社団法人　情報処理学会　研究報告
IPSJ SIG Technical Reports

2007－ＳＬＤＭ－132（４）
2007／11／20

第一階述語論理のサブクラスに対する項の高さ縮減を用いた不変条件の近似的検証アルゴリズム

清水　博章†　　浜口　清治†　　柏原　敏伸†

† 大阪大学大学院情報科学研究科
〒 560-8531 大阪府豊中市待兼山町 1-3
E-mail: †{h-simizu,hama,kashi}@ist.osaka-u.ac.jp

あらまし　自動的にハードウェア設計の形式的検証を行う手法にモデルチェッキング技術があるが，モデルチェッキングには，扱うシステムが大規模になると状態数が爆発してしまい，計算量的に適用できないという問題がある．計算量を軽減するための１つの方法として，EUF と呼ばれる第一階述語論理のサブクラスを用いてシステムを抽象化する手法が提案されているが，EUF を用いたモデルチェッキングは一般に決定不能であることが知られている．本稿では，EUF の項の高さに上限を与えることで，現れうる項の形を制限し，状態集合の数え上げを終了させる手法を提案する．また数え上げた状態集合に対して，指定した不変条件の成立を保証できる近似アルゴリズムを提案する．さらに，アルゴリズムを簡単な回路設計に適用した例を示す．
キーワード　モデルチェッキング，第一階述語論理，抽象化，不変条件

An Approximate Invariant Property Checking Using Term-Height Reduction for a Subset of First-Order Logic

Hiroaki SHIMIZU†, Kiyoharu HAMAGUCHI†, and Toshinobu KASHIWABARA†

† Graduate School of Information Science and Technology, Osaka University
Machikaneyama-cho 1-3, Toyonaka, Osaka 560-8531, Japan
E-mail: †{h-simizu,hama,kashi}@ist.osaka-u.ac.jp

Abstract　Model checking technique, which is a method to verify systems automatically, have attracted attentions. Model checking, however, cannot be applied to systems which are large or complicated. To handle this problem, abstraction techniques using a subset of first-order logic, called EUF, have been proposed, but a model checking using EUF is generally undecidable. In this report, we introduce a method to terminate state enumeration by using term-height reduction. Finally we show an overapproximate algorithm which can check if a designated invariant always holds for the system. Furthermore, we show an example of application of our algorithm to simple hardware designs.
Key words　model checking, first-order logic, abstraction, invariant

1. はじめに

近年，ハードウェアの大規模化，複雑化が進み，従来の人手による設計記述の検証や，シミュレーションによるテストだけでは，十分に信頼性の高いハードウェアを実現することが困難になりつつある．またその一方で，ハードウェア開発にかかる期間を削減したいという要求もあり，できるだけ早期の段階で設計の誤りを発見することや，テスト工程に要する期間を短縮できるようになることが望まれている．これらの状況を解決するための１つの方法として期待されているのが，自動的にシステムの検証を行う手法であり，そのなかでも特にモデルチェッキングと呼ばれる技法が注目されている．

モデルチェッキング [1] は，与えられた設計が，指定したプロパティ（性質）を満たすかどうかを検証する技術であり，起こりうるあらゆる状態を網羅的に探索することから，テストパターンを作成する必要がなく，かつ，指定したプロパティに関してはすべての誤りを発見することが可能であるという利点がある．実際に，SMV [6], SPIN [7], CBMC [8], UCLID [9] 等の複数のモデルチェックツールがすでに実用化されており，その成功適用事例も報告されている．

項 := 変数 | 関数記号 (項, ... , 項) | ITE(論理式, 項, 項)
論理式 := *true* | *false* | 論理変数 |
(項 = 項) | 述語記号 (項, ... , 項) |
論理式 ∨ 論理式 | 論理式 ∧ 論理式 | ¬ 論理式

図 1 EUF の構文

```
x:=1;
if (y=1)
    y:=(x+y)*2;
else
    y:=(x+y)/2;
```

→

```
x:=c1;
if (y=c1)
    y:=g(f(x,y),c2);
else
    y:=h(f(x,y),c2);
```

図 2 ソースコードの抽象化例

しかし，モデルチェッキングには，扱う対象が大規模になると到達可能な状態の数が爆発してしまい，計算量的に適用できないという問題がある．状態爆発を抑える方法の 1 つとして，設計をそのままブールレベルの状態遷移関数へ変換するのではなく，設計を抽象化することで得られるモデルをモデルチェッカの入力とすることが考えられる．本稿では，限量子を含まない等号付き第一階述語論理（Quantified-free First-order Logic with Equality and Uninterpreted Functions，以下 EUF という）[2] と呼ばれる論理体系を用いて設計を抽象化することで，モデルチェッキングを行うことを考える．

