社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
SS2010-36 (2010-10)

等式理論を法とするDPLL遷移系について

馬場 達也† 坂部 俊樹†† 西田 直樹†† 草刈圭一朗†† 酒井 正彦††

†,†† 名古屋大学 大学院情報科学研究科
〒464–8603 名古屋市千種区不老町
E-mail: †tbaba@sakabe.i.is.nagoya-u.ac.jp, ††{sakabe, nishida, kusakari, sakai}@is.nagoya-u.ac.jp

あらまし SMTソルバは，指定された述語理論の下で論理式の充足可能性判定を行うツールであり，配列，リスト，キューなどの多くの理論を法として，論理式の充足可能性判定を行うことができる．しかし，利用者自身が定義した理論を法とする論理式の充足可能性判定を行うのは容易ではない．なぜならば，その理論に対する決定手続きを合わせて与えることが必要であるためである．本稿では，等式理論を法とする充足可能性判定手続きを状態遷移系 DPLL(R) として定式化し，その実装方式を提案する． DPLL(R) は，与えられた等式理論から項書換え系の完備化手続きを用いて決定手続きを自動的に生成し，充足可能性判定を行う．従って，利用者は等式集合を等式理論として与えるだけでよく，決定手続きを与えなくてよい．

キーワード SMTソルバ，DPLL，プログラム検証，項書換え系

On DPLL Transition Systems Modulo Equational Theories

Tatsuya BABA†, Toshiki SAKABE††, Naoki NISHIDA††,

Keiichirou KUSAKARI††, and Masahiko SAKAI††

† Graduate School of Information Science, Nagoya University
Furo–cho, Chikusa–ku, Nagoya–City, 464–8603 Japan
E-mail: †tbaba@sakabe.i.is.nagoya-u.ac.jp, ††{sakabe, nishida, kusakari, sakai}@is.nagoya-u.ac.jp

Abstract SMT solvers are tools for deciding satisfiability of formulas under given theories such as arrays, lists, queues and so on. It is however not easy for users of SMT solvers to specify theories since the theories have to be associated with their decision procedures. In this paper, we formulate the procedure for solving the problem of satisfiability modulo an equational theory (SMET) as a state transition system DPLL(R), and propose its implementation scheme. DPLL(R) generates a decision procedure from a given equational theory and use it to solve the SMET problem. Thus, users can specify equational theories without giving decision procedures.

Key words SMT solver, DPLL, program verification, term rewriting system

1. はじめに

近年，ソフトウェアやハードウェアのシステム検証技術において形式的な証明手法への期待が高まっている．命題論理式の充足可能性を判定する手法に関する技術の向上は目覚ましく，その技術を実装したSATソルバの各方面での利用が広がっている．実際，モデル検査やプログラム検証においてSATソルバが基盤的ツールとして利用されている．しかし，一般的にシステム検証においては命題論理より表現力の高い論理が望まれる．述語論理式が定理であることを自動的に証明する手法の研究の進展により，定理自動証明系の性能は格段に向上した．しかし，そのような一般目的の定理自動証明系では，現実的なシステム検証に使うには十分な性能は得られていない．システム検証においては，関数記号や述語記号の解釈を固定して，その解釈の下で論理式が真であることを示すことができれば十分であることが多い．つまり，特定の理論における定理自動証明ができれば十分である．特定の理論あるいは解釈に固定すれば，その理論特有の証明技術を開発することで，効率よい定理証明が期待できる．このような観点から，最近，理論を法とする充足可能性判定問題の研究が進み，配列，リスト，キュー，ビットベクトル，ハッシュテーブルなどの多くの理論を法として，論理式の充足可能性を効率のよく判定する手続きが考案されている[1]．そして，その手続きに基づいて充足可能性判定をするツールであるSMTソルバが各所で開発され，システム検証

への応用も広がっている．SMT ソルバの実装手法については，文献 [6] において，命題論理の充足可能性判定手続きとして代表的な DPLL と理論の決定手続き（理論ソルバと呼ばれる）を組み合わせる手法が提案されている．その実装 DPLL(T) では，理論ソルバはパラメータ化され，DPLL の推論規則から理論ソルバを呼び出すためのインターフェースの規定を守った理論ソルバならば，組み込むことができる．

