# 時相論理による述語抽象化のための充足可能性判定手続き

(A Decision Procedure for Temporal Formulas in Automated Predicate Abstracrion)

湯浅 能史 †
Yoshifumi YUASA

田辺 良則 †
Yoshinori TANABE

関澤 俊弦 †
Toshifusa SEKIZAWA

高橋 孝一 ‡
Koichi TAKAHASHI

† 科学技術振興機構, CREST/産業技術総合研究所システム検証センター
CREST, Japan Science and Technology Agency/ Research Center for Verification and Semantics, National Institute of Advanced Industrial Science and Technology

‡ 産業技術総合研究所システム検証センター
Research Center for Verification and Semantics, National Institute of Advanced Industrial Science and Technology

グラフ書換系の性質を検証するために，時相論理を用いて抽象化を行う手法が提案されている．この手法を述語抽象化の枠組みに適用することで，モデル検査によってプログラムのヒープ構造に関する性質を検証することが可能になる．これを実現するには，ポインタを扱う実行文に対して前条件を求めることと，時相論理式に関する充足可能性判定を行うことが必要となる．本発表では，後者の目的に適した充足可能性判定手続きを提案し，そのプロトタイプ実装について報告する．

## 1 はじめに

時相論理や様相論理における充足可能性判定手続きにはさまざまな応用があり，並行プログラムの合成や XML 文書を処理するプログラムの解析などに適用されている．

著者らは，グラフ書換え系の検証のために，時相論理を用いた抽象化手法を提案しており [1]，ここでも充足可能性判定手続きが重要な役割を果たしている．書換え対象のグラフをクリプキ構造とみなすので，論理式を充足するクリプキ構造 (Kripke structure) の有無を判定する手続きが有用となる．グラフ書き換え系の例と考えることのできるシステムには，様々なものがあるが，その一例である 1 次元セルオートマトンに対して，我々はこの抽象化手法を適用して解析を行い [2]，またそこで必要となる 2 方向 CTL の充足可能性判定手続きについて，BDD を用いた効率的な実装を行った [3].

グラフ書換え系の例として重要なものに，プログラムのヒープがある．これに関する検証にも，上述の抽象化手法は原理的に適用可能なはずである．ところが，単純に一般論を適用するだけでは，あまり精度の良い結果が得られない．これは，プログラムヒープとみなせるクリプキ構造は，ある条件を満たしているのに対し，既存の手続きが論理式を「充足可能である」と判定したときに実際に充足するクリプキ構造は，必ずしもその条件を満足しないことに起因している．

本研究では，このギャップを埋めるべく，プログラムヒープから作成されるクリプキ構造が満たすべき条件を検討し，時相論理式に対して，その条件を満たすクリプキ構造が存在するかどうかを判定する手続きを構成した．さらに，[3] と同様に BDD を用いてこの判定手続きを実装し，実際にプログラムヒープの検証に対して適用を行い，従来の手続きを用いた場合には検証できなかった項目が検証可能となることを確認した．

次節以降の構成を述べる。第 2 節では、2 方向 CTL の構文および意味論を間単に紹介する。また第 3 節以降で用いる概念・用語等の導入も行う。第 3 節で

*本研究は、科学技術振興機構戦略的創造研究推進事業 (CREST) 研究領域「情報社会を支える新しい高性能情報処理技術」研究課題「検証における記述量爆発問題の構造変換による解決」として実施された。

は、ヒープの数学的な定式化であるポインタ構造の定義とこれをクリプキ構造として扱う方法、およびその基本的な性質について述べる。第 4 節が本稿の主要部である。ポインタ構造を少し一般化した構造を定義して、これに限定した 2CTL 式の充足可能性を判定する方法について説明する。第 5 節では、第 4 節で示した充足可能性判定手続きを、BDD を利用して実装する方法を述べる。また、実際に実装した判定器での、プログラムヒープの検証例を幾つか紹介する。第 6 節は結びである。

本稿で用いる数学記法は概ね標準的である。集合 $S$ の冪集合は $\wp(X)$ と書く。集合 $S$ 上の関係 $R$ は $S^2$ の部分集合と考え、逆関係は $R^{-1}$ と記す。また写像の順像を一般化して：

$$R[X] = \{s \in S \mid \exists x \in X \;\; xRs\}$$

という記法を用いることにする。有限または無限の列 $\sigma$ に対して、その長さを $|\sigma|$ と書く。全ての $k < |\sigma|-1$ で $\sigma_k R \sigma_{k+1}$ となる時、列 $\sigma$ は $R$-鎖であるという。

## 2 2 方向 CTL について

本稿で扱う時相論理体系は、2 方向（多様相）CTL である。以下に文法および意味論を簡単に説明する。二三の細かな点を除けば、基本的に [3] と同じものなので、詳細についてはそちらを参照されたい。

様相ラベルの集合 $M$ を固定する。各様相 $m \in M$ について、逆方向を表すラベル $\bar{m}$ を用意し、それらの集合を $\bar{M}$ とする。順方向ラベル $M$ と逆方向ラベル $\bar{M}$ を併せたものを $Mod$ と書く。2 方向 CTL の論理式（以下、略して「2CTL 式」と呼ぶ）を次で定義する。

$$\begin{aligned}\varphi ::= {} & \top \mid \bot \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \\ & \mathrm{E}_A\mathrm{X}\varphi \mid \mathrm{A}_A\mathrm{X}\varphi \mid \\ & \mathrm{E}_A(\varphi\mathrm{U}\varphi) \mid \mathrm{A}_A(\varphi\mathrm{R}\varphi) \mid \mathrm{A}_A(\varphi\mathrm{U}\varphi) \mid \mathrm{E}_A(\varphi\mathrm{R}\varphi)\end{aligned}$$

