1E1-R-1

結論発見システム SOLAR の分割統治法による高速化

A divide and conquer strategy for consequence finding system SOLAR

寄特勇紀[*1] 鍋島英知[*2]
Yuki Kidoku Hidetomo Nabeshima

[*1]山梨大学大学院医学工学総合教育部コンピュータ・メディア工学専攻
Computer Science and Media Engineering, Department of Education Interdisciplinary
Graduate School of Medicine and Engineering, University of Yamanashi

[*2]山梨大学医学工学総合研究部
Department of Research Interdisciplinary Graduate School of Medicine and Engineering, University of Yamanashi

In this paper, we propose a divide and conquer strategy for SOLAR which is a consequence finding system. SOLAR is efficient implementation of SOL tableau calculus. In general, SOL tableau calculus sometimes causes the repeated generation of sub-tableaux to enumerate consequences. To avoid such generation, we divide a tableau into sub-tableaux, then solve them independently and finally merge sub-solutions into the original solutions. The merge process is efficiently computed by a trie structure of sub-solutions. We show the effectiveness of the proposed approach by the experimental results.

1. はじめに

本稿では，一階述語論理の結論発見システム SOLAR [1] の分割統治法に基づく高速化手法を提案する．公理集合から自明でない興味深い結論を導出することを結論発見と呼ぶ．結論発見は演繹・帰納・発想推論を実現する手段として，デフォルト推論 [2]，知能コンパイル [3] など多くの応用分野に適用できる有用な枠組みである．その中で最もよく知られている手続きの一つが SOL 導出 [4] であり，SOL タブロー計算法 [5] は SOL 導出をタブロー計算法で再定式化したものである．そして，SOL タブロー計算法の効率的実装が SOLAR である．

SOLAR で解を探索する際は，求める結論を全て列挙するために，タブローの解き方についてあらゆる可能性を網羅する必要がある．そのためには，同じような部分タブローを繰り返し計算する必要があり，探索空間の爆発を引き起こしてしまう．

先行研究では SOLAR における部分タブローの繰り返し計算を避けるため，特殊な場合 (反駁発見) における分割統治法に基づく新しい探索戦略を提案し，その正当性を証明した [6].

本稿では SOLAR の分割統治法を，結論発見に拡張する方法を提案する．結論発見への拡張では，部分問題の解を単純に統合するだけでは得られない解を補完する必要がある．さらに，部分問題の解の指数関数的な組み合わせを試行する必要がある．本稿では特に，後者の問題に対して，部分問題の解集合をトライ構造で表現して統合することで，解集合を統合する際の重複計算を削減する方法を提案する．また，計算量のシミュレーションによって本手法が効果的であることを示す．

2. SOL タブロー計算法 [5]

本章では SOL タブロー計算法とその問題点を述べる．

2.1 SOL タブロー計算法の定義

まず諸定義を行う．述語とその否定を**リテラル**という．また，リテラルの選言を**節**という．**生成領域** $\mathcal{P}$ とは求めたい結論 (**特徴節**という) を表すバイアスであり，リテラルの集合 L と特定の条件 $Cond$ によって定義される．ある節 C に含まれる全てのリテラルが L に含まれ，かつ C が条件 $Cond$ を満たしているとき，C は $\mathcal{P}$ に属するという．また，節集合 Σ に対して，Σ の論理的帰結で生成領域 $\mathcal{P}$ に属する節の集合を $Th_{\mathcal{P}}(\Sigma)$ と表す．また，$\mu\Sigma$ は Σ 中の任意の節により真に包摂されない節集合を表す．節集合を Σ，生成領域を $\mathcal{P}$ とすると，$\mathcal{P}$ に関する Σ の特徴節集合 $Carc(\Sigma, \mathcal{P})$ を $\mu Th_{\mathcal{P}}(\Sigma)$ と定義する．すなわち，生成領域に属する節で包摂のもとで閉じている節が特徴節である．

