情報処理学会論文誌　プログラミング　Vol.11　No.2　9–19 (June 2018)

# 反復的な試行にもとづくSMTソルバ実行戦略の検討

高野 保真[1,a)]　千代 英一郎[1,b)]

受付日 2017年12月19日, 採録日 2018年3月17日

**概要**：近年，背景理論付き SAT（Satisfiability Modulo Theories，SMT）技術の発展にともなって，ハードウェア・ソフトウェアの検証など様々な問題を SMT の制約式として表現して利用する試みがさかんである．それにともない，SMT ソルバの高速化に対する需要が高まり，数多くの研究が高速化を目指している．多くの制約式を効率的に解けるようになったものの，与える制約式によっては現実的な時間で解を求めることができず，そのような場合にはユーザが得られる情報はほとんどないことが問題となっている．さらに，個々の SMT ソルバの実装は大変複雑になっており，制約式を変更した際の影響が予想しにくいため，情報を得るためにユーザが制約式を変更するのも難しい．そこで本論文は，与えられた制約式に対して制約を緩和した制約式を生成し，反復的に SMT ソルバを利用することで，反例を蓄積する SMT ソルバの実行戦略を示す．本論文で蓄積する反例は，充足不能であると判定された部分割当て，もしくは，時間内に判定できなかった部分割当てである．そのため，打ち切り時間を設けて SMT ソルバをブラックボックスとして利用するアルゴリズムを提案し，SMT-LIB 形式の制約式を対象とするプログラムを実装した．個々のソルバの最適化方法には立ち入らないことで，SMT ソルバ内部の探索方法によらない変数選択，制約の緩和方法など実行戦略を検討する．本論文では，いくつかの制約の緩和方法について，実行時間を比較したので，その結果を示す．

キーワード：SMT ソルバ

# Execution Strategy of SMT Solvers with Repetitive Trials

Yasunao Takano[1,a)]　Eiichiro Chishiro[1,b)]

Received: December 19, 2017, Accepted: March 17, 2018

***Abstract:*** Recently, Satisfiability Modulo Theories (SMT) technology has been developed actively, and there are many attempts to express and use various problems such as verification of hardware and software as SMT expressions. With the demand for speeding up of SMT solver is increasing, many studies have been worked on the design and implementation of search algorithms of SMT solvers. Although the performance of solvers that judge the satisfiability of expressions has been remarkably improving, in some cases, it is not possible to obtain a solution in a realistic time. In such a case, because the implementation of individual SMT solvers is very complex, it is difficult for the user to change the SMT expression in order to obtain information. In this paper, we propose an execution strategy of SMT solvers with repetitive trials. In each trial with a certain predetermined limited time, we relax constraints and changing the range of constraints to maximize the information finally obtained by using SMT solver. A characteristic feature of our strategy is regarding the SMT solvers as a black-box and does not go into the optimization method of individual solvers. In this paper, we compared the execution times for some relaxing methods.

***Keywords:*** SMT Solvers

## 1. はじめに

近年，背景理論付き SAT（Satisfiability Modulo Theories，以下 **SMT**）技術の発展はめざましく [18], [19]，幅広い分野で利用されるようになってきている．たとえば，

[1] 成蹊大学理工学部情報科学科
Department of Computer and Information Science, Faculty of Science and Technology, Seikei University, Musashino, Tokyo 180–8633
[a)] yasunao-takano@st.seikei.ac.jp
[b)] chishiro@st.seikei.ac.jp

ソフトウェアやハードウェアが満たすべき仕様を SMT の制約式で表現し，その制約式の充足性を判定することにより，仕様を静的に検証できる．SMT の制約式は，整数，等号，算術演算，ビットベクタなどに関する背景理論を使って記述できるため，表現力が高く，SAT の制約式に帰着するよりも元の問題の記述を保ったまま表現できるという特徴がある．

そのような表現力の高さに加えて，SMT 技術が広く利用されるようになった一因として，SMT の制約式を解くソフトウェアである**SMT ソルバ**の発展があげられる．特に実行効率の向上が著しく，多くの制約式を実用的な時間で解くことが可能となった．SMT ソルバは，命題論理を扱う SAT ソルバと，背景理論を解くための個別のソルバを組合せ，制約式中の変数に値を割り当てることで，制約式が充足するか確認する．

数多くの SMT ソルバの研究・開発の成果から SMT ソルバの高速化が図られているものの，与える制約式によっては，現実的な時間で解を求めることができない．このようなことが無視できない確率で起こることが知られており [11]，解が得られないときには，ソルバを実行したユーザは何の情報も得られなかったこととなる．さらに，個々の SMT ソルバの実装は大変複雑になっており，制約式を変更した際の影響が予想しにくいため，情報を得るためにユーザが制約式を変更するのも難しい．

そこで本論文は，与えられた制約式に対して，制約を緩和して SMT ソルバを繰り返し利用することで，ユーザが得られる情報を増やすための実行戦略を示す．ここで情報とは，一定時間内で SMT ソルバから得られた充足不能な部分割当て，もしくは，時間内に判定できなかった部分割当ての集合のことをいう．SMT ソルバの利用時には，打ち切り時間を設けることで，時間のかかる値割当ての探索を打ち切り，最終的に得られる情報を増やすことを目指す．また，SMT ソルバをブラックボックスとして利用し，利用する個々のソルバの最適化には立ち入らない．このことで，SMT ソルバの内部の探索方法によらない制約の緩和方法などの実行戦略を検討する．本論文では，いくつかの制約の緩和方法について，実行時間を比較し，その結果を示す．

なお，本論文で対象とする制約式中の変数は，整数のみに限定する．この限定が必要な理由は，変数の値を変更して試行する際に，整数であることを利用した緩和が効果的であるためであり，3.3 節で詳しく述べる．ただし，打ち切り時間を設けて反復的に SMT ソルバを利用する手法自体には，整数であることが必須の条件ではない．また，実行時間がかかる制約式が前提であるため，主に高次の項を含む等式・不等式による制約式を扱う．

$$(0 < a_0) \wedge (0 < a_1) \wedge (0 < a_2) \wedge (0 < a_3)$$
$$\wedge (1 \leq a_1 * a_3 - a_0 * a_2 - a_2 * a_2 * a_3) \quad (1)$$
$$\wedge (0 \leq a_2 - a_1) \quad (2)$$

**図 1**　SMT の制約式
**Fig. 1**　Example of a SMT formula.

## 2. SMT

### 2.1　SMT の制約式と SMT ソルバによる判定

本論文で扱う SMT の制約式は，高次の多項式で表現された数式による不等式（等式を含んでもよい）の連言で表されていることとする．たとえば，整数をとる 4 つの変数と，6 つの不等式からなる**図 1** のような制約式を扱う．整数に加えて，限定子，未解釈関数など複数の背景理論が組み合わせた制約式でも，背後で利用する SMT ソルバが対応してさえいれば，本論文の提案手法の対象となる．

SMT ソルバは，制約式中の変数に割り当てる値の組み合せ（以降，**値割当て**）を探索する．本論文で「**解が得られる**」とは，

- 与えた制約式を満たす値割当てが存在する（**充足可能**，`sat`）
- 満たす値割当てが存在しない（**充足不能**，`unsat`）
- 結果が不明（`unknown`）