一方，EUF モデルチェッキングは一般に決定不能であることが知られている [3]．実際，EUF を用いて記述した状態遷移関数に対して，到達可能な状態の数え上げを行うと，状態変数に代入されうる項の形が無限に存在するようになり，手続きが停止しない．この問題の対処法の 1 つとしては，あらかじめ指定した有限ステップ内でのみプロパティ判定を行う限定モデルチェッキング [5] が挙げられるが，この方法ではすべての到達可能な状態でプロパティが満たされることを保証できない．

本稿では，モデルチェッキングの部分問題である，不変条件を検証する問題を取り上げる．この問題に対して，項の高さに上限を定め，現れうる項の形を制限することにより，状態の数え上げを停止させる方法を提案する．さらに，与えられた不変条件が，提案手法で列挙した有限個の状態に対して常に満たされるなら，もともとの設計に対してもその不変条件が常に満たされることを示す．ただし，このアルゴリズムは到達可能な状態集合を包含する近似集合を求めるため，逆は言えず，求めた状態集合に対して不変条件が成立しない場合は，実際には，何も結論することができない．

以下，2 章で EUF と状態機械の定義を行い，3 章で EUF を用いた状態空間探索の説明，および問題定義を行う．4 章で提案アルゴリズムの詳細を述べ，5 章でアルゴリズムの適用例を示す．最後にまとめと今後の課題を述べる．

2. EUF と状態機械

2.1 EUF

2.1.1 構　　文

EUF は，一般の第一階述語論理から限量子を取り除き，特別な記号として等号を加えた論理体系である．EUF は項と論理式からなり，構文は図 1 で示される．*true*, *false* はそれぞれ真偽に対応する定数である．また，ITE 項は *if-then-else* に対応する構文である．

ITE 項を含まない任意の項 t を考えると，一般に，t は関数記号の入れ子構造をとる．項 t の高さ *term-height*(t) は次の再帰的定義により与えられる．

term-height(t) :=

- $t = f(t_1, \ldots, t_n)$ ならば，
  $MAX(\textit{term-height}(t_1), \ldots, \textit{term-height}(t_n)) + 1$
- t が変数ならば 0

ただし MAX は引数のなかから最大値を返す関数である．例えば，`c1`, `c2` を変数，`f` を 1 引数関数記号，`g` を 2 引数関数記号とすると，項 `c1`，`f(c1)`，`g(g(c1,f(c2)),f(c1))` の高さはそれぞれ 0, 1, 3 となる．

本稿では，等式，述語，論理変数のことを原子論理式と呼び，原子論理式，またはその否定をリテラルと呼ぶ．積項とは，リテラルを論理積で結合することで得られる論理式，または 1 つのリテラルからなる論理式のことをいう．積項を論理和で結合することで得られる論理式，または 1 つの積項からなる論理式を積和標準形と呼ぶ．

2.1.2 意　　味

空でないドメイン $\mathcal{D}$，解釈 σ に対して，論理式の真偽が定義される．解釈 σ は k 引数の関数記号，述語記号にそれぞれ関数 $\mathcal{D}^k \to \mathcal{D}$，$\mathcal{D}^k \to \{true, false\}$ を割り当てる．また，各変数には $\mathcal{D}$ の要素を，論理変数には *true* または *false* をそれぞれ割り当てる．項 t，論理式 α の評価 $\sigma(t)$，$\sigma(\alpha)$ をそれぞれ次のように定義する．

- 項 $t = f(t_1, t_2, \ldots, t_n)$ に対して，$\sigma(t) = \sigma(f)(\sigma(t_1), \sigma(t_2), \ldots, \sigma(t_n))$
- 項 $t = ITE(\alpha, t_1, t_2)$ に対して，$\sigma(\alpha) = true$ なら $\sigma(t) = \sigma(t_1)$，そうでないなら $\sigma(t) = \sigma(t_2)$
- 論理式 $\alpha = p(t_1, t_2, \ldots, t_n)$ に対して，$\sigma(\alpha) = \sigma(p)(\sigma(t_1), \sigma(t_2), \ldots, \sigma(t_n))$
- 論理式 $\alpha = (t_1 = t_2)$ に対して，$\sigma(t_1) = \sigma(t_2)$ ならば $\sigma(\alpha) = true$，$\sigma(t_1) \neq \sigma(t_2)$ ならば $\sigma(\alpha) = false$
- 論理式 $\alpha = \neg\alpha_1$ に対して，$\sigma(\alpha) = \neg\sigma(\alpha_1)$
- 論理式 $\alpha = \alpha_1 \circ \alpha_2$ （$\circ$ は $\vee$ または $\wedge$）に対して，$\sigma(\alpha) = \sigma(\alpha_1) \circ \sigma(\alpha_2)$