プログラム検証においては，プログラミング言語に含まれるデータ（整数，実数，配列など）を規定する理論を法とする充足可能性判定が必要であるが，それだけでは十分ではない．関数群や抽象データ型のようなプログラムモジュールで階層的に構成されたプログラムの検証ではそれらのモジュールの性質を表す理論を法とする充足可能性判定も必要である．関数群や抽象データ型の性質を表現するには等式を用いるのが便利である．

本稿では，プログラム検証への応用を目的として，等式理論を法とする充足可能性判定手続きを状態遷移系 DPLL(R) として定式化し，その実装方式を提案する．DPLL(R) では，等式理論を自由に与えることができるようにする．文献 [6] の DPLL(T) においても，等式理論を与えて充足可能性を判定できるが，その等式理論の決定手続きも合わせて与えなければならない．しかし，DPLL(R) では個々の等式理論に対する決定手続きを与えることは必要としない．つまり，DPLL(R) は項書換え系の技術を利用して与えられた等式理論に対する決定手続きを自動的に発見する．

本稿は次のように構成される．2 節では，本稿で必要な概念と記法を準備する．3 節では，等式理論を法とする抽象 DPLL 遷移系 DPLL(R) の定義とそれに関する諸定義を与え，4 節では，DPLL(R) の正当性を証明する．5 節では，DPLL(R) の実装手法について述べ，6 節ではプログラム検証への応用例を示す．7 節では，まとめと今後の課題を述べる．

2. 準　　備

本節では，本稿で必要な諸定義を与える．

項書換え系 R は書換え規則と呼ばれる項の対（それぞれ，左辺，右辺という）の有限集合である．ただし，書換え規則の右辺に現れる変数は左辺にも出現しなくてはならない．R によって定義される項集合上の書換え関係を $\rightarrow_R$ とかく．項書換え系に関する記法，定義は [3] 等にある一般的なものに従う．

原子論理式は，命題記号，あるいは，等式 $s = t$ であるとする．ここで現れる s と t はそれぞれ，変数が出現しない項である．原子論理式 p とその否定 $\neg p$ を**リテラル**という．リテラル l に対して $\neg l$ を次のように定める．$l = p$ のとき $\neg l = \neg p$，$l = \neg p$ のとき $\neg l = p$ とする．リテラルの論理和 $l_1 \vee \cdots \vee l_n$ を**節**という．節の論理積 $C_1 \wedge \cdots \wedge C_n$ を **CNF 論理式**という．しばしば，CNF 論理式を $\{C_1, \ldots, C_n\}$ のように集合で表記する．

等式理論はすべての変数が全称限量された等式の有限集合とし，しばしば，その等式の論理積とみなす．本稿では，等式理論は項書換え系とみなせるものに限る．F を CNF 論理式，R を等式理論とする．$\neg R \vee F$ が一階論理の意味で恒真であるとき F は R **恒真**であるという．$F \wedge R$ が一階論理の意味で充足可能であるとき，F は R **充足可能**，あるいは，R **無矛盾**であるという．そうでないとき，R **矛盾**，あるいは，R **充足不能**であるという．

リテラルの有限系列 M は，原子論理式 p とその否定 $\neg p$ を同時に要素として持たないとき**部分割り当て**であるという．部分割り当ては，しばしば，リテラルの有限集合，あるいは，リテラルの論理積，従って，CNF 論理式とみなされる．リテラル l が M に含まれるとき l は M において**真**であるといい，$\neg l$ が M に含まれるとき M において**偽**であるという．それ以外の場合，そのリテラルは M において**未定義**であるという．部分割り当て M に対して，$M^{=}$ を次のように定める．

$$M^{=} = \{s = t \mid s = t \in M\}$$

節 C に現れるリテラルの少なくとも一つが M に含まれる時，C は M において真であるといい，$M \models^{PL} C$ と書く．C 中のすべてのリテラルが M において偽のとき，C は M において偽である，あるいは，**衝突している**といい，$M \models^{PL} \neg C$ と書く．CNF 論理式 F 中のすべての節が M において真であるとき，M は F の**命題論理モデル**であるといい，$M \models^{PL} F$ と書く．