のいずれかのを SMT ソルバが返した場合をいう．また，「**解が得られない**」とは想定する時間内で SMT ソルバが終了しないことをいう．解が得られない場合には，SMT ソルバの変数選択ヒューリスティックや探索空間の刈り込みが十分でない場合が考えられる．

ここで，図 1 の制約式は，後ろ 2 つの不等式 (1) と (2) において，式 (2) で $a_1 \leq a_2$ であることと式 (1) で $a_2$ よりも $a_1$ を大きな値をとる必要がありそうであるため，`unsat` であると人目でみると予想できる．しかし，一部の SMT ソルバでは解を得ることができない[*1]．

### 2.2　SMT の値割当て

本論文は，2.1 節で述べた SMT ソルバをそのまま利用しただけでは解が得られない制約式に注目する．そのような制約式に対して，一部の変数に値を割り当てて解が得られれば，与えた制約式に対する解ではないものの，ユーザが制約式を変更する際に役立てることができる．たとえば，図 1 において，以下の部分値割当ては，式 (1) を満たさないため SMT ソルバにより `unsat` と即座に判断できると期待できる．

$$[(a_0, 1), (a_1, 1), (a_2, 1)]$$

[*1] ちなみに，この制約式は SMT-LIB ベンチマーク中の制約式を，説明のために改変したものである．代表的な SMT ソルバである Z3 [8] では解が得られず，CVC4 [1] で `unsat` と求めることができる．

このとき，変数 $a_3$ のとる値は任意の整数となる．このような値割当てを蓄積すれば，制約式中で変数の持つ意味を考慮したうえでユーザが制約式を変更できる可能性がある．また，解が得られなかった値割当ても，その変数に値を割り当てることが，SMT ソルバにおける探索で，与えた制約式の制約の緩和にはあまり寄与しないことが推定できるため，有用な場合もある．

つまり，制約式中の変数に対して値を割り当て，それが充足不能であるか解が得られないという事実を蓄積することができれば，SMT ソルバをそのまま利用するだけでは得られなかった情報が得られる．提案手法では，一定時間内で SMT ソルバから得られた充足不能な部分割当て，もしくは，時間内に判定できなかった部分割当てを「**反例**」として，蓄積する．

## 3. 提案手法

提案手法が対象とする制約式は，すでに解が得られないことが分かっているものとする．そのような制約式であっても，SMT の性質より，ある値が定まれば探索空間が大きく狭まり，反例が蓄積できると期待できる．

本章では，与えられた制約式に対して，制約を緩和して SMT ソルバを繰り返し利用する提案手法について述べる．まず，提案手法の全体の流れについて 3.1 節で述べる．続いて，個々の手順である制約の緩和方法について 3.2 節で，制約の緩和のための値割当てについて 3.3 節でそれぞれ述べる．また，得られた結果は，各変数に対する値割当ての組合せとなるため，その分析指針についても 3.4 節で述べる．

### 3.1 SMT ソルバの反復的な利用

**図 2** に提案手法の手続き `smttrial` を示す．`smttrial` の引数は制約式 $formula$，個別の判定の制限時間 $limit$ である．2.1 節で述べたように，$formula$ は不等式の連言で表現されている．

まず，補助手続き `relax` で制約式を緩和し，緩和の結果残す不等式の集合 $f_{st}$ と，使わない不等式の集合 $f_{rm}$ に分け（詳細は 3.2 節で説明する），残す不等式の集合を変えながら，以降の処理を繰り返す．`solve` は第一引数の不等式を SMT ソルバを利用して求める関数で，第二引数 $limit$ までで結果が得られなかったときは，処理を打ち切る．`solve` は，判定結果の状態（`sat`，`unsat`，`unknown`，`timeout` のいずれか）と `sat` の場合にはその値割当て $m_1$ の組である．ここで，緩和した状態で `sat` であれば（4 行目），値割当て $m_1$ を基準として，割り当てる値を変更して，得られる反例を増やす（補助手続き `attempt`）．ここで，緩和して得られた部分問題に対する値割当て $m_1$ は，与えられた元の制約式に対する反例ではないため記録しない．また，`unsat` であれば（6 行目），連言を分けた部分問題において `unsat` であるため，緩和する前の制約式でも `unsat` と分かる．緩和したにもかかわらず解が得られなければ（8 行目），取り除く制約を変えて同様に試す．取り除く制約を決める際には，元の制約式 $formula$ から改めて取り除く不等式を決める方法と，緩和済みの不等式集合である $f_{st}$ からさらに不等式を取り除く方法をとりうる．

```
1:  procedure smttrial(formula, limit)
2:      foreach (f_st, f_rm) in relax(formula, limit) do
3:          (r_1, m_1) := solve(f_st, limit)
4:          if r_1 = sat then
5:              attempt(f_st, f_rm, m_1, limit)
6:          else if r_1 = unsat then
7:              end("unsat")
8:          else if r_1 = timeout or unknown then
9:              continue
10:
11: procedure attempt(f_st, f_rm, m_1, limit)
12:     (r_2, m_2) := solve(m_1 ∧ f_rm, limit)
13:     if r_2 = sat then
14:         end("sat")
15:     else if r_2 = unsat then
16:         record that m_1 is unsat
17:         changevalue(f_st, f_rm, m_1, limit)
18:     else if r_2 = timeout or unknown then
19:         return
```

**図 2**　提案アルゴリズム
**Fig. 2**　Algorithm for repetitive trials.

補助手続き `attempt` は，$f_{st}$ で `sat` と判定できた値割当て $m_1$ について，緩和に使わなかった $f_{rm}$ で反例を増やすために利用する．値割当て $m_1$ という制限を加えて $f_{rm}$ が `sat` と分かれば（13 行目），与えられた元の制約式でも `sat` であると結論できる．`unsat` であれば（15 行目），値割当て $m_1$ が `unsat` であるという反例を記録したうえで，値割当て $m_1$ に含まれる変数を選択し，別の値に変えることで反例を増やす（詳細は 3.3 節）．

図 2 で `unknown` と `timeout` だったときに，何も試していない組合せよりは反例が増えていると考えて，最終的な結果から，`timeout` する要因となっている変数を洗い出すために利用することもできる．

以上の手続き `smttrial` を，個別の制限時間 $limit$ とは別に，全体の制限時間を設定して呼び出す．制限時間が経過した際には，補助手続きなど各処理の途中であっても打ち切ることとし，記録した反例の集合を最終的に出力する．

### 3.2 制約の緩和

与えられた制約式を，緩和の結果残った不等式と，使わない不等式に分ける．緩和方法の前提として，不等式を個別に修正することはせず，各不等式を使うかどうかのみ考えることとする．不等式を個別に修正する場合，部分的に得られた `sat` や `unsat` という判定結果を図 2 の 7 行目や 14 行目のように利用することができず，また，得られた反