任意のドメインと解釈 σ に対して論理式 α が $\sigma(\alpha) = true$ を満たすとき，α は恒真であるという．

簡単のため，本稿では *TRUE* という特別な変数を導入して，述語 $p(t_1, \ldots, t_n)$ を関数として扱うことにする．このとき，$\neg p(t_1, \ldots, t_n)$ は $\neg(p(t_1, \ldots, t_n) = TRUE)$ で表現される．これにより，リテラルはすべて等式，論理変数，およびそれらの否定により表現される [4]．

2.1.3 EUF を用いた抽象化

設計の高位記述が与えられた場合の抽象化例を図 2 に示す．

この例では，加算，乗算，除算をそれぞれ関数記号 f，g，h で置き換え，演算の意味を考えず単に記号として扱うようにしている．また整数値 1, 2 も，それぞれ変数 c1, c2 で置き換えて，記号として扱うようにしている．

2.2 EUF 状態機械

EUF 状態機械は遷移関数の集合により定義される．遷移関数は次の 4 つのタイプの変数を用いて記述される．

- 論理状態変数 $b_1, \ldots, b_m$
- 項状態変数 $t_1, \ldots, t_n$
- 論理変数 $a_1, \ldots, a_p$
- 変数 $c_1, \ldots, c_q$

次状態における論理状態変数を $b'_1, \ldots, b'_m$，項状態変数を $t'_1, \ldots, t'_n$ とすると，遷移関数は $b'_i := F_i$ $(1 \leqq i \leqq m)$，および $t'_j := T_j$ $(1 \leqq j \leqq n)$ で与えられる．F_i は命題論理式であり，論理状態変数と論理変数を含む．T_j は項であり，上で挙げた 4 つのタイプの変数を含む．なお，論理変数と変数は入力変数に相当する．また，次の論理式を遷移関係と呼ぶ．

$$\bigwedge_{1 \leqq i \leqq m} (b'_i = F_i) \wedge \bigwedge_{1 \leqq j \leqq n} (t'_j = T_j) \qquad (1)$$

EUF 状態機械の動作は，初期状態と各ステップにおける解釈によって決まる．初期状態において，各論理状態変数には *true* または *false* が割り当てられ，各項状態変数には変数が割り当てられる．また，解釈系列 $\tilde{\sigma} = (\sigma^0, \sigma^1 \ldots)$ を考え，σ^i が i ステップ目での解釈を表すものとする．なお，解釈系列は無限長である．無矛盾性を保証するために，解釈系列は次の条件を満たす必要がある．

- 各変数，論理変数，関数記号，述語記号には，任意のステップで同じ解釈が割り当てられる．すなわち，任意の i について，各変数 c_j，論理変数 a_j，関数記号 f_j，述語記号 p_j の解釈は $\sigma^0(c_j) = \sigma^i(c_j)$，$\sigma^0(a_j) = \sigma^i(a_j)$，$\sigma^0(f_j) = \sigma^i(f_j)$，$\sigma^0(p_j) = \sigma^i(p_j)$ である．
- 遷移関数が $b'_j := F_j$，$t'_j := T_j$ であるとき，任意の i について，状態変数の値は $\sigma^{i+1}(b_j) = \sigma^i(F_j)$，$\sigma^{i+1}(t_j) = \sigma^i(T_j)$ である．
- 初期状態で各状態変数 t_j に変数 c_j が代入されるとき，$\sigma^0(t_j) = \sigma^0(c_j)$ である．

これらの条件をすべて満たす解釈系列を，標準的な解釈系列と呼ぶ．以下では，解釈系列として標準的な解釈系列のみを考える．

3. 状態空間探索と不変条件の判定

設計の初期状態から到達可能な状態をすべて求めることで，不変条件判定を行う．本章では，EUF 状態機械が与えられたときの状態空間探索の方法を述べる．ここでは，状態は EUF の項ベクトルを用いて表現されており，状態の数え上げの段階では，その意味については考慮しない．意味を考慮するのは，調べようとしている不変条件の恒真性判定を行う段階である．

3.1 状態の構成要素

各状態は次の 3 つの要素で構成されるものとする．ただし，T は ITE 項を含まないすべての項からなる集合とする．

- 状態ベクトル $\vec{v} = (\vec{b}, \vec{t})$
- 同値関係の集合 $E \subseteqq T \times T$
- 非同値関係の集合 $N \subseteqq T \times T$