部分割り当て M が R 無矛盾であり，かつ，CNF 論理式 F 中の命題論理モデルであるとき，M は F の R **モデル**であるという．

CNF 論理式 F，G に対して，$F \wedge \neg G$ が R 矛盾であるとき，G は F の R **帰結**であるといい，$F \models_R G$ と表す．また，$F \models_R G$ かつ $G \models_R F$ のとき，F と G は R **同値**であるという．

R 帰結関係 $\models_R$ と R 矛盾について次の命題が成り立つ．

命題 2.1　M を任意の部分割り当て，R を任意の等式理論とする．$s \overset{*}{\leftrightarrow}_{R \cup M^{=}} t$ であるとき，かつ，そのときに限り $s = t$ は M の R 帰結，すわなち，$M \models_R s = t$ である．

命題 2.2　M を任意の部分割り当て，R を任意の等式理論とする．$u \overset{*}{\leftrightarrow}_{R \cup M^{=}} v$ である $u \neq v \in M$ が存在するとき，かつ，そのときに限り M は R 矛盾である．

3. 等式理論を法とする抽象 DPLL 遷移系

本節では，与えられた等式理論 R に対して，CNF 論理式の R 充足可能性を判定する手続きを状態遷移系 DPLL(R) として定式化する．ただし，等式理論は停止性と合流性を持つ項書換え系であるとする．

DPLL(R) における**状態**は，部分割り当て M と CNF 論理式 F の対（$M||F$ と書く），もしくは，特別な状態 $FailState$ である．$\emptyset \parallel F$ は F の充足可能性を調べるときの初期状態である．$FailState$ は初期状態の CNF 論理式が充足不能であることを意味する状態である．状態 $M||F$ の M 中のリテラル l には l^d のようにタグが付けられることがある．タグが付いたリテラル l^d を**決定リテラル**と呼ぶ．

状態から状態へ遷移を次のように定義する．

定義 3.1 DPLL(R) の遷移関係 $\Longrightarrow$ を以下の遷移規則により定義する．

$$\textit{UnitPropagate} : M \parallel F, C \vee l \Longrightarrow M\ l \parallel F, C \vee l$$
$$\textit{if} \begin{cases} M \text{ において } C \text{ は衝突している，かつ，} \\ l \text{ は } M \text{ 中で未定義} \end{cases}$$
$$\textit{Decide} : M \parallel F \Longrightarrow M\ l^d \parallel F$$
$$\textit{if} \begin{cases} l \text{ または} \neg l \text{ が } F \text{ 中に出現し，かつ，} \\ l \text{ は } M \text{ 中で未定義} \end{cases}$$
$$\textit{TheoryPropagate} : M \parallel F \Longrightarrow M\ l \parallel F$$
$$\textit{if} \begin{cases} l \text{ は } M \text{ の } R \text{ 帰結であり，} \\ l \text{ または} \neg l \text{ が } F \text{ 中に出現し，かつ，} \\ l \text{ は } M \text{ 中で未定義} \end{cases}$$
$$\textit{Fail} : M \parallel F, C \Longrightarrow \textit{FailState}$$
$$\textit{if} \begin{cases} M \text{ 中に決定リテラルが存在しない，かつ，} \\ \left(\begin{array}{l} M \text{ において } C \text{ は衝突している} \\ \text{または} \\ M \text{ は } R \text{ 矛盾である} \end{array} \right) \end{cases}$$
$$\textit{Backtrack} : M\ l^d\ N \parallel F, C \Longrightarrow M\ \neg l \parallel F, C$$
$$\textit{if} \begin{cases} N \text{ 中に決定リテラルが存在しない，かつ，} \\ \left(\begin{array}{l} M\ l^d\ N \text{ において } C \text{ は衝突している} \\ \text{または} \\ M\ l^d\ N \text{ は } R \text{ 矛盾である} \end{array} \right) \end{cases}$$

$\Longrightarrow$ の反射推移閉包を $\Longrightarrow^*$ と書く．$S_i \Longrightarrow S_{i+1}$ $(i = 0, 1, 2, \ldots)$ であるとき，$S_0 \Longrightarrow S_1 \Longrightarrow S_2 \Longrightarrow \cdots$ を**遷移系列**という．状態 S から $\Longrightarrow$ による遷移が存在しないとき，状態 S を**最終状態**という．