節タブロー T とはラベル付き順序木であり，T の根以外の全てのノードはリテラルでラベル付けされている．本稿ではリテラルとノードを同一視する．あるノード直下のリテラルが $L_1, ..., L_n$ であるとき，節 $L_1 \vee ... \vee L_n$ を**タブロー節**と呼ぶ．根の直下のタブロー節を**先頭節**と呼ぶ．**連結タブロー**とは，根以外の全ての葉でないノード L が直下の子として $\neg L$ を持つ節タブローをいう．**マーク付タブロー**とは，いくつかの葉に *closed* または *skipped* で印付けされた節タブローである．マーク付けされていない葉は**サブゴール**と呼ぶ．T が**解けた**とは，T の全ての葉ノードがマーク付けされている場合をいう．$skip(T)$ を *skipped* でマーク付けされたすべてのリテラルの集合とする．選択関数 ϕ は，タブローからサブゴールへの写像である．

代入とは，変数と項の関係付けの有限の集合である．例として以下を考える．

$$\{X/A,\ Y/b,\ Z/f(c)\}$$

この例では，変数 X に変数 A，変数 Y に定数 b，変数 Z に関数 $f(c)$ が関係付けられている (代入されている)．ここで，項が変数に関係付けられることを**束縛する**といい，項によって束縛された変数を**束縛変数**という．変数は高々一つの項と関係付けられ，項と関係付けられた変数はその代入のどの項にも現れてはならない．

SOL タブロー計算法を以下で定義する．Σ を節集合，C を節，$\mathcal{P}$ を生成領域，ϕ を選択関数とする．$\Sigma + C$ と $\mathcal{P}$ からの ϕ による節 S の **SOL 演繹**は，次の条件を満たすタブローの

連絡先: 山梨大学大学院医学工学総合教育部コンピュータ・メディア工学専攻，〒 400-8511 山梨県甲府市武田 4-3-11, E-mail：kidoku@nabelab.org

列 $T_0, T_1, ..., T_n$ である．

1. T_0 は，先頭節 C のみからなるタブローである．
2. T_n は，解けたタブローである．
3. 各 T_i において，$skip(T_i)$ は $\mathcal{P}$ に属す．
4. T_{i+1} は，以下の手順で導出する．まず ϕ によりサブゴール K を選択する．次に，以下のいずれかの操作を適用し，T_{i+1} を得る．
   (a) **Skip:** もし，$skip(T_i) \cup \{K\}$ が，$\mathcal{P}$ に属するなら，K に *skipped* とラベル付けする．
   (b) **Factoring:** もし，$skip(T_i)$ がリテラル L を含み，K と L が θ によって単一化可能ならば，K に *skipped* とラベル付けし，T_i に代入 θ を適用する．
   (c) **Extension:** $\Sigma \cup \{C\}$ から節 B を選択し，変数の名前変えをした亜種を $B' = L_1 \vee ... \vee L_m$ をとする．もし $\neg K$ とリテラル L_j が代入 θ によって単一化可能ならば，サブゴール K に新しい子 $L_1, ..., L_m$ を付加し，L_j には *closed* をラベル付けする．得られたタブローに代入 θ を適用する．
   (d) **Reduction:** もし，K が祖先ノードに L を持ち，$\neg K$ と L が θ によって単一化可能ならば，K に *closed* とラベル付けし，T_i に θ を適用する．

SOL タブロー計算法の正当性は次の定理で示される．

定理 1. [5]
健全性: もし $\Sigma + C$ と $\mathcal{P}$ からの ϕ による節 S の SOL 演繹が存在するのならば，S は $Th_{\mathcal{P}}(\Sigma \cup \{C\})$ に属する．
完全性: もし節 F が $Th_{\mathcal{P}}(\Sigma)$ には属さないが $Th_{\mathcal{P}}(\Sigma \cup \{C\})$ に属する場合，F を包摂する節 S の $\Sigma + C$ と $\mathcal{P}$ からの ϕ による SOL 演繹が存在する．