例の解釈が複雑になるためである．

たとえば，以下の緩和方法が考えられる．

- ある変数を含む不等式をすべて取り除く
- 不等式を 1 つ取り除く

いずれの場合も選択する変数と不等式を複数選べば，全体の制約をより緩和したことになる．ここで，どの順番で緩和を試すか，どの変数・不等式から選ぶかが問題となる．SAT ソルバであれば，Maximum Occurences of Minimum Size（MOM's）が単純で有効であると考えられている [9]．ただ，SMT ソルバの場合，ある変数が決まったからといって，他の変数のとりうる値の範囲が狭まるとは一概にいうことができない．$0 > x + y$ で $x$ が決まったからといって，$y$ の範囲が狭められるだけで，他の不等式のほうが値割当ての際の強い制約となる場合が考えられるためである．

また，緩和のための計算に時間がかかってしまっては，制限時間内で得られる反例を増やすという本研究の目的に沿わないため，複雑な解析をすることは避ける必要がある．ここで，可能な緩和方法は有限個であり，各緩和に対して SMT ソルバを制限時間を設けて利用するため，どのような優先順位を用いるとしても，極端に狭い範囲のみが探索の対象となることはない．

以上のように，様々な緩和方法や繰返し方がありうるものの，どの方法が有効であるかは一概に議論できない．そのため，本論文では，いくつかの緩和方法を実装し，実験的に比べることで得られた反例を議論する．

### 3.3　変数のとる値の変更

図 2 のアルゴリズムでは，$f_{st}$ で `sat`，$f_{rm}$ で `unsat` と分かっている値割当て $m_1$ から変数を選択して，その変数に割り当てていた値を変更することで，反例を増やす．値割当ては，制約式中の一部または全部の変数に対して，整数値が 1 つ決まっている．

**図 3** にアルゴリズムを示す．基本的な方針は，$m_1$ から，変数を 1 つ選び，その値を変化させ，変化させた値割当てを $f_{st}$ で再び SMT ソルバで確認する．`sat` であれば（6 行目），図 2 の `attempt` と同様の処理（`changevalue` を再帰的に呼び出さないため `attempt'` としている）によって，$f_{rm}$ で $m_2$ の値割当てを加えて判定する．`unsat` であれば（8 行目），元の制約式に対しても `unsat` という反例が増える．

```
 1: procedure changevalue(f_st, f_rm, m_1, limit)
 2:     foreach (var, val) in m_1 do
 3:         for changing the value of val do
 4:             m'_1 is the same as m_1 except for var
 5:             (r_1, m_2) := solve(m'_1 ∧ f_st, limit)
 6:             if r_1 = sat then
 7:                 attempt'(f_st, f_rm, m_2, limit)
 8:             else if r_1 = unsat then
 9:                 record that m'_1 is unsat
10:             else if r_1 = timeout or unknown then
11:                 continue
```

**図 3**　値割当ての変更アルゴリズム

**Fig. 3**　Algorithm for value assignment.

図 3 の 3 行目における値の変化のさせ方は，本来であれば，$var \neq val$ という制約を追加したうえで，SMT ソルバを利用すれば，SMT ソルバが有望と考える値を決めさせることができる．もし，そのように SMT ソルバを利用することができれば，1 章の最後に述べた本論文で値を変える対象を整数に限定する必要はなくなる．しかし，予備的な実験によると，次の値を決めるためにも SMT ソルバを利用すると値を変えるための時間が無視できない長さがかかるため，整数に限定することとした．

整数に限定した場合，たとえば，以下の戦略が考えられる．

- 1 ずつ増やす
- ランダムに変える
- ランダムに増やす

まず，1 ずつ増やすと，連続で値を試すことができるため，試した組合せを分析しやすくなる．また，値が 1 増えたぐらいでは，`timeout` にならずに解が得られると期待すると，反例を増やすのに有効である．ただし，真の結果が `sat` である制約式に含まれる変数に対して，SMT ソルバが境界値の下限から値割当てを求めることを仮定している．この仮定に反した SMT ソルバを用いたとき，いくら変数の値を変えたところで `sat` が得られない可能性があることを意味するが，反例を増やせれば十分という楽観的な立場から，本研究の値の変更方法として採用できる戦略である．

ランダムに値を変える場合には下限値を仮定する問題は起こらないが，最終的に得られた反例を分析することが難しくなる．ランダムに増やす場合は，1 ずつ増した場合の間を飛ばしても，試した値割当てから変数がとりうる大まかな範囲を調べられることを期待する戦略である．やはり，得られた結果が連続ではないので，最終的な反例集合の分析が必要となる．

なお，ここで試した値割当てについては，以降の SMT ソルバの利用時に，値割当てを否定した項として反映することで，繰り返し同じ値を調べずに済ませることができる．

### 3.4　得られた結果の分析方法

提案アルゴリズムは，`unsat` と判定された値割当てを蓄積する（図 2 の 16 行目と，図 3 の 9 行目）．同様にして，`timeout` と判定された値割当てが制約式の変更に役立つことも考えられる（図 2 の 9 行目と，図 3 の 11 行目）．これらの結果から，提案手法を用いない場合に判定時間がかかる制約式に対して，ユーザに何らかの変更を加えるための情報を示すことが，最終的な目標である．たとえば，ある変数 $a_0$ に対して，$[(a_0, 1)]$，$[(a_0, 2)]$，... がすべて `unsat`

と分かれば，変数のとりうる範囲が限定できる．このような情報を得るためには，ある反例を記録する際に，その値割当てから取り除いても打ち切り時間内である変数のみに絞り込んでおく必要がある．

また，5 章で示すように，一般の制約式に対して提案手法を適用すると，大量の反例が得られるため，必要な反例を人手で見分けることができるとは限らない．蓄積した反例は，必ずしもある変数の範囲を限定できるわけではなく，値割当てとして各変数とその値の組において，`unsat` もしくは `timeout` であることに注意しなくてはならない．当然のことであるが，変数 $a_1$ と $a_2$ について $[(a_1, 10), (a_2, 20)]$ で `unsat` という反例から，$a_1$ がつねに 10 をとらないとは判断してはならない．これを考慮して，反例を分析する方法は明らかではなく，本論文では結果を得るための戦略を検討するまでで，得られた反例の分析方法については今後の課題である．

### 3.5　SMT ソルバの利用

以上のように，本論文のアルゴリズムは，SMT ソルバが内部で行う変数選択や値選択を行っている．しかし，提案手法の目的はできるだけ多くの反例を得るところにあり，本論文の単純な打ち切り時間を定めた判定を SMT ソルバ自身に実装することとはならない．また，SMT ソルバであれば有望と判断する方向への探索であっても，提案手法では時間がかかる方向への探索が打ち切られ，解が得られないという反例となる場合もある．

また，SMT ソルバを背後で利用することの意義としては，本アルゴリズムを複数の SMT ソルバに対して利用することができる点である．SMT ソルバには制約式ごとの得意不得意があり，バージョンが違うだけで求解時間に大きな差がある場合がある．そのような問題ごとのばらつきを抑えるためには，様々な SMT ソルバから判定結果が得られることが望ましい．図 2 と図 3 に示したアルゴリズム中の `solve` 手続きに複数の SMT ソルバに分ければよく，提案アルゴリズムは背後の SMT ソルバから独立している．

## 4.　実装

3 章で示したアルゴリズムを，SMT の制約式を扱うライブラリ PySMT [10] を利用し，プログラミング言語 python のプログラムとして実装した．PySMT は複数の SMT ソルバの API を背後で利用できるように設計されているため，本論文のアルゴリズムを実装するのに相応わしい．