ただし，$\vec{b} = (b_1, \ldots, b_m)$, $\vec{t} = (t_1, \ldots, t_n)$ とする．$b_i (1 \leqq i \leqq m)$ は *true* または *false*，$t_j (1 \leqq j \leqq n)$ は項である．E と N は，その状態に到達するために満たされなければならない同値関係，非同値関係をそれぞれ保持する集合である．ただし，各要素に含まれる変数は，$\vec{t}$ に現れる変数に限る（もし $\vec{t}$ に現れない変数が含まれる場合は，その要素を削除する）．

3.2 ITE 項の除去

遷移関係中に ITE 項を含む等式 $T = ITE(\alpha, t_1, t_2)$ が含まれている場合は，等式を次の式で置換することで，ITE 項を除去する．

$$(\alpha \wedge T = t_1) \vee (\neg\alpha \wedge T = t_2) \qquad (2)$$

ITE 項が入れ子になっている場合は，外側の ITE 項から順次置換していくことで，遷移関係中の ITE 項をすべて除去することができる．

3.3 状態空間探索

EUF 状態機械から得られる遷移関係にしたがって，初期状態から到達可能な状態を順次求めていくことで状態の列挙をおこなう．遷移関係は，あらかじめ ITE 項をすべて除去し，さらに積和標準形に変形しておくものとする．

現状態 $s = (\vec{v}, E, N)$ から次状態 $s' = (\vec{v}', E', N')$ を構成する方法は以下の通りである．まず，遷移関係の式中に現れる現状態の変数をすべて $\vec{v}$ の内容で置換することにより，論理式 $\alpha \triangleq \alpha_1 \vee \alpha_2 \vee \cdots \vee \alpha_p$ を得る．ただし，各 α_i は次状態変数 $\vec{v}'$ と変数，論理変数から構成される積項である（状態変数は含まれない）．各 $\alpha_i \triangleq \beta_1 \wedge \beta_2 \wedge \cdots \wedge \beta_q$（各 β_j はリテラル）について，α_i が充足可能であるなら，α_i に対応する次状態を生成する．その際，各 β_j に対して，β_j が次状態変数 v'_k を含む等式 $v'_k = t$ であるならば，次状態における v_k の内容を t とする．また，β_j が次状態変数を含まない等式 $e_1 = e_2$ ならば E' に同値関係 (e_1, e_2) を，次状態変数を含まない等式の否定 $\neg(n_1 = n_2)$ ならば N' に非同値関係 (n_1, n_2) を，それぞれ追加する（E', N' はそれぞれ E, N で初期化されているとする）．β_j が論理変数，またはその否定であるならば，β_j を真とするように論理変数に *true* または *false* を割り当てる．真偽を決定できない論理変数については，*true* を割り当てた場合，および *false* を割り当てた場合の 2 通りの次状態を生成する．なお，α_i が充足不能である場合は，α_i から次状態は生成されない．

以上の操作により，状態 s から状態 s' が構成されるとき，s から s' へは 1 ステップで遷移可能であるという．EUF 状態機械 M と初期状態 s^0 が与えられたとき，任意の i $(\geqq 0)$ に対して s^i から s^{i+1} へ 1 ステップで遷移可能であるならば，$(s^0, s^1, \ldots)$ は M の状態遷移系列であるという．状態遷移系列は無限長である．

3.4 プロパティ判定

プロパティ P として，本稿では不変条件のみを扱うこととす

る．P は論理式であり，論理変数，変数，論理状態変数，項状態変数からなる．ただし，各状態においてプロパティ判定を行う際には，論理状態変数，項状態変数はその状態における真偽値，および項でそれぞれ置換される（置換後，プロパティに状態変数は残らない）．

状態遷移系列 $s = (s^0, s^1, \ldots)$ が与えられたとする．状態 $s^i = (\vec{v}^i, E^i, N^i)$ において，P に含まれる状態変数を $\vec{v}^i$ の内容で置換することで得られる論理式を P_{v^i} と書くことにする．このとき，状態 s^i でプロパティ P が満たされるとは，次の論理式 P^i が恒真であることをいう．

$$P^i \triangleq \bigwedge_{(e_1,e_2)\in E^i} (e_1 = e_2) \wedge \bigwedge_{(n_1,n_2)\in N^i} (n_1 \neq n_2) \rightarrow P_{v^i} \qquad (3)$$

論理式系列 $(P^0, P^1, \ldots)$ を，状態遷移系列 s に対する P のプロパティ系列と呼ぶ．

3.5 EUF 不変条件判定問題

任意の i について α^i が恒真であるような論理式系列 $\tilde{\alpha} = (\alpha^0, \alpha^1, \ldots)$ を恒真式系列という．

入力として，検証対象である設計をモデル化した EUF 状態機械 M とプロパティ P が与えられるものとする．このとき，M から得られる任意の状態遷移系列 s に対する P のプロパティ系列が恒真式系列であるなら，M は P を常に満たすという．