2.2 問題点：部分タブローの繰り返し計算

SOL タブロー計算法では，タブローの分岐を順番に解いていく．例として図 1 を考える．なお，三角形は解けた部分タブローを表すものとする．また，タブローの分岐は左から解かれていく (選択関数 ϕ は最左のサブゴールを選択する) ものとする．ここで，分岐 P_1 から分岐 P_5 までの全ての分岐を解くことができた場合，タブロー全体が解けたことになり，結論 $skip(T)$ を出力する．次に，別の解を探索するため P_5 の分岐に対して行った探索をリセットし，別の部分結論を求めて探索する．このような処理を繰り返し，分岐 P_5 について全ての可能性を網羅的に探索した後，分岐 P_4 までバックトラックして分岐 P_4 について別の部分結論を探索する．その後，再び分岐 P_5 について全ての部分結論を求めていく．これは，P_4 の解き方によって P_5 の解が異なってくることがあるからである．以上の様な処理を，分岐 P_1 から分岐 P_5 の全ての解き方の組み合わせを試すまで繰り返す．ここで，分岐 P_1 から分岐 P_5 がそれぞれ 10 通りの解き方を持つと仮定すると，最悪で 10^5 通りの解き方を試す事になる．この様に，SOL タブロー計算法では分岐が増えると考慮すべきタブローの解き方が増加し，繰り返し計算によって探索空間が指数関数的に爆発してしまう可能性がある．

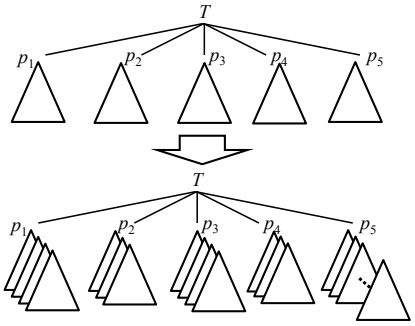

図 1: 探索空間の爆発

3. SOL タブロー計算法における分割統治法

2.2 節で紹介した部分タブローの繰り返し計算を回避するため，SOL タブロー計算法に分割統治法を適用する．まず，タブローの分岐を分割してそれぞれを独立して探索し，各サブタブローから得た部分問題の解を統合して全体の解を得る．ここで部分問題の解は，部分結論と代入によって表すことができる．図 1 を例として考えると，まず P_1 から P_5 までの分岐を 5 つに分割する．次に，分割された 5 つの分岐をそれぞれ独立して探索し，5 つの部分問題の解集合を得る．そして，それらの解集合を統合することで全体の解集合を得る．

従来の SOL タブロー計算法では，バックトラックを繰り返しながら，各サブゴールを何度も繰り返し解き直す必要がある．一方，分割統治法を適用した場合，各サブゴールを一度全解探索するだけでよくなり，同じサブゴールを何度も解き直すことを避けることが可能になる．その結果，サブゴールの繰り返し計算による探索空間の爆発を回避することができる．しかしながら，分割統治を行わない場合は代入が適用された特殊なサブゴールを解くのに対し，分割統治法を適用した場合は最も一般的なサブゴールを解くことになる．よって，サブゴールを探索する際の探索空間が増大してしまう可能性がある．

3.1 反駁発見問題における分割統治法

SOL タブロー計算法に分割統治法を適用する場合，部分問題の解の合成として代入と部分結論の合成を考える必要がある．そこで，まず結論発見問題の特殊例である反駁発見問題の分割統治アルゴリズム [6] を提案し，正当性を証明した．反駁発見問題は，公理集合が矛盾するかの判定問題であり部分結論は存在しないため部分問題の解の統合では代入のみを考える．反駁発見問題における分割統治法を評価した結果を表 1 に示す．反駁発見問題における評価実験は，SOL タブロー計算法を実装した結論発見システム SOLAR [1] に分割統治法を実装して行った．ベンチマーク問題として TPTP-v3.5.0 を用い，通常の SOLAR と解けた問題数を比較した．実験に使用したマシンは Mac mini(Core 2 Duo 1.83GHz，2GB RAM) である．また，1 問あたり 1 分の制限時間を設けた．実験は，分割する分岐の深さや分岐数が結果に影響するか調べるため以下の 4 種類を行った．