SMT-LIB 形式 [2] の制約式を読み込み，全体の制限時間と，SMT ソルバの 1 回の利用の制限時間をそれぞれ設定する．現状対応している SMT ソルバは，Z3 [8] と CVC4 [1] である．Z3 は `"timeout"` オプション，cvc は `"tlimit-per"` オプションを使って，判定を打ち切る．なお，現状では incremental solving を利用しておらず，アルゴリズム中の `solve` ごとに API を呼び出し，新たに SMT ソルバを起動する．

実装は，ほぼ 3 章のアルゴリズムに従っているが，図 2 の 16 行目と 17 行目の間に，`unsat` と分かった値割当てに対して，さらに値割当て中の変数を減らすことができるか，打ち切り時間を設けて判定する処理を加えている．たとえば，$[(a, 1), (b, 10), (c, 20)]$ という値割当てが `unsat` であるという反例が得られたときに，それぞれの変数がない場合でも制限時間内に結果が得られるかを判定する．もし，$[(a, 1)]$ だけで制限時間内に `unsat` と分かれば，反例として十分であるためである．この処理は制約式の unsatisfiable core [5] が `unsat` だった原因の分析として有用なのと同様に，`unsat` となってしまう値割当てを提示できるようにする．制限時間を設けることで，`unsat` になるのに最低限必要な値割当てだけに絞ることはあきらめて，別の値割当てを調べることを優先する．

3.2 節で述べた緩和方法は，以下の方法を実装した．

**(a)**　ある変数を含む不等式をすべて取り除く
　**(a-1)**　ランダム
　**(a-2)**　同時に出現する変数の種類数が少ない順
**(b)**　すべての不等式に対して，ランダムな確率で，その不等式を含めるか決める

(a) では，2 つの方式で変数の優先度を決める．また，(b) の確率は，2 分の 1 としている．

緩和を繰り返す際には，元の制約式（図 2 中の $formula$）から取り除く不等式を緩和が必要になるごとに決めるように実装した．このように実装することで，(a-2) のように優先度を付ける場合，元の制約式に対して，あらかじめ優先度を計算しておけば，繰返しごとに優先度を求め直す必要がない．なお，優先度は，辞書式順序の組合せを生成して，順に候補とした．

さらに，3.3 節で述べた変数のとる値の変更方法を実装した．

(i)　1 ずつ増やす
(ii)　ランダムに変える
(iii)　ランダムに増やす

つまり，緩和方法と変数のとる値の変更方法で，9 つの組合せが可能である．

## 5.　実験

提案手法により，得られる反例を確認するため，次の 2 点について実験を行った．

- 緩和方法と変数のとる値の変更方法の比較
- 打ち切り時間の比較

背後で利用する SMT ソルバの影響が大きいと考えられるため，両方の実験とも Z3 [8]（version 4.5.1），CVC4 [1]（version 1.5）のそれぞれについて，得られた反例を集計した．

表 1　比較する組合せ
**Table 1**　Relaxing strategies of our method.

| | その変数を含む不等式を取り除く | | ランダム |
|---|---|---|---|
| | ランダムに変数を選択 | 同時に出現する変数の種類数が少ない順 | |
| 1 ずつ変える | ① | ② | ③ |
| ランダムに変える | ④ | ⑤ | ⑥ |
| ランダムに増やす | ⑦ | ⑧ | ⑨ |

実験環境は，CPU が Intel® Xeon® CPU（X3430）2.40 GHz で，メインメモリが 8 GB の PC 上で動く Linux 4.4 である．

全実験に共通のパラメータとして，問題の制限時間を 10 分とした．また，緩和方法において，ランダムに値を増やす場合には 1〜100 までの値とした．

4 章で述べた方法について，**表 1** の 9 通りについて，比較する．

### 5.1　実験セット

SMT-LIB ベンチマークの QF_NIA（2017-03-29 版）から，calypto サブセットを選んだ．calypto サブセットは Calypto Design Systems の Sequential equivalence checking を題材とした制約問題で，177 個の問題を含む．今回は，その中から，上の実験環境で 1 時間経っても結果が得られない問題を対象とした．1 時間経っても結果が得られなかった問題数は，それぞれ以下のとおりである．

- Z3：4 問
- CVC4：10 問

このうち，両者に共通の問題が 2 問である．

提案手法の性質上，背後で動く SMT ソルバが対応してさえすれば[*2]，calypto サブセットではない SMT-LIB ベンチマークの任意のサブセットを対象としてもよい．しかし，以下の 2 点により，calypto サブセットを選択した．

- 結果の分析の点から，変数が少ないものの時間がかかる問題が望ましい．
- 本手法を用いずに SMT ソルバを呼び出した場合の結果が `sat`，`unsat`，`unknown` のそれぞれである問題を含んでいるほうが望ましい．

用いた問題の変数数などの条件を**表 2** に示す．「status」は提案手法を用いずに得られる結果であり，解答としてベンチマークファイル中に記述されている．また，「用いた SMT ソルバ」は本実験環境でその SMT ソルバを用いたときに 1 時間以上かかることを示している．

### 5.2　実験 1：緩和方法と変数のとる値の変更方法の比較

緩和方法と変数のとる値の変更方法による違いを比較するため，対象とする問題の総試行回数を計測した．ここでの SMT ソルバ利用時の打ち切り時間は，5 秒である．

[*2] 厳密には，実装に用いた PySMT ライブラリが対応している必要がある．

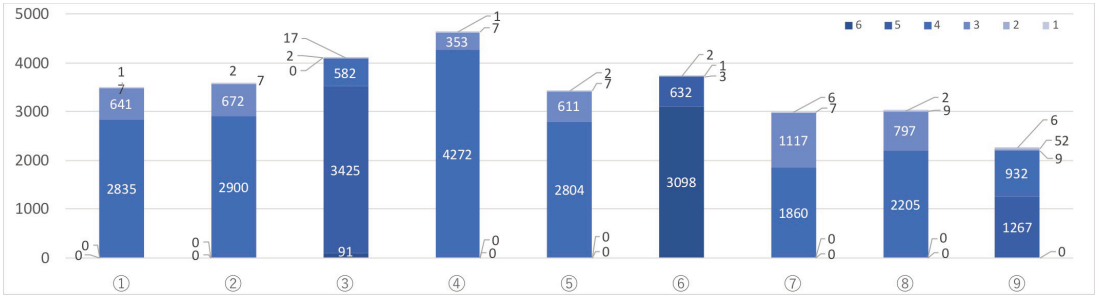
表 2　問題セット calypto
**Table 2**　calypto benchmark set.

| 問題名 | status | 変数数 | 用いた SMT ソルバ |
|---|---|---|---|
| problem-001419 | `unsat` | 6 | Z3 と CVC4 |
| problem-005612 | `sat` | 50 | |
| problem-006492 | `unknown` | 8 | Z3 のみ |
| problem-006498 | `unknown` | 10 | |
| problem-000209 | `unsat` | 5 | CVC4 のみ |
| problem-000210 | `unsat` | 5 | |
| problem-000447 | `unsat` | 5 | |
| problem-000797 | `unsat` | 6 | |
| problem-001415 | `unsat` | 4 | |
| problem-002617 | `sat` | 10 | |
| problem-002936 | `unsat` | 4 | |
| problem-002950 | `unsat` | 5 | |