それぞれの遷移規則の直観的な説明は以下の通りである．

- *UnitPropagate*: CNF 論理式が真であるためには，CNF 論理式中のすべての節が真でなければならない．従って，F 中に節 $C \vee l$ があって，現在の部分割り当て M において l は未定義，かつ，節 C 中のリテラルが偽である場合，l が真でなくてはならない．この規則では，そのような l を現在状態の M に追加する．この l は単位節と呼ばれることがある．
- *Decide*: この遷移規則は，現在の状態で真とも偽とも決められていないリテラルがあれば，それを強制的に真とみなすことに対応する．F 中に出現する未定義であるリテラル l が選ばれ，M に加えられる．付け加えられるリテラル l は決定リテラルと呼ばれ，タグ d を付けられる．もし，$M\ l^d \parallel F$ から続く状態遷移の結果が成功に到らなかった場合，後述の遷移規則 *Backtrack* によって，$M \neg l \parallel F$ からの状態遷移が行われる．タグ d はこのバックトラックの目印である．
- *TheoryPropagate*: この遷移規則は DPLL(R) の最も特徴的な規則の内の 1 つである．リテラル l が現在状態の部分割り当て M の R 帰結であれば l は真となるので，l が未定義であれば，それを M に追加する．これによって，遷移規則 *Decide* の適用によって生じるバックトラックを避けることができる．
- *Fail*: この遷移規則は，現在の部分割り当て M に決定リテラルが含まれない時に，衝突している節を見つけるか，または M が R に矛盾している場合，*FailState* へと状態を遷移する．
- *Backtrack*: 衝突している節が存在し，かつ，*Fail* の規則が適用できないとき，または，M が R に矛盾し，かつ，*Fail* の規則が適用できないとき，もっとも最近加えられた決定リテラル l^d を置き換え，さらに l 以降の M 中の l より後の部分列を取り除くことによって，決定レベルを一つ戻す規則である．

ここで，今回提案した DPLL(R) と，DPLL(T) [6] の違いについて説明する．5 つの遷移規則のうち，*UnitPropagate* と *Decide* については，DPLL(T) で提案された規則と同じものを用いている．今回新たに提案した遷移規則については，まず，*TheoryPropagate* がある．*Fail* と *Backtrack* については，それらの遷移条件を変更することで新たに提案している．

例 3.2 DPLL(R) の状態遷移の例を示す．まず，項書換え系 R と CNF 論理式 F を次のようにする．

$$R : \{f(a) \to b\}$$
$$F : g(b) = b \vee a = b,$$
$$g(b) \neq b \vee a = b \vee g(f(a)) \neq b,$$
$$a \neq b \vee g(f(a)) \neq b,$$
$$a = b \vee g(f(a)) = b$$

初期状態 $\emptyset \parallel F$ からの遷移系列のひとつは次の通りである．それぞれの遷移は適用された遷移規則によって注釈されている．

$$\begin{array}{lll} & \emptyset & \parallel F \\ \Rightarrow \text{(D)} & g(b) = b^d & \parallel F \\ \Rightarrow \text{(TP)} & g(f(a)) = b^d\ g(f(a)) = b & \parallel F \\ \Rightarrow \text{(UP)} & g(f(a)) = b^d\ g(f(a)) = b\ a = b & \parallel F \\ \Rightarrow \text{(BT)} & g(b) \neq b & \parallel F \\ \Rightarrow \text{(UP)} & g(b) \neq b\ a = b & \parallel F \\ \Rightarrow \text{(UP)} & g(b) \neq b\ a = b\ g(f(a)) \neq b & \parallel F \end{array}$$

ここで，各行の状態遷移について説明する．まず，初期状態において，F 中に単位節が存在せず，また，現在状態 $M_1 \parallel F$ の部分割り当て M_1 の R 帰結であるようなリテラルが存在しないため，*UnitPropagate* と *TheoryPropagate* は適用できない．F 中のすべてのリテラルが M_1 中で未定義であるので，F 中の任意のリテラルを *Decide* の遷移規則によって真とみなすことができる．この例では $g(b) = b^d$ を真とみなし，状態遷移を起こす．次に，2 行目の状態を $M_2 \| F$ とおき，次のような項書換え系 R_2 を考える．

$$R_2 = R \cup M^{=} = \{f(a) \to b,\ g(b) \to b\}$$

この時，F 中の等式リテラル $g(f(a)) = b$ に注目すると，以下

のように，両辺の書き換え結果が一致するので，$g(f(a)) = b$ は M_2 の R 帰結といえ，また *TheoryPropagate* のすべての遷移条件を満たすため，この規則が適用される．

$$g(f(a)) \rightarrow_{R'_2} g(b) \rightarrow_{R'_2} b \quad \leftarrow^{0}_{R'_2} b$$