- 深さが 1 以下の分岐（先頭節）を分割.
- 深さが 2 以下の分岐を分割.
- 全ての分岐を分割.
- 長さが 4 以上である全ての分岐を分割.

表 1: 反駁発見問題における単解探索での分割統治法の評価

	従来手法	分割統治法			
	SOLAR	depth ≤ 1	depth ≤ 2	all nodes	all nodes len ≥ 4
♯ of solvedunit EQ	162	162	154	141	162
♯ of solved non-unit EQ	696	656	647	647	666
♯ of solved non-EQ	1249	1249	1228	1173	1244
Total	2107	2067	2029	1961	2072

表 2: 反駁発見問題における全解探索での分割統治法の評価

	従来手法	分割統治法			
	SOLAR	depth ≤ 1	depth ≤ 2	depth ≤ 3	depth ≤ 4
♯ of solvedunit EQ	115	165	165	165	165
♯ of solved non-unit EQ	453	623	630	628	628
♯ of solved non-EQ	1074	1188	1174	1175	1173
Total	1642	1976	1969	1968	1966

反駁発見問題における分割統治法を評価した結果を表 1 に示す．表 1 は，それぞれの条件でどれだけの問題を解くことができたかを表している．各列が問題を解く条件であり，最も左の列が SOLAR，右の 4 列が分割統治法である．各行は問題の種類を表し，unit EQ は単位節のみで構成された等号を含んだ問題である．non-unit EQ は unit EQ 以外の等号を含んだ問題である．non-EQ は等号を含まない問題である．Total は解けた問題の総数を表している．表 1 より，どの条件でも分割統治法のほうが解けた問題数が少なく，特に，タブローの分岐を分割するほど解けた問題が少ない．このことから，タブローの分岐を過剰に分割することは効率低下を引き起こすと考えられる．さらに，本実験は単解探索で行なっている．本来なら，解を一つ発見した時点で処理を停止することができるが，分割統治を適用した場合，各サブゴールについて全解探索を行った後に部分問題の解集合を統合するため必ず全解探索を行う．このことも，解けた問題数が少ない原因と考えられる．

しかしながら，本研究の目標は結論発見問題に分割統治法を適用することである．そこで，結論発見問題における分割統治法の性能を推定するため，反駁発見問題を全解探索する追実験を新たに行った．反駁発見問題を全解探索した場合の評価結果を示す．実験には，上記で用いたソルバーを全解探索仕様にしたものを用いた．ベンチマーク問題は TPTP-v3.5.0 を用い，実験に使用したマシンは Mac mini(Core Duo 1.66GHz, 2GB RAM) である．また，1 問あたり 1 分の制限時間を設けた．さらに，分割統治法ではタブローを分割する深さを 1，2，3，4 とした 4 つの条件で実験を行った．実験結果を表 2 に示す．表 2 より，どの条件でも分割統治法のほうが多く問題を解いていることが分かる．この結果より，全解探索を行うことによる探索空間の爆発が分割統治法によって軽減することができると考えられる．

なお現在，よりシンプルで効率の良い代入統合アルゴリズムとして Martelli らの単一化アルゴリズム [7] の利用を検討しており，今後実装して評価を行なっていく予定である．

4. 結論発見問題への拡張の検討

分割統治法の結論発見問題への拡張を考えた場合，考慮すべき点が二つ存在する．一つは，完全性の確保である．結論発見問題において，タブローを分割して探索を行った場合，二つの部分タブロー間で発生する Factoring 操作を再現し，全ての結論を求める必要がある．二つ目は，部分問題の解集合の合成が指数関数的な組み合わせになることである．本稿では，後者の問題を改善するため，部分問題の解の組み合わせを削減する方法を提案する．