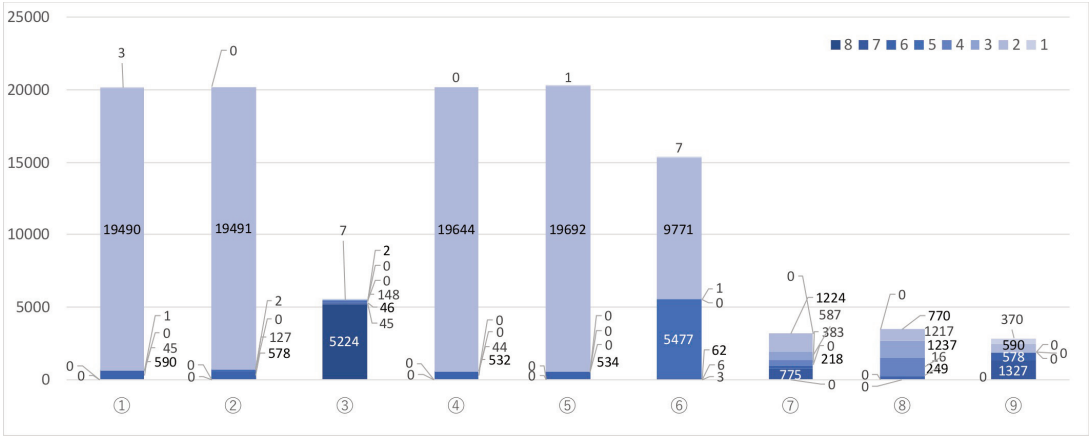
まず，Z3 の結果を示す．problem-005612 は，本手法を用いることで，9 つの組合せのいずれの場合も元問題を `sat` と判定できた．その際の判定にかかる時間は，0.52 秒〜0.67 秒であり，本手法を用いずに SMT ソルバを用いた場合は 1 時間経っても結果が得られなかったのに対して，非常に短い時間で結果が得られた．このときの得られた値割当ては，50 個の変数中，1 が割り当てられる変数が 3 個，0 が割り当てられる変数が 47 個という非常に単純な解であった．本論文は SMT ソルバをブラックボックスとして扱っているため中身にはふみ入らないが，Z3 のような質の高い SMT ソルバで解けない理由が不明である．

それ以外の 3 問の結果について，**図 4** に示す．図 4 は，横軸に表 1 の 9 つの組合せ，縦軸に得られた反例の数をとる．ここで反例の数は，図 2 の 16 行目と図 3 の 9 行目にあたる処理の回数を計測している．積み上げ棒グラフ中の 1 から 6（問題によって 8 と 10 まで）は，その問題中の変数うち，何個の変数に値の制約を加えたかを示す．たとえば，(a) の問題の ① の 2,835 は，その緩和方法において 4 つの変数に値を入れて，他の 2 つが制約なしの場合に，5 秒以内で `unsat` が得られた回数が 2,835 回であることを示す．つまり，濃い青であればあるほど，限定された組合せを試行したことを示し，薄い青であれば加えた制約が少ない状態であっても時間内で `unsat` が得られることを示す．

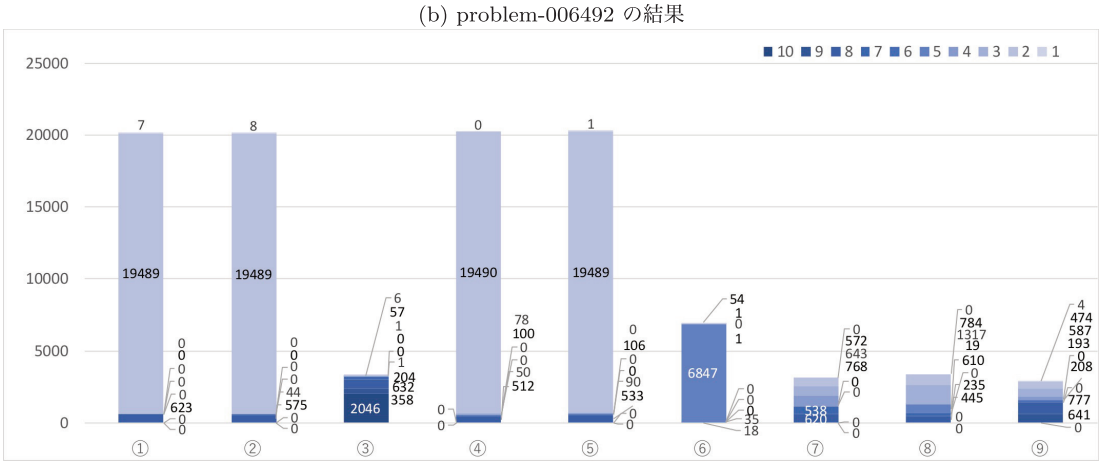
ここで，1 つの変数のみに値を割り当てる場合に注目する．たとえば，problem-001419 の ③ で 17 となっている結果では，P2 という変数に −4，−3，−2，−1，0 といった値を割り当てる反例を蓄積していた．つまり，他の変数に関係なく P2 にこれらの値が入る場合には `unsat` であると分かる．さらに，元の制約式中でこの変数は $P2 \leq 65535$ という制約が与えられているため，これらを合わせて元の制約式を解析できる．




(a) problem-001419 の結果




(b) problem-006492 の結果




(c) problem-006498 の結果

**図 4**　緩和方法と変数のとる値の変更方法の比較：Z3

**Fig. 4**　Comparison between relaxing strategies by Table 1 (Z3).

また，status が unknown とされている 2 問について，合計 20,000 程度の試行回数があったということは，少なくとも unsat という組合せが列挙されたととらえることができる．

続いてそれぞれの緩和方法を比較すると，まず，ランダムに不等式を緩和する ③，⑥，⑨ の場合には，3 問に共通して，限定された組合せを試行する傾向がある．緩和の方法から，① など他の場合と比べて，不等式が大きく取り除かれた式を優先的に調べていくことになるためであると考えられる．最終的な結果を分析する際には，unsat を期待するならば，多くの組合せを提示されるよりも加えた制約が少ない反例が提示されるほうが望ましい．

次に，1 ずつ増やす ① や ② と，ランダムな増分の ⑦ や ⑧ を比較すると，総試行回数に差がある．総試行回数が重視されるべきかは，最終的な結果をどう分析できるかに関わってくるが，calypto の 3 問の結果だけから判断すると，1 ずつ増やした場合のほうが多く，ランダムな増分で調べたときの利点はみられなかった．

また，① と ②，④ と ⑤，⑦ と ⑧ のそれぞれを比較すると，今回の 3 問に限っては緩和の基準となる変数をどういう優先度で調べるかに大きな違いはなさそうである．