以下では，M が P を常に満たすかどうか判定する問題を考える．この問題は一般には決定不能となる [3]．以下のアルゴリズムでは，項の高さの上限値 $maxh(\geq 0)$ を与え，これを利用した近似的な到達可能状態集合を求めることを考える（詳細は 4 章で述べる）．このアルゴリズムは到達可能な状態集合を包含する近似集合を求めるため，すべての状態で P が成立すると判定されれば，その結果は保証されるが，そうでない場合は，何も結論することができない．

4. 到達可能な状態集合の近似

EUF を用いて状態ベクトルを表現した場合，状態ベクトルに現れる項の形が無限に存在しうるため，状態の数え上げが停止しない可能性がある．本章では，状態の数え上げを停止させ，不変条件の成立を保証できる近似アルゴリズムを示す．

4.1 アルゴリズムの概要

本稿で提案するアルゴリズムの処理の流れを図 3 に示す．R はその時点までに探索した状態を保持する集合である．R を初期状態集合で初期化した後，R から 1 回の遷移で到達可能な未訪問の状態 s' を順次訪問していき，その都度，R に s' を追加していく．ただし，状態ベクトル中の項の高さが $maxh$ を超えた場合は，項の高さを縮減する．また，訪問済みのある状態 $s \in R$ に対して，4.3 節で述べる $s \geqq s'$ という関係が成り立つならば，s' は s に包含されると見なし，s' を R に追加せず，s' より先の状態の訪問も行わない．

以下で，項の高さを縮減する処理と状態の包含関係判定について説明する．

4.2 項の高さの縮減

状態ベクトル中に，高さが $maxh$ よりも大きい項が現れた場合には，最も内側にある関数を 1 つの新たな変数に置き換えることで抽象化し，項の高さを縮減する．例えば，高さが 3 の項 g(g(c1,f(c2)),f(c1)) の高さを 2 に制限すると，g(g(c1,c3),f(c1)) となる（f(c2) を新変数 c3 で置き換えた）．さらに高さを 1 に制限すると，g(c4,c5) となる（g(c1,c3), f(c1) をそれぞれ新変数 c4, c5 で置き換えた）．

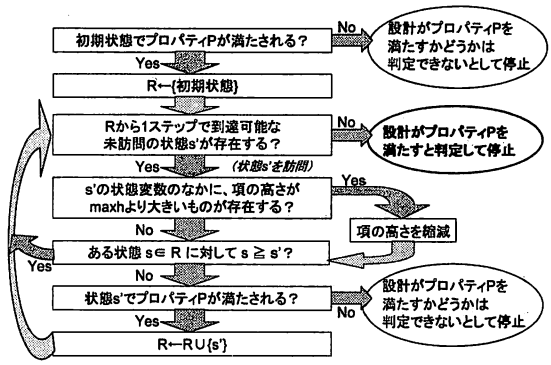

図 3 アルゴリズムの処理の流れ

[定理 1] 任意の EUF 論理式 F に対して，F の項の高さを縮減することで得られる論理式 F' が恒真であるなら F も恒真である．

（証明略） □

4.3 状態の包含関係

状態 $s = (\vec{v}, E, N)$（ただし $\vec{v} = (\vec{b}, \vec{t})$）において，$\vec{t}$ に現れる変数の集合を V_t とし，変数集合 $D = \{d_1, d_2, \ldots, d_{|V_t|}\}$ を考える（$V_t \cap D \neq \phi$）．V_t から D への一対一写像を map_t としたとき，$\vec{t}$ の成分 t_i に現れる各変数 $c \in V_t$ を $map_t(c)$ で置き換えて得られる項を便宜上 $map_t(t_i)$ と書く．

項ベクトル $\vec{t}$ の各成分 t_i を $map_t(t_i)$ で置き換えて得られる項ベクトルを $\vec{t}_{map_t}$ と書く．また，同値関係集合 E の各要素 (e_1, e_2) を $(map_t(e_1), map_t(e_2))$ で置き換えて得られる集合を E_{map_t} と書く．同様に，非同値関係集合 N についても N_{map_t} を定義する．

[定義 1] 状態 $s = (\vec{v}, E, N)$ と状態 $s' = (\vec{v}', E', N')$ が次の条件をすべて満たすとき，s は s' を包含するといい，$s \geqq s'$ と書く．ただし，$\vec{v} = (\vec{b}, \vec{t})$，$\vec{v}' = (\vec{b}', \vec{t}')$ とする．

- $\vec{b} = \vec{b}'$
- $|V_t| = |V_{t'}|$
- ある $map_t, map_{t'}$ が存在して，$\vec{t}_{map_t} = \vec{t}'_{map_{t'}}$ かつ $E_{map_t} \subseteqq E'_{map_{t'}}$ かつ $N_{map_t} \subseteqq N'_{map_{t'}}$ □