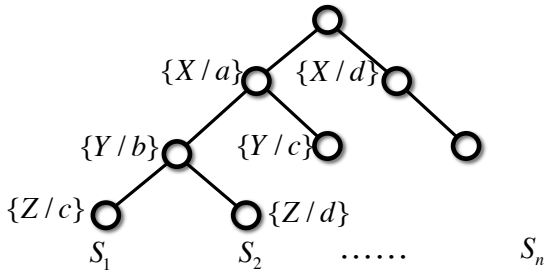

図 2: 解集合のトライ表現

4.1 解集合のトライ構造による表現

部分問題の解の組み合わせを削減する方法として，解集合のトライ構造による表現を提案する．

SOL タブロー計算法では，ある頂点に適用可能な推論規則は複数存在する．よって一般的に，求められた解集合の中には，探索において途中まで全く同様にタブローを解き，途中のある頂点から別の解き方を行うことで求められた解が複数含まれる．そして，途中までは同様の解き方で求められているため，このような解同士は互いに共通する要素を持つはずである．ここで，ある二つの解集合同士の統合において，二つの解集合の要素の全ての組み合わせを試行すると，上記の共通部分の統合が重複して行われることになる．例として，あるサブタブローを閉じて得た解集合 Σ_1 に以下のような解 S_1，S_2 が含まれている場合を考える．

表 3: 計算量の比較

Representation	Comprexity
Set	$O(pdb^{2d} + (1-p)qb^{2d})$
Trie	$O(md + md\log d + md^2 + pb^{2d} + (1-p)qb^{2q})$

$$S_1 :< \{p(a), q(b), r(c)\}, \{X/a, Y/b, Z/c\} >$$
$$S_2 :< \{p(a), q(b), s(d)\}, \{X/a, Y/b, Z/d\} >$$

S_1 と S_2 のうち，$< \{p(a), q(b)\}, \{X/a, Y/b\} >$ が同様の解き方で求められた部分である．ここで，別のサブタブローから得た解集合 Σ_2 に S_3 という解が含まれているとする．そして，Σ_1 と Σ_2 の統合を行う場合，S_1 と S_3，S_2 と S_3 の統合が行われることになる．このとき，$< \{p(a), q(b)\}, \{X/a, Y/b\} >$ の部分は共通しているため，S_1 と S_3，S_2 と S_3 の統合で重複計算が発生する．よって，S_1 と S_3，S_2 と S_3 の統合をどちらも行うのは効率的でない．

そこで，部分問題の解の組み合わせを削減する方法として，解集合をトライ構造で表現して統合する方法を提案する．トライ構造は，根から葉に至るパスの一つ一つがそれぞれ解を表しており，あるタブローに対して全解探索を行ったすべての解は，根からいずれかの葉までのパスをたどることで得ることができる．例として，上記の解集合 S_1 と S_2 をトライ構造で表すと図 2 のようになる．二つのトライ構造を統合する際は，根から葉まで順番にノード同士を合成していく．このような構造で解集合を表現することで，あるノード同士での統合が失敗した場合や，ある解同士の統合に成功した際には，バックトラックして兄弟ノードが統合可能かどうかを調べるだけで良くなり，部分問題の解の組み合わせ数を削減することが可能である．