続いて，CVC4 について同様の実験を行った．Z3 との比較のため，共通で 1 時間かかっていた問題（problem-001419）を図 5 に示す．

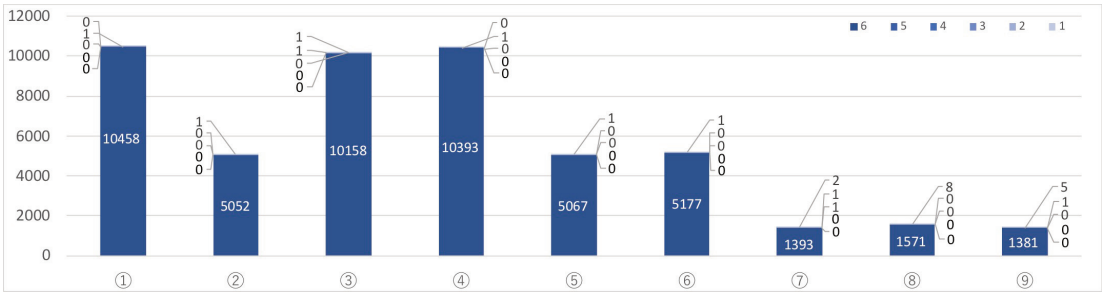



**図 5**　CVC4 の場合の problem-001419 の結果
**Fig. 5**　Results of problem-001419 (CVC4).

Z3 の傾向と大きく異なり，CVC4 ではすべての変数に値を割り当てた試行がほとんどであった．なお，calypto を対象とした CVC4 の他の結果にも同様の傾向がみられた．また，Z3 で `sat` と判定できた問題（problem-005612）は，CVC4 を用いた場合では `sat` が得られなかった．

Z3 と CVC4 で得られた結果の違いは，アルゴリズム中の $f_{st}$ で得られる値割当てによるものであると考えられる．この結果から，利用する SMT ソルバによって，緩和したときの挙動が大きく異なることが予想されるため，SMT ソルバを組み合わせて利用するなど，ユーザが必要とする情報に合わせて利用できるように今後考えていく必要がある．

### 5.3　実験 2：打ち切り時間の比較

実験 1 では，打ち切り時間を 5 秒としたが，同様の問題で打ち切り時間を 1 秒とし，計測した．**図 6** に結果を示す．

打ち切り時間によって，縦軸に取った総試行回数が，3 問ともで変わっている．これは，打ち切られた計算が増えるためで，当然の結果ではあるが，その分で，別の値割当てを試すこととで総試行回数が増えるという結果にはならなかった．一方で，① から ⑥ に比べると，⑦ から ⑨ の総試行回数の減少率は少ない．

### 5.4　実験のまとめ

以上の結果は，あくまで calypto サブセットに対する結果であるので，他のサブセットに関しても実験が必要である．

ただ，緩和方法や変数のとる値の変更方法など，さらに多くの方法が考えられるものの，ブラックボックスとして SMT ソルバを扱っても振舞いを変えることができる可能性があることは分かった．それぞれの試行回数の分布を数値化して扱うことができれば，現在の実装のままでパラメータを変えてデータをとれば，実行戦略の優劣を判断できる．また，本研究の非常に単純な手法でも，緩和によって `sat` が得られる問題があることが分かった．一方で緩和と変数のとる値の変更を繰り返すのみでは，`unsat` を求めるのは現実的ではない．すべての範囲を試せる変数であれば矛盾を導けることがありうるが，それは変数の範囲次第である．

今回の例のように元の制約式が単純であれば，蓄積した反例が多くても，反例から元の制約式を変更する際に参照することが可能であるが，制約式の変数や含まれる節が多くなると，得られた反例を利用するのが難しくなってしまう．そのため，蓄積した反例を制約式の分析に結び付ける手法が今後必要である．

## 6.　関連研究

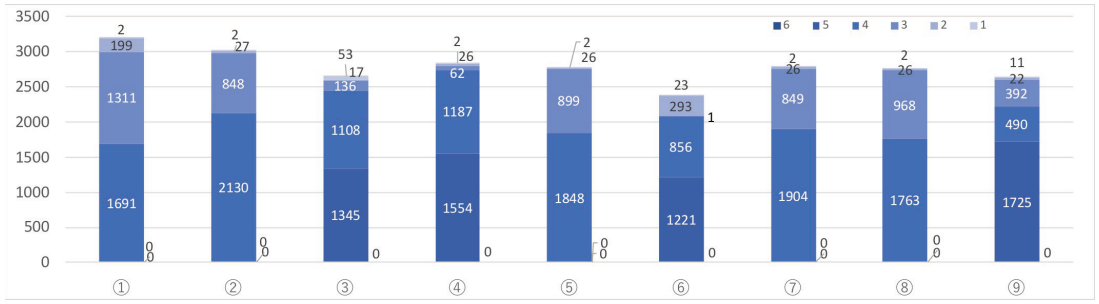
### 6.1　蓄積する情報

既存の SMT ソルバは，判定結果が `unsat` だった際に，unsatisfiable core や proof などとして充足不能性の原因となる節の部分集合（以後，合わせて **UC** と呼ぶ）を得ることができる [7]．UC が与えた制約式中の節で表現されているのに対して，2.2 節で述べたように，提案手法は `unsat` であった場合の値割当ての組合せを蓄積している．そのようにしているのは，提案手法による結果から変数がとりうる値の範囲をユーザが推定すること考慮したためである．ただ，与えた制約式によっては UC も制約式の変更に有効であり，今後は UC も合わせて蓄積することも考えられる．現状では，背後で利用する SMT ソルバで UC が得られることを前提としないために，実装上の理由から値割当ての組合せのみを保持しているが，代表的な SMT ソルバの多くは UC を出力できるため対応は容易である．対応する際には，SMT ソルバが出力する UC が必ずしも極小であるわけではなく，値割当ての組合せのみを蓄積するのに比べて保持する反例が増えるので，どのように UC を蓄積するか検討が必要である．なお，そのように UC を合わせて蓄積した場合であっても，打ち切り時間をベースとした反例の蓄積は有効である．

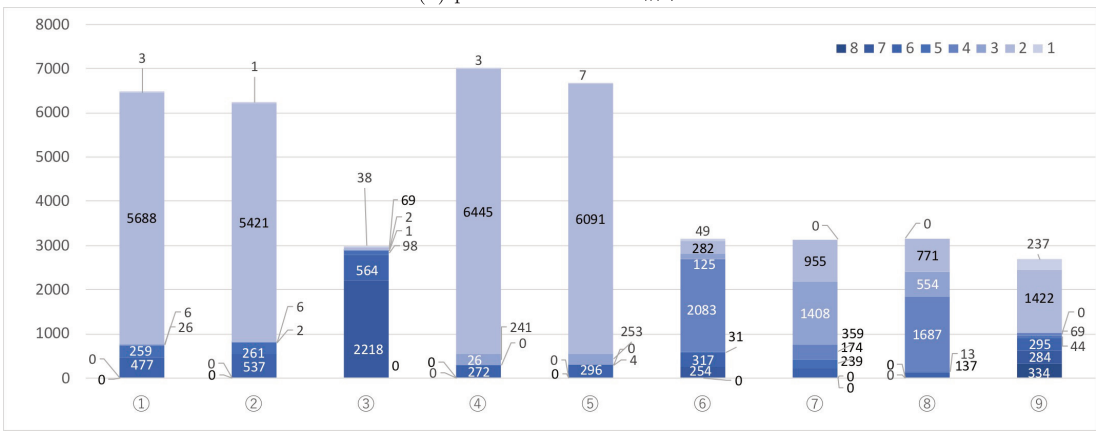
### 6.2　条件の緩和による反復的な試行

提案手法は，制約式の節を削除することで緩和し，得られた反例を反復的に蓄積する．このような抽象化を反復的に用いる手法は，モデル検査手法として一般的であり，多くの研究がなされている [3], [6], [15], [16].