3 行目の状態 $M_3||F$ では，リテラル $a = b$ が M_3 において未定義かつ，単位節であるため，つまり，節 $g(b) \neq b \vee a = b \vee g(f(a)) \neq b$ において $a = b$ 以外のリテラルが M_3 において偽であるため，*UnitPropagate* の遷移規則が適用される．4 行目の状態 $M_4||F$ では，節 $a \neq b \vee g(f(a)) \neq b$ が衝突している．M_4 中に決定リテラルが存在するので，遷移規則 *Backtrack* が適用される．5 行目の状態では，節 $a = b \vee g(f(a)) = b$ において，$a = b$ が単位節であるため，*UnitPropagate* が適用され，6 行目では，同様にして，単位節 $g(f(a)) \neq b$ が真となる．7 行目の状態 $M_7||F$ は最終状態であり，かつ，*FailState* という状態ではないので，M_7 は F の R モデルである．

4. DPLL(R) の正当性

本節では，DPLL(R) が CNF 論理式の R 充足可能性問題に対して健全性を持つことと停止性を持つこと証明する．

補題 4.1 $\emptyset||F \Longrightarrow^{*}_{R} M||G$ のような遷移系列が存在するとき，以下が成り立つ．

(1) M と G 中に現れるすべての原子論理式は F 中に存在する．

(2) M はリテラル l とその否定 $\neg l$ を同時に含まない．

(3) G と F は R 同値である．

(4) もし M が $M_0 l_1 M_1 \ldots l_n M_n$ の形をしているとし，かつ，$l_1, \ldots, l_n$ はすべて M 中の決定リテラルであるならば，$F, l_1, \ldots, l_i \models_R M_i$ となる．$(0 \leq i \leq n)$

略証 性質 (4) について，*TheoryPropagate* の遷移の場合のとき，このことが成り立つことを示す．$M'||F' \Longrightarrow M''||F''$ のような遷移を考え，$M'||F'$ は，性質 (4) を満たすと仮定する．また，M' は $M'_0 l_1 M'_1 \ldots l_n M'_n$ の形をしているとする．ここで出現する $l_1 \ldots l_n$ はすべて決定リテラルである．この時，M'' は，*TheoryPropagate* の遷移規則より，$M'l$ の形をしている．ここで出現する l は，*TheoryPropagate* の遷移条件を満たすリテラルである．性質 (4) を示すためには，$F, l_1, \ldots, l_n \models_R l$ が成り立つことを示せば十分であり，このことは，以下によって示すことができる．

- $F, l_1, \ldots, l_n \models_R M'$
- $M' \models_R l$
- l は F 中に出現するリテラル
- F と G は R 同値

□

補題 4.2 もし，$\emptyset||F \Longrightarrow^{*}_{R} S$ かつ S が最終状態の時，S は *FailState* か，もしくは $M||F'$ の形をしていて，かつ，M は F の R モデルである． □

定理 4.3 （健全性） $\emptyset||F \Longrightarrow^{*}_{R} S$ のような遷移系列において，S が DPLL(R) に関して最終状態であり，かつ，もし S が $M||F'$ の形をしているときに M が R 無矛盾であるならば，以下のすべてが成り立つ．

(1) S が *FailState* ならば，かつその時に限り，F は R 充足不能である．

(2) もし S が $M||F'$ の形をしているならば M は F の R モデルである．

証明 (1, $\Rightarrow$) もし S が *FailState* ならば，$\emptyset||F \Longrightarrow^{*}_{R} M||F' \Longrightarrow_{R} FailState$ のような $M||F'$ が存在する．(i) *Fail* の遷移規則より，M 中に決定リテラルは存在せず，かつ $M \models \neg C$ のような節 C が F' 中に存在する場合，補題 4.1 の (3) より F と F' は R 同値なので，$F \models C$ がいえる．しかし，$M \models \neg C$ であることと，補題 4.1 の (4) から $F \models \neg C$ がいえるので，F は充足不能であるといえる．