状態空間探索の際，現状態 r から新たに訪れた状態 s' が，訪問済みのある状態 s に対して $s \geqq s'$ ならば，r から s' へ遷移する代わりに r から s へ遷移し，s' から先の状態は訪問しないようにする．このとき，s' は s に吸収されたという．

項の高さを制限し，かつ，状態吸収による遷移を追加することで得られる状態遷移グラフを近似状態遷移グラフと呼ぶ．

[定理 2] 任意の EUF 状態遷移グラフに対して，その近似状態遷移グラフの状態数は有限である．

（証明）

項の高さを $maxh$ に制限すると，状態変数に現れる項の形は有限種類となる．また，各状態において，状態変数に現れる変数の総数も有限となるため，同値関係集合，非同値関係集合の内容も，有限種類しか存在しない．以上より，変数名の違いを無視すると状態構成要素の内容は有限種類しか存在しないので，近似状態遷移グラフの状態数は有限である． □

［定理 3］ 任意の EUF 状態遷移グラフ G と，その近似状態遷移グラフ G_a に対して，G_a の任意の状態でプロパティ P が満たされるなら，G の任意の状態で P は満たされる．

（証明）

refine を，項の高さ縮減により導入された新変数から，縮減を一切行わない場合のもともとの項を求める関数とする．ただし，以下の（2）における変数名書き換えにより導入された変数は，書き換え前の変数名に戻して処理をするものとする．

G に任意の標準的な解釈系列 $\tilde{\sigma}$ を与えることで得られる状態遷移系列を $\tilde{s} = (s^0, s^1, \ldots)$ とする．G_a にある解釈 σ_a を与えることで得られる状態遷移系列を $\tilde{s_a} = (s_a^0, s_a^1, \ldots)$ とする．ここで，σ_a は次のように定めるものとする．

（1） s_a^i から s_a^{i+1} へ遷移する際，項の高さ縮減が起こる場合：

導入された各新変数 c_{new} に対して，$\sigma_a(c_{new}) \triangleq \sigma(refine(c_{new}))$ と定める．

（2） s_a^i から s_a^{i+1} へ遷移する際，状態の吸収が起こる場合：状態の吸収を考えない場合の s_a^i からの遷移先状態を r_a^{i+1} とする．状態変数のマッピング判定方法より，r_a^{i+1} の状態変数に現れる変数名から s_a^{i+1} の状態変数に現れる変数名へのある一対一写像を r_a^{i+1} の状態変数内の各変数に適用することで，s_a^{i+1} と r_a^{i+1} の各状態変数の内容を完全に一致させることができる．ゆえに，変数名の違いを除けば，s_a^{i+1} と r_a^{i+1} の各状態変数はすべて同一のものであるといえる．また s_a^i から s_a^{i+1} へ遷移する際，s_a^{i+1} の状態変数に現れる変数名は，r_a^{i+1} の状態変数に現れる変数名をすべて未使用の新しい名前で書き換えたものであると見なすことができる．r_a^{i+1} の状態変数に現れる変数 $c_1, \ldots, c_m$ が，それぞれ未使用の名前をもつ変数 $c_1', \ldots c_m'$ で書き換えられたとき，$\sigma_a(c_j') \triangleq \sigma_a(c_j)$ と定める．

（3） その他の変数 c については，$\sigma_a(c) \triangleq \sigma(c)$ とする．

以上のように解釈 σ_a を定めると，任意の $\tilde{s}$ と同値な状態遷移系列 $\tilde{s_a}$ を必ず得ることができる．

次に，任意のプロパティ P が与えられたとき，$\tilde{s_a}$ に対する P のプロパティ系列が恒真式系列であるならば，$\tilde{s}$ に対する P のプロパティ系列も恒真式系列であることを示す．

上の（2）で述べたように，変数名の違いを除けば，吸収先の状態と吸収される側の状態の各状態変数の内容はすべて同一のものであるといえる．また状態吸収の際，変数名のマッピングのもとで，吸収先の状態の同値集合，非同値集合はそれぞれ，吸収される側の状態の同値集合，非同値集合の部分集合となっているため，式 (3) を用いてプロパティ判定を行うと，吸収先の状態で P が満たされるなら，吸収される側の状態でも P は満たされる．よって，任意の i について，状態 s_a^i で P が満たされるなら，状態 s^i でも P は満たされる． □