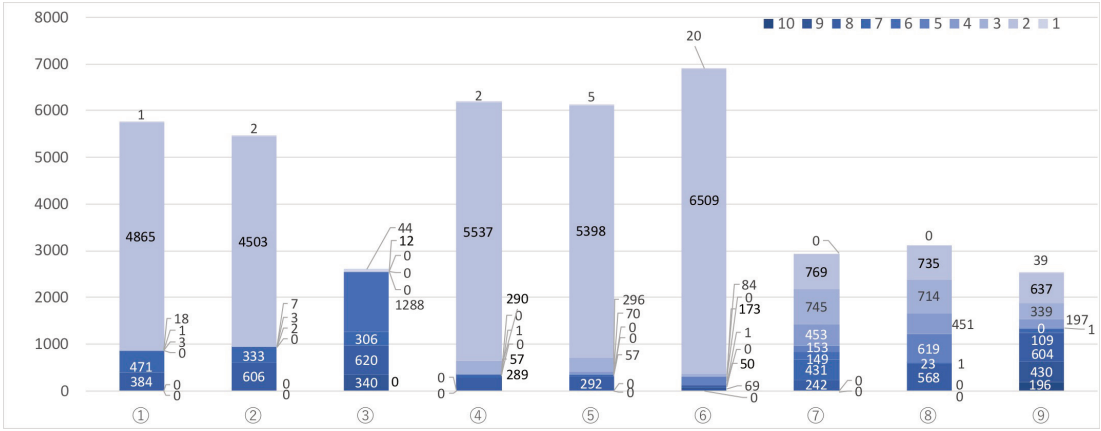
Counterexample-Guided Abstraction Refinement（CE-




(a) problem-001419 の結果




(b) problem-006492 の結果




(c) problem-006498 の結果

**図 6**　打ち切り時間ごとの比較：Z3

**Fig. 6**　Comparison between timeout periods (Z3).

GAR）[6] は，モデル検査の一般的な枠組みで，対象のモデルを過度（over-approximation）に抽象化したモデルに対する検査をまず行い，その検査で与えられた性質を満たすならば対象のモデルにも同じ性質が成り立つと判定する．over-approximation のモデルが与えられた性質を満たさないときは，その反例を用いて抽象モデルを洗練し，これを繰り返すことでモデル検査を行う．提案手法において，節を削除した制約式に対する SMT ソルバの利用を CEGAR の抽象化，その結果を利用して変数のとる値を変更していく処理を CEGAR の抽象モデルの洗練ととらえると，広い意味では CEGAR の枠組みに似ているといえる．提案手法では，洗練に相当する図 3 のアルゴリズムで値を変更させた後，別の節を取り除く緩和処理に戻る．このとき，緩和の際には単純に別の節を取り除くだけで，それまでの緩和と洗練で得られた反例を使ってはいない．それは，緩和と洗練の計算コストを抑えて，定められた時間内に調べられる反例を増やすことを目指すためである．

CEGAR と同様に抽象化を繰り返すシステムとして，eager アプローチをとる SMT ソルバ UCLID がある [3], [15]．UCLID は，制約中の変数の範囲の限定（under-approximation）と，式の一部を真理値と置き換える緩和（over-approximation）を繰り返すことで，充足性を求める．提案手法が変数に値を割り当てるのは，変数の範囲を 1 点に限定しているのと同じことであるため，抽象と緩和の基本的な点で，提案手法と UCLID は同様の方針をとっている．しかし，UCLID は under-approximation の洗練

時に変数のとる範囲を広げていくため，提案手法で別の値を 1 点で割り当てることとは異なる．提案手法が変数の範囲を制約として加えるのではなく，1 点で値を割り当てる方針をとったのは，高次の項を含む制約式において変数を範囲で与えることが十分な緩和とならず，解を得られないことが多いという経験則からである．変数の範囲による制約を，3.3 節で示した値の割り当て方法の 1 つであるととらえて，今後実装と実験による比較が必要である．

McMillan らの研究 [16] は，反例を用いて洗練する代わりに，proof を用いる．そのことで，不要な状態変数を除外し，モデル検査の効率の向上を目指している．本論文の実験の範囲では，蓄積した反例の数が問題となることはなかったが，与える制約式や打ち切り時間などの設定によっては，保持する情報量が実装上の問題となることがあり得る．その際には，6.1 節で述べたように，提案手法でも proof を蓄積し，McMillan らの研究の活用が期待できる．

なお，上記のいずれのシステムも打ち切り時間は採用しておらず，本論文が対象とする実行時間がかかることがあらかじめ分かっている制約式に対象を絞っていない．

図 2 で示したアルゴリズムは，SAT 制約式における同様の緩和方法で `unsat` を期待する場合に有効であることが示唆されている [9]．そこでは，本論文のように打ち切り時間を定めて探索するといったことまでは述べられていない．なお，`sat` を期待する場合には，節からリテラルを削除することが有望であるとされているが，本論文では不等式中の項を削除する方針はとらなかった．

### 6.3 MAX-SAT/MAX-SMT 問題

MAX-SAT/MAX-SMT 問題は，SAT/SMT 問題を拡張し，すべての節を満たす値割当てを求める代わりに，より多くの節を満たす値割当てを探索する問題である [17], [20]．典型的な解法の 1 つとして，阻止変数と呼ばれる命題変数を問題の制約式の各節に追加し，追加後の制約式をソルバを繰り返し呼び出すことで問題の最適解を求める手法がある．制約式を充足しつつ，阻止変数の数を最小にするという問題ととらえて，ブラックボックスとしてソルバを利用できる点で提案手法と類似している．

その一手法として，Cimatti らは，外部の SMT ソルバと MAX-SAT ソルバを組み合せる手法を提案している [4]．その手法では，背後で利用するソルバを選択可能である点で，提案手法が Z3 や CVC4 など SMT ソルバを切り換えて利用できること同様である．ただ，背後の SMT ソルバから $T$-lemma が得られることを前提としており，提案手法が充足性と `sat` のときの値割当てのみが必要であることと異なる．SMT ソルバから得られる判定結果と反例として蓄積する情報の間には密接な関係があるので，6.1 節のように検討が必要である．

### 6.4 打ち切りを設けた解の探索

SAT ソルバにおいて，ランダムに探索を始めからやり直すリスタート戦略が有効であることが示されている [11], [12]．短い時間でリスタートすれば，探索時間が非常に長くなってしまう場合を排除できるため，全体の探索時間から不安定さを取り除くことができる．提案手法が短い時間で SMT ソルバの探索を打ち切るのも，同様の理由からである．また，背後で利用する SMT ソルバのリスタート戦略が働いていることを考えると，二重で同じことをやっていることになってしまう．しかし，本論文の目的は得られる反例を増やすことにあるため，SMT ソルバのリスタートとは分けて扱う必要があると考えている．また，実験では，5 秒や 1 秒といった値でしか確認していないが，打ち切り時間の決定に際しても一定な値をとるのではなく，既存のリスタート戦略に関する研究を参考にして，可変の値をとる方法もありうる．