(1, $\Leftarrow$) もし S が *FailState* でなければ，$M||F'$ の形をしている．しかし，補題 4.2 より M は F の R モデルだといえる．

(2) 補題 4.2 より明らか．

□

定理 4.4 （停止性） $\emptyset||F \Longrightarrow_{R} S_1 \Longrightarrow_{R} \cdots$ のような無限の遷移系列は存在しない．

停止性については，DPLL(T) の停止性の証明と同様に証明することができる．その理由としては，DPLL(R) では *TheoryPropagate*, *Fail*, そして *Backtrack* という遷移規則を新たに提案したが，停止性の証明において，それらの遷移規則は DPLL(T) における，*UnitPropagate*, *Fail*, *Backtrack* に置き換えて証明することができるからである．

5. DPLL(R) のアーキテクチャ

本節では，これまでに紹介した等式理論を法とする抽象 DPLL 遷移系を実装した，DPLL(R) のアーキテクチャについて説明する．

以下に示される図は，DPLL(R) の概要を表したものである．

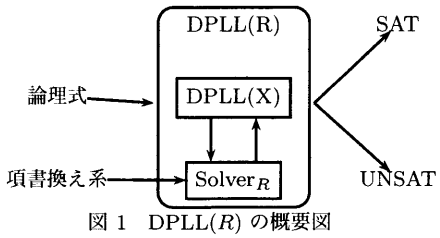

図 1 DPLL(R) の概要図

DPLL(R) は，DPLL(X) と呼ばれる，理論とは独立して DPLL アルゴリズムを処理するモジュールと，項書換え系 R によって表される等式理論の決定アルゴリズムを処理する $Solver_R$ と呼ばれるモジュールによって構成されている．両者のモジュール間にはインタフェースが存在し，扱われる理論は X によって

パラメータ化されている.

これの元となるアイデアである DPLL(T) は DPLL(X) モジュールと，理論の決定アルゴリズムを処理する $Solver_T$ と呼ばれるモジュールによって構成されている．利用者は，扱う理論に応じて $Solver_T$ を選択することにより各理論 T における充足可能性判定を行うことができる．しかし，利用者自身が $Solver_T$ を与えたい場合，利用者は，その理論の決定手続きを発見する必要があり，大変困難だと考えられる．

一方，DPLL(R) では，利用者が扱う等式理論として項書換え系 R を与えることで，自動的に以下のインタフェースを満たす $Solver_R$ を生成する．

(i) 状態遷移が行われるたびに現状態の部分割り当て M が指定された R に対して，R 矛盾してるかどうかを確認する．

(ii) R と $M^{=}$ から未定義の R 帰結となるリテラルが存在する時，そのリテラルを真とする．

R から，その等式理論に対する決定手続きを行う $Solver_R$ が自動的に生成されることによって利用者は自由に扱う等式理論を指定できる．

ここで，上記 (i)，(ii) における R 帰結と R 矛盾の判定手続きの実装について説明する．指定された項書換え系を R，現状態の部分割り当てを M としたとき，R 帰結と R 矛盾の判定には，命題 2.1, 2.2 により，項書換え系 $R \cup M^{=}$ の書換え関係 $\overset{*}{\leftrightarrow}_{R \cup M^{=}}$ の決定手続きがあればよい．本稿では，この決定手続きとして Knuth-Bendix 完備化手続き [5] により $R \cup M^{=}$ を完備化して得られる停止性と合流性のある項書換え系を利用し，(i)，(ii) のインタフェースを実現している．ここで注意したいことは，一般的に常に R 帰結と R 矛盾の判定が行えるわけではないということである．なぜならば，Knuth-Bendix 完備化手続きがかならずしも停止しないからである．この問題についての対応は今後の課題である．

6. プログラム検証への応用例

本節では，DPLL(R) を使ってフィボナッチ数の計算プログラムを検証する．フィボナッチ数の関数 fib は，以下の項書換え系 R によって定められているとする．

$$R = \begin{cases} fib(0) \to 0, \\ fib(1) \to 1, \\ fib(n+1+1) \to fib(n+1) + fib(n) \end{cases}$$