［定理 4］ 近似状態遷移グラフに対してプロパティが常に満たされるならば，もともとの（抽象化されていない）設計に対してもプロパティは常に満たされる．

（証明）

EUF による抽象化が一般にコンサバティブであることと，定理 3 より明らかである． □

4.4 状態空間探索の例

提案アルゴリズムを用いた状態空間探索を，図 4 の EUF で抽象化されたコードを例にして説明する．このコードは，普通に状態の数え上げを行った場合には停止しない例である．コードの左端の番号は行ラベルであり，各状態は論理状態変数を用いて現在の実行箇所のラベル情報を保持しているものとする．このコードに対応する遷移関数は図 5 のようになる．

```
0:  while (t1!=t2){
        t1 := f(t1,t2);
    }
1:  t2 := g(t2);
```

図 4 EUF コードの例

$$b1' := \mathrm{ITE}(b1 = 0, \mathrm{ITE}(t1 = t2, 1, 0), 1)$$
$$t1' := \mathrm{ITE}(b1 = 0, \mathrm{ITE}(t1 = t2, t1, f(t1, t2)), t1)$$
$$t2' := \mathrm{ITE}(b1 = 0, \mathrm{ITE}(t1 = t2, g(t2), t2), t2)$$

図 5 図 4 のコードに対応する遷移関数

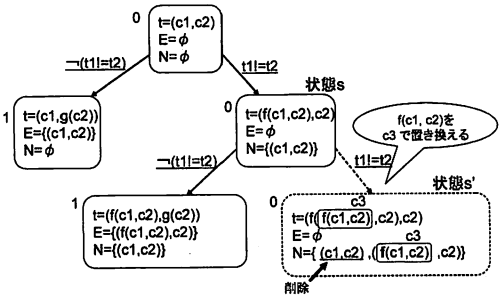

図 6 図 5 の遷移関数に対する状態空間探索（$maxh = 1$）

4.4 状態空間探索の例

提案アルゴリズムを用いた状態空間探索を，図 4 の EUF で抽象化されたコードを例にして説明する．このコードは，普通に状態の数え上げを行った場合には停止しない例である．コードの左端の番号は行ラベルであり，各状態は論理状態変数を用いて現在の実行箇所のラベル情報を保持しているものとする．このコードに対応する遷移関数は図 5 のようになる．

$maxh = 1$ としたときの状態空間探索が進む様子を図 6 に示す．ただしプロパティ判定は省略している．また，項状態変数 t1, t2 の初期値はそれぞれ c1, c2 としている．図中の状態 s から s' へ遷移する際，t1 の項の高さが $maxh$ を超えるため，項の高さを縮減する処理が行われ，新変数 c3 が導入される．同時に，s' の状態変数に c1 が含まれなくなったので，非同値関係 (c1,c2) は削除される．このとき，s と s' は同じ行ラベル 0 をもつので論理状態変数の内容が等しく，かつ，$map_t = \{\mathtt{c1} \to \mathtt{d1}, \mathtt{c2} \to \mathtt{d2}\}$，$map_{t'} = \{\mathtt{c3} \to \mathtt{d1}, \mathtt{c2} \to \mathtt{d2}\}$ のもとで，項状態変数，同値関係集合，非同値関係集合についての条件も満たされるので，$s \geqq s'$ と判定される．ゆえに s' は s に吸収される．したがって，s から s' へ遷移する辺の代わりに，s から s 自身への自己ループが追加されたものが近似状態遷移グラフとして得られ，状態数は 4 で終了する．

```
X8 := X; // X: external input
X7 := X8;
X6 := X7;
X5 := X6;
X4 := X5;
X3 := X4;
X2 := X3;
X1 := X2;

M8 := X8 * H0;
M7 := X7 * H1;
M6 := X6 * H2;
M5 := X5 * H3;
M4 := X4 * H4;
M3 := X3 * H5;
M2 := X2 * H6;
M1 := X1 * H7;

OUT := ((M7+M5)+(M3+M1))+((M8+M6)+(M4+M2));
```

図 7 FIR フィルタ（1）の遷移関数

5. 適 用 例

提案アルゴリズムを C++言語を用いて実装し，計算機（CPU: Intel Celeron 1.46GHz，物理メモリ:2GB，OS:Windows XP）上で実験を行った．SAT ソルバは Yices [10] を用いた．アルゴリズムの適用例として，2 通りの方法で実装された 8 タップの FIR フィルタ設計記述の等価性判定を考える．これらの設計は乗算回路を含んでいるため，ブール関数レベルの記述を対象とした従来のモデルチェッキングでは計算量が大きくなりすぎて対処できない．また，新たな入力が次々と与えられ続けるので，自明な方法での状態の数え上げは一般には停止しない．