## 7. おわりに

本論文は，与えられた制約式に対して制約を緩和した制約式を生成し，緩和方法を変えながら反復的に SMT ソルバを利用することで，反例を蓄積する SMT ソルバの実行戦略を示した．打ち切り時間を設けて SMT ソルバを利用し，最終的に得られる反例を増やすことを目指した．緩和方法や変数の値の決定方法など，検討する項目を出し尽せていないものの，ブラックボックスとして SMT ソルバを扱っても振舞いを変えることができる可能性があることは分かった．また，本研究の非常に単純な手法でも，緩和によって `sat` が得られる問題があることが分かった．

今後の課題は，次のとおりである．

- 複数の SMT ソルバの組合せ
- 結果の分析
- 制限時間や変数の値の変更方法の動的な決定

1 つ目の項目は，3.5 節で示したように，SMT ソルバごとの得手不得手を埋めることを目標とする．2 つ目の項目は，現状，得られた結果から制約式の分析をする仕組みはできていない．しかし，1 つの変数に割り当てて `unsat` という反例が記録されていることからだけでも，その変数がとりえない値とすることができる．この値を連続に変えて，たとえば，この範囲の制限を加えたらよいというように提案することができれば，ユーザにとって有益である．3 つ目の項目は，`timeout` が続くようならば，制限時間や変数に割り当てる値の決定方法を動的に変更するような戦略についてである．ソルバで利用されているリスタート戦略 [12], [13], [14] を参考に，過去の制限時間などを利用できないかと考えている．

## 参考文献


[1] Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A. and Tinelli, C.: CVC4, *Proc. 23rd International Conference on Computer Aided Verification*, pp.171–177 (2011).

[2] Barrett, C., Fontaine, P. and Tinelli, C.: The SMT-LIB Standard Version 2.5 (2015), available from ⟨http://smtlib.cs.uiowa.edu/papers/smt-lib-reference-v2.5-r2015-06-28.pdf⟩.

[3] Bryant, R.E., Kroening, D., Ouaknine, J., Seshia, S.A., Strichman, O. and Brady, B.: Deciding Bit-Vector Arithmetic with Abstraction, *Proc. 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp.358–372 (2007).

[4] Cimatti, A., Griggio, A., Schaafsma, B.J. and Sebastiani, R.: A Modular Approach to MaxSAT Modulo Theories, *Proc. 16th International Conference on Theory and Applications of Satisfiability Testing*, Berlin, Heidelberg, pp.150–165 (2013).

[5] Cimatti, A., Griggio, A. and Sebastiani, R.: Computing Small Unsatisfiable Cores in Satisfiability Modulo Theories, *Journal of Artificial Intelligence Research*, Vol.40, pp.701–728 (2011).

[6] Clarke, E., Grumberg, O., Jha, S., Lu, Y. and Veith, H.: Counterexample-guided Abstraction Refinement for Symbolic Model Checking, *Journal of ACM*, Vol.50, No.5, pp.752–794 (2003).

[7] De Moura, L. and Bjørner, N.: Proofs and Refutations, and Z3, *Proc. LPAR 2008 Workshops, Knowledge Exchange: Automated Provers and Proof Assistants, and the 7th International Workshop on the Implementation of Logics*, pp.123–132 (2008).

[8] De Moura, L. and Bjørner, N.: Z3: An Efficient SMT Solver, *Proc. Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp.337–340 (2008).

[9] Freeman, J.W.: *Improvements to Propositional Satisfiability Search Algorithms*, PhD Thesis, University of Pennsylvania (1995).

[10] Gario, M. and Micheli, A.: PySMT: A Solver-Agnostic Library for Fast Prototyping of SMT-Based Algorithms, *Proc. 13th International Workshop on Satisfiability Modulo Theories*, pp.373–384 (2015).

[11] Gomes, C.P., Selman, B., Crato, N. and Kautz, H.: Heavy-Tailed Phenomena in Satisfiability and Constraint Satisfaction Problems, *Journal of Automated Reasoning*, Vol.24, No.1-2, pp.67–100 (2000).

[12] Gomes, C.P., Selman, B. and Kautz, H.: Boosting Combinatorial Search Through Randomization, *Proc. 15th National Conference on Artificial Intelligence*, pp.431–437 (1998).

[13] Horvitz, E., Ruan, Y., Gomes, C., Kautz, H., Selman, B. and Chickering, M.: A Bayesian Approach to Tackling Hard Computational Problems, *Proc. 17th Conference on Uncertainty in Artificial Intelligence*, pp.235–244 (2001).

[14] Kautz, H., Horvitz, E., Ruan, Y., Gomes, C.G. and Selman, B.: Dynamic Restart Policies, *Proc. 18th National Conference on Artificial Intelligence*, pp.674–681 (2002).

[15] Kroening, D., Ouaknine, J., Seshia, S.A. and Strichman, O.: Abstraction-based Satisfiability Solving of Presburger Arithmetic, *Proc. 16th International Conference on Computer-Aided Verification*, pp.308–320 (2004).

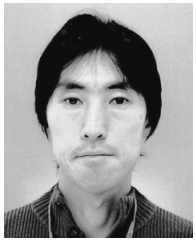
[16] McMillan, K.L. and Amla, N.: Automatic Abstraction without Counterexamples, *Proc. 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp.2–17 (2003).

[17] Nieuwenhuis, R. and Oliveras, A.: On SAT Modulo Theories and Optimization Problems, *Proc. 8th International Conference on Theory and Applications of Satisfiability Testing*, pp.156–169 (2006).

[18] 岩沼宏治，鍋島英知：SMT:個別理論を取り扱う SAT 技術（<特集> 最近の SAT 技術の発展），人工知能学会誌，Vol.25, No.1, pp.86–95 (2010).

[19] 梅村晃広：SAT ソルバ・SMT ソルバの技術と応用，コンピュータソフトウェア，Vol.27, No.3, pp.24–35 (2010).

[20] 平山勝敏，横尾　真：*-SAT:SAT の拡張（<特集>最近の SAT 技術の発展），人工知能学会誌，Vol.25, No.1, pp.105–113 (2010).


**高野 保真**

1981 年生．2004 年電気通信大学電気通信学科情報工学科卒業．2006 年電気通信大学大学院電気通信学研究科情報工学専攻博士前期課程修了．2008 年株式会社コマ・システムズを設立，2013 年退職．2011 年電気通信大学大学院電気通信学研究科情報工学専攻博士後期課程単位取得退学．2015 年電気通信大学大学院情報理工学研究科情報・通信工学専攻博士後期課程修了．2016 年より成蹊大学理工学部情報科学科助教．博士（工学）．プログラミング言語とその処理系に関する研究に従事．



**千代 英一郎**　（正会員）

2009 年東京大学大学院情報理工学系研究科博士課程修了．（株）日立製作所横浜研究所研究員を経て成蹊大学理工学部情報科学科准教授．博士（情報理工学）．コンパイラ，プログラム解析・検証，グラフデータベース等の研究開発に従事．