フィボナッチ数を計算するプログラムのプルーフアウトラインを以下に示す．

```
1   {n ≧ 0}
2   if n = 0
3   then
4     {n ≧ 0 ∧ n = 0}
5     {0 = fib(n)}
6     f:=0
7     {f = fib(n)}
8   else
9     {n ≧ 0 ∧ n ≠ 0}
10    {1 = fib(1) ∧ 0 = fib(0)}
11    k:=0; g:=0; f:=1;
12    {inv : f = fib(k+1) ∧ g = fib(k)}
13    while k+1 ≠ n do
14      {f = fib(k+1) ∧ g = fib(k) ∧ k+1 ≠ n}
15      {f+g = fib(k+1+1) ∧ f = fib(k+1)}
16      t:=g; g:=f; f:=f+t; k:=k+1
17      {f = fib(k+1) ∧ g = fib(k)}
18    od
19    {f = fib(k+1) ∧ g = fib(k) ∧ k+1 = n}
20    {f = fib(n)}
21  fi
22  {f = fib(n)}
```

上記のプログラムの正しさを示すには以下の検証条件がすべて R 恒真であることが証明できればよい．

(i) $4 \supset 5$

(ii) $9 \supset 10$

(iii) $14 \supset 15$

(iv) $19 \supset 20$

なお，ここで表される数字は上記のプログラム仕様の行番号の論理式を表す．

一般には，検証条件が R 恒真であることを示すためは，証明したい論理式の否定をスコーレム化し，さらに，CNF に変換する．こうして得られる論理式の R 充足不能性を DPLL(R) によって示せばよい．得られる CNF には変数が含まれるが，それらは全称限量されたものであるので，定数として扱ってよい．上記の例では，検証条件には限量子が出現していないので，スコーレム化する前の段階から変数はすべて定数として扱っている．

つまり，この例題の場合，以下の論理式を入力として，最終状態が $FailState$ となることを示すことができれば論理式の R 恒真性を示せたことになる．

(i) $F_1 : n \geqq 0 \wedge n = 0 \wedge 0 \neq fib(n)$

(ii) $F_2 : n \geqq 0 \wedge n \neq 0 \wedge (1 \neq fib(1) \vee 0 \neq fib(0))$

(iii) $F_3 : f = fib(k+1) \wedge g = fib(k) \wedge k+1 \neq n \vee (f+g \neq fib(k+1+1) \vee f \neq fib(k+1))$

(iv) $F_4 : f = fib(k+1) \wedge g = fib(k) \wedge k+1 = n \wedge f \neq fib(n)$

ここで注意したいことは，$F_1 \sim F_4$ で現れる $+$ は解釈を持たない関数記号であり，n は定数であることである．さらに，$n \leqq 0$ と $n = 0$ はそれぞれ，1 つの命題記号として扱われる．それぞれの論理式と R を入力とした DPLL(R) の実行結果は以下のとおりである．

- $F_1 : n \geqq 0 \wedge n = 0 \wedge 0 \neq fib(n)$

$$
\begin{array}{lll}
 & \emptyset & \| \ F_1 \\
\Rightarrow (UP) & n \geqq 0 & \| \ F_1 \\
\Rightarrow (UP) & n \geqq 0 \ n = 0 & \| \ F_1 \\
\Rightarrow (UP) & n \geqq 0 \ n = 0 \ 0 \neq fib(n) & \| \ F_1 \\
\Rightarrow (Fail) & FailState &
\end{array}
$$

- $F_2 : n \geqq 0 \wedge n \neq 0 \wedge (1 \neq fib(1) \vee 0 \neq fib(0))$

$$
\begin{array}{lll}
 & \emptyset & \| \ F_2 \\
\Rightarrow (UP) & n \geqq 0 & \| \ F_2 \\
\Rightarrow (UP) & n \geqq 0 \ n \neq 0 & \| \ F_2 \\
\Rightarrow (TP) & n \geqq 0 \ n = 0 \ 1 = fib(1) & \| \ F_2 \\
\Rightarrow (UP) & n \geqq 0 \ n = 0 \ 1 = fib(1) \ 0 \neq fib(0) & \| \ F_2 \\
\Rightarrow (Fail) & FailState &
\end{array}
$$