FIR フィルタの遷移関数を図 7，8 に示す．図 7 は FIR フィルタを単純に実装したものである．一方，図 8 は乗算器，加算器の個数を半分に減らして，同様の FIR フィルタを実装したものである．図 7，8 の FIR フィルタをそれぞれ FIR1，FIR2 と呼ぶことにする．FIR1 の出力 OUT からは，最初の外部入力値がレジスタ X8 に入力されてから 9 サイクル後に初めて有効な値が得られ，以後は 1 サイクルごとに新たな出力が得られる．一方，FIR2 の出力 OUT からは，最初の外部入力値がレジスタ D8 に入力されてから 16 サイクル後に初めて有効な値が得られ，以後は 2 サイクルごとに新たな出力が得られる．ここでは，FIR1 の 9 サイクル目の OUT と FIR2 の 16 サイクル目の OUT を比較し，それ以降は，FIR1 の OUT を 1 サイクルごとに，FIR2 の OUT を 2 サイクルごとにそれぞれ取り出し，それらの内容を比較した．その結果，FIR1 と FIR2 の出力が常に等価であることを判定することができた．なお，この例では，0 以上の任意の $maxh$ に対して設計の等価性を保証することができ，いずれの場合も実行に要する時間は 1 秒以下であった．

6. おわりに

本稿では，一般には状態の数え上げが終了しない EUF を用いた状態機械に対して，項の高さの縮減，および状態間の包含関係を導入することで，状態の数え上げを終了させる手法について述べた．また，数え上げた有限の状態集合に対してプロパティ（不変条件）が常に満たされるなら，抽象化されていない設計に対してもそのプロパティが常に満たされることを保証できる近似アルゴリズムを示した．

```
// TURN is a mod 2 counter.
TURN := if (TURN == Zero) then One else Zero;

D8 := if (TURN == Zero) then X  else D8;
D7 := if (TURN == Zero) then D8 else D7;
D6 := if (TURN == Zero) then D7 else D6;
D5 := if (TURN == Zero) then D6 else D5;
D4 := if (TURN == Zero) then D5 else D4;
D3 := if (TURN == Zero) then D4 else D3;
D2 := if (TURN == Zero) then D3 else D2;

E8 := if (TURN == Zero) then E8 else D8;
E6 := if (TURN == Zero) then E6 else D6;
E4 := if (TURN == Zero) then E4 else D4;
E2 := if (TURN == Zero) then E2 else D2;

F0 := if (TURN == Zero) then H1 else H0;
F2 := if (TURN == Zero) then H3 else H2;
F4 := if (TURN == Zero) then H5 else H4;
F6 := if (TURN == Zero) then H7 else H6;

ACC := if (TURN == Zero) then ACC else MA;
OUT := if (TURN == Zero) then ACC + MA else OUT;
※ただし MA = E8 * F0 + E6 * F2 + E4 * F4 + E2 * F6
```

図 8 FIR フィルタ（2）の遷移関数

今後の課題として，プロパティが満たされないことも判定できるようにアルゴリズムを構築することが考えられる．そのためには，必要に応じて EUF 式の一部をブールレベル等の抽象度の低い論理体系へ変換するなどの手法を取り入れていくことが必要であると考えられる．また，不定長の配列やメモリを扱えるようにアルゴリズムを拡張することも重要な課題である．

文 献

[1] E. M. Clarke, O. Grumberg, D. A. Peled, "Model Checking," The MIT Press, 1999.

[2] J. R. Burch, and D. L. Dill, "Automated verification of pipelined microprocessor control," Computer-Aided Verification CAV'94, LNCS 818, pages 68-80, 1994.

[3] R. Hojati, A. Isles, D. Kirkpatrick, R. K. Brayton, "Verification using uninterpreted functions and finite instantiations," Formal Methods in Computer-Aided Design, LNCS 1166, pages 218-232, 1996.

[4] H. Kozawa, K. Hamaguchi, T. Kashiwabara, "Satisfiability Checking for Logic with Equality and Uninterpreted Functions under Equivalence Constraints," IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences, to appear, 2007.

[5] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu, "Symbolic Model Checking without BDDs," Proceedings of Tools and Algorithms for the Analysis and Construction of Systems (TACAS'99), LNCS 1579, pages 193-207, 1999.

[6] K. L. McMillan, "Symbolic Model Checking," Kluwer Academic Publishers, 1993.

[7] G. J. Holzmann, "The model checker SPIN," IEEE Trans. Softw. Eng., 23(5):279-295, 1997.

[8] E. M. Clarke, D. Kroening, and F. Lerda, "A Tool for Checking ANSI-C Programs," In Proc. TACAS, pages 168-176, 2004.

[9] R. E. Bryant, S. German, and M. N. Velev, "Modeling and Verifying Systems using a Logic of Counter Arithmetic with Lambda Expressions and Uninterpreted Functions," Computer-Aided Verification CAV'02, LNCS 2404, pages 78-92, 2002.

[10] Yices: An SMT Solver, "http://yices.csl.sri.com/"