれぞれ *Darwin1.3*, *DCTP10.21p*, *DarwinFM1.3*, *Paradox2.0a*, *Geo2006i*, *iProver0.1*, *E0.99*, *Vampire8.1*, *Equinox1.0a* と R-SATCHMO を表し, *No.* はその問題において導出した *SHU* の数であり, G.u. は give-up, T.o. は time-out をそれぞれ表す.

なお, SYN419-1, SYN423-1, SYN823-1, SYN872-1 と SYN888-1 はサブエルブラン領域により R-SATCHMO が解決した問題である. その中, SYN823-1, SYN872-1 と SYN888-1 は本論文の手法により制限時間内に解決したものである.

5. おわりに

本稿では, 効率的なサブエルブラン領域を計算するアルゴリズムを提案した. 自動定理証明のベンチマーク問題を用いて提案手法の有効性を確認した.

今後の課題として, 本手法と不等式処理手続きを取り込んだ R-SATCHMO を C 言語または C++言語で実装し, 国際定理証明システムコンテストに出場しチャレンジしたい.

参考文献

1) He, L.: 'I-SATCHMO: an Improvement of SATCHMO', *J. of Automated Reasoning*, **27**, pp.313-322 (2001).
2) He, L., Chao, Y. and Itoh, H.: 'R-SATCHMO: Refinements on I-SATCHMO', *J. of Logic and Computation*, **14**, pp.117-143 (2004).
3) 何 立風, 巣 宇燕, 川那 宜充, 加藤 昇平, 中村 剛士, 伊藤 英則：サブエルブラン領域の特定およびモデル生成定理証明への応用, 情報処理学会論文誌, Vol.45, NO.5, pp.1335-1344 (2004).
4) Manthey, R. and Bry, F.: 'SATCHMO: a theorem prover implemented in Prolog', *Proceedings of 9th intl. Conf. on Automated Deduction*, Argonne, Illinois, USA, pp.415-434 (1988).
5) Schutz, H. and Geisler, T.: 'Efficient Model Generation through Compilation', *Information and Computation*. 162, pp.138-157 (2000).
6) Morales, J., Carro, M. and Hermenegildo, M.: Improving the Compilation of Prolog to C Using Type and Determinism Information: Preliminary Results. *Colloquium on Implementation of Constraint and LOgic Programming Systems (ICLP associated workshop)*, pp.89-102 (2003).