- $F_3 : f = fib(k+1) \wedge g = fib(k) \wedge k+1 \neq n \vee (f+g \neq fib(k+1+1) \vee f \neq fib(k+1))$

$$
\begin{array}{lll}
 & \emptyset & \| \ F_3 \\
\Rightarrow (UP) & f = fib(k+1) & \| \ F_3 \\
\Rightarrow (UP) & f = fib(k+1) \ g = fib(k) & \| \ F_3 \\
\Rightarrow (UP) & f = fib(k+1) \ g = fib(k) \ k+1 \neq n & \| \ F_3 \\
\Rightarrow (TP) & f = fib(k+1) \ g = fib(k) \ k+1 \neq n & \\
 & f+g = fib(k+2) & \| \ F_3 \\
\Rightarrow (UP) & f = fib(k+1) \ g = fib(k) \ k+1 \neq n & \\
 & f+g = fib(k+1+1) \ f \neq fib(k+1) & \| \ F_3 \\
\Rightarrow (Fail) & FailState &
\end{array}
$$

- $F_4 : f = fib(k+1) \wedge g = fib(k) \wedge k+1 = n \wedge f \neq fib(n)$

$$
\begin{array}{lll}
 & \emptyset & \| \ F_4 \\
\Rightarrow (UP) & f = fib(k+1) & \| \ F_4 \\
\Rightarrow (UP) & f = fib(k+1) \ g = fib(k) & \| \ F_4 \\
\Rightarrow (UP) & f = fib(k+1) \ g = fib(k) \ k+1 = n & \| \ F_4 \\
\Rightarrow (UP) & f = fib(k+1) \ g = fib(k) \ k+1 = n & \\
 & f \neq fib(n) & \| \ F_4 \\
\Rightarrow (Fail) & FailState &
\end{array}
$$

F_1 から F_4 のすべてが R 充足不能となったので，フィボナッチ数の計算プログラムのプルーフアウトラインの検証条件はすべて成立すると言える．

7. おわりに

従来の SMT ソルバにおいて，利用者は与えられた理論の下でのみ一階論理式の充足可能性判定を行うことができなかったが，今回紹介した等式理論を法とする DPLL 遷移系では，利用者が項書換え系を与えることで，自由に理論を指定することができるようになった．

今後の課題としては，DPLL(R) のプログラム検証への応用範囲を広げることを考える．具体的には，まず，配列のソートプログラムのような汎用的な例題を挙げ，それに必要となる理論を扱えるように状態遷移系を拡張する．とりわけ，整数理論はプログラム検証を行う上で特に必要となる理論と考えられるため，これを扱えるようなものにすることを考える．実装上の課題としては，DPLL(R) の効率化について考える．現在の実装では，状態遷移が毎回起こるたびに，部分割り当て M が項書換え系 R に対して R 矛盾かどうかの判定を行っている．これは，非常に非効率であると考えられるため，効率を良くする必要がある．また，R 矛盾と R 帰結の判定において，Knuth-Bendix 完備化が停止しない場合の扱いについて考える必要がある．

謝辞 本研究は，一部，文部科学省科学研究費#20300010, #20500008, #21700011 の助成を受けたものである．

文　献

[1] Armin Biere, Marijn Heule, Hans van Maaren and Toby Walsch: *Chapter 12 Satisfiability Modulo Theories*, Handbook of Satisfiability IOS Press, 2008.

[2] Bruno Dutertre and Leonardo de Moura: *A fast linear-arithmetic solver for DPLL(T)*, In CAV '06, LNCS 4144, pages 81-94, 2006.

[3] Franz Baader and Tobias Nipkow: *Term Rewriting and All That*, Cambridge University Press (1998).

[4] Harald Ganzinger, George Hagen, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tineli: *DPLL(T): Fast Decision Procedures*, CAV 2004, pp. 175-188.

[5] Knuth, D. E. and Bendix, P. B.: *Simple word problems in universal algebras*, Computational Problems in Abstract Algebra (Ed. J. Leech) pages 263-297, 1970.

[6] Robert Nieuwenhuis, Albert Oliveras and Cesare Tinelli: *Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T)*, Journal of the ACM, Vol. 53, No. 6, 2006, pp. 937-977.

[7] Saurabh Srivastava and Sumit Gulwani: *Program Verification using Templates over Predicate Abstraction*, PLDI'09

[8] Microsoft Research: Z3, *http://research.microsoft.com/en-us/um/redmond/projects/Z3/*