4.2 計算量の比較

部分問題の解集合をそのままで統合する場合と，提案手法であるトライ構造で統合する場合の計算量をシミュレーションを行なって比較した．両手法における計算量を表 3 に示す．シミュレーションでは，トライ構造は分岐数が b，解の長さを d，としている．また，解の統合に成功する確率を $p(0.0 \leq 1.0)$ とし，統合が失敗した場合のトライの深さを q，解集合の要素数を m としている．解集合からトライ構造を作成する場合，変数に代入されている項の種類数を調べるために $O(md)$，変数の出現順序をソートするために $O(md\log d)$，トライ構造の構築のために md^2 の計算量がかかるがトライ構造を用いた統合は，最も悪い場合 $(p = 1.0)$ で d 倍，最も良い場合だと d^b 倍速いため，影響は少ないと考えられる．表 3 に基づいて $b = 5$，$d = 10$，$q = 5$，$m = 10$ とした場合における，統合の成功確率 p ごとの性能向上率の推移を図 3 に示す．図 3 は，縦軸が性能向上率，横軸が解の統合に成功する確率を表している．図 3 より，統合に失敗しやすいほど計算効率が高くなり，最も悪い場合でも，10 倍程度計算効率が上昇している．

ここで，解のトライ表現を作成する方法として，解集合からトライを作成するのではなく，SOL タブロー計算法の探索空間であるタブロー探索木を利用した場合を考える．この場合，解集合からトライ構造を作成する処理が不要になる．しかし，タブロー探索木を利用した場合は二つのトライにおける変数の出現順序が同一とは限らないため，トライ同士を統合する際に枝刈りができなくなってしまい，計算量は解集合を集合で統合する場合と変わらない．よって，解集合からトライ構造を作成したほうが良いと考えられる．以上より，部分問題の解をトライ構造で表現して統合する手法による効率化が見込める．

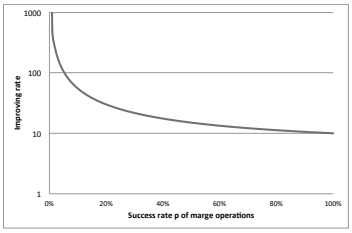

図 3: 性能向上率の推移

5. まとめと今後の課題

本研究では，SOL タブロー計算法の分割統治法による効率化を提案した．また，予備実験の結果より，結論発見問題に対する分割統治の有用性を示した．さらに，分割統治法の結論発見問題への拡張方法を提案した．

解集合をトライ構造で表現して統合することで，部分問題の解の組合わせ数を削減できるが，トライ構造を統合する効率的なアルゴリズムはまだ考慮できていないため，トライ構造を統合するアルゴリズムの調査，検討が必要である．また，完全性を確保するための効率的な方法の検討も今後の課題である．

謝辞

本研究の一部は，科学研究費補助金基盤研究 (A)(No.20240016) ならびに若手研究 (B)(No.23700164) による．

参考文献

[1] Hidetomo Nabeshima, Koji Iwanuma, Katsumi Inoue, and Oliver Ray, SOLAR: An automated deduction system for consequence finding, *AI Communications*, Vol. 23, No. 2-3. pp.183-203, 2010.

[2] Katsumi Inoue, Koji Iwanuma, and Hidetomo Nabeshima. Consequence finding and computing answer with defaults. *Journal of Intelligent information system*, Vol. 26, No.1, pp. 41-58, 2006.

[3] Alvaro del Val. A new method for consequence finding and compilation in restricted languages. *Proceedings of AAAI-99*,pp. 259-264, 1999.

[4] Katsumi Inoue, Linear resolution for consequense finding. *Artificial Intelligence*, Vol.56,pp.301-353, 1992.

[5] Koji Iwanuma, Katsumi Inoue, and Ken Satoh, A Tableau-based Reconstruction of Consequence Finding Procedure SOL. *IEICE technical report*, Vol. 100, No. 321, pp.17-24. 2000.

[6] 寄特勇紀, 鍋島英知, 結論発見手続き SOL タブロー計算法の分割統治法に基づく効率化, *The 25th Annual Conference of the Japanese Society for Artificial Intelligence*, 2011.

[7] Alberto martelli, and Ugo Montanari, An Efficient Unification Algotirhm, *ACM Transactions on Programing Langurages and Systems*, Vol.4, No.2, pp.258-282, 1982.