但し、$p$ は原子論理式, $A$ は $Mod$ の部分集合である。集合 $A$ が一元集合 $\{a\}$ の時には、式 $\mathrm{E}_A\mathrm{X}\varphi$ を単に $\mathrm{E}_a\mathrm{X}\varphi$ と略す。式 $\mathrm{A}_A\mathrm{X}\varphi$ 等についても同様である。含意 $\psi_0 \Rightarrow \psi_1$ を用いることもあるが、これは通常のように $\neg\psi_0 \vee \psi_1$ の略記とする。また、良く知られた次の略記法も用いる。

$$\begin{aligned}\mathrm{E}_A\mathrm{F}\psi &= \mathrm{E}_A(\top\mathrm{U}\psi) \\ \mathrm{A}_A\mathrm{G}\psi &= \mathrm{A}_A(\bot\mathrm{R}\psi) \\ \mathrm{A}_A\mathrm{F}\psi &= \mathrm{A}_A(\top\mathrm{U}\psi) \\ \mathrm{E}_A\mathrm{G}\psi &= \mathrm{E}_A(\bot\mathrm{R}\psi)\end{aligned}$$

次に 2CTL 式の意味論について概説する。以下を満す三つ組 $\mathcal{M} = \langle S, \{R_m \mid m \in M\}, \lambda\rangle$ を**クリプキ構造**という。

$$S : \text{集合}, \;\; R_m \subseteq S^2, \;\; \lambda : S \to \wp(AP)$$

ここで $AP$ は原子命題の集合である。集合 $S$ の要素を**状態**と呼ぶ。二項関係 $R_{\bar{m}}$ を $R_m^{-1}$ で定め、関係 $R_a (a \in Mod)$ を様相 $a$ に関する**遷移**と呼ぶ。また $A \subseteq Mod$ の時に：

$$R_A = \bigcup_{a \in A} R_a$$

と定める。極大な $R_A$-鎖を $A$-**パス**と呼び、状態 $s$ で始まる $A$-パスの集合を $pth_A(s)$ と記する。

状態 $s \in S$ において 2CTL 式 $\varphi$ が**成り立つ**ことを、記号で $s \models \varphi$ と書く。これは論理式の構成に関する帰納法で以下のように定義される。

- $s \models \top, \; s \not\models \bot, \; s \models p \Leftrightarrow p \in \lambda(s)$,
- $s \models \neg\psi \Leftrightarrow s \not\models \psi$,
- $s \models \psi_0 \vee \psi_1 \Leftrightarrow s \models \psi_0 \; or \; s \models \psi_1$,
  $s \models \psi_0 \wedge \psi_1 \Leftrightarrow s \models \psi_0 \;\&\; s \models \psi_1$,
- $s \models \mathrm{E}_A\mathrm{X}\psi \Leftrightarrow \exists s' \;\; (sR_As' \;\&\; s' \models \psi)$,
  $s \models \mathrm{A}_A\mathrm{X}\psi \Leftrightarrow \forall s' \;\; (sR_As' \Rightarrow s' \models \psi)$,
- $s \models \mathrm{E}_A(\psi_1\mathrm{U}\psi_0) \Leftrightarrow$
  $\exists\sigma \in pth_A(s) \, \exists k < |\sigma|$
  $(\sigma_k \models \psi_0 \;\&\; \forall i < k \;\; \sigma_i \models \psi_1)$,
  $s \models \mathrm{A}_A(\psi_1\mathrm{R}\psi_0) \Leftrightarrow$
  $\forall\sigma \in pth_A(s) \, \forall k < |\sigma|$
  $(\sigma_k \not\models \psi_0 \Rightarrow \exists i < k \;\; \sigma_i \models \psi_1)$,
- $s \models \mathrm{A}_A(\psi_1\mathrm{U}\psi_0) \Leftrightarrow$
  $\forall\sigma \in pth_A(s) \, \forall k < |\sigma|$
  $(\sigma_k \models \psi_0 \;\&\; \forall i < k \;\; \sigma_i \models \psi_1)$,
  $s \models \mathrm{E}_A(\psi_1\mathrm{R}\psi_0) \Leftrightarrow$
  $\exists\sigma \in pth_A(s) \, \exists k < |\sigma|$
  $(\sigma_k \not\models \psi_0 \Rightarrow \exists i < k \;\; \sigma_i \models \psi_1)$.

また、クリプキ構造を明示する必要があるときには、これを $\mathcal{M}, s \models \varphi$ のように書くことにする。

クリプキ構造 $\mathcal{M}$ が式 $\varphi$ を**充足する**とは、ある状態 $s$ が存在して $\mathcal{M}, s \models \varphi$ となることである。その

ようなクリプキ構造が存在するとき、式 $\varphi$ は充足可能であるという。

以上が標準的な 2 方向 CTL の構文および意味論である。しかし、本稿のようにヒープの状態記述を目的とする場合、通常の充足可能性では記述に困難が生じる時がある。これに対処するため、少し一般化した充足可能性の概念を導入する。

**定義 2.1 ($\vec{\alpha}$-充足可能性)** クリプキ構造 $\mathcal{M}$ で、その全ての状態において式 $\alpha_0, \cdots, \alpha_n$ $(= \vec{\alpha})$ が成り立つものを $\vec{\alpha}$-**構造**と呼ぶ。式 $\chi$ を充足する $\vec{\alpha}$-構造が存在するとき、この式は $\vec{\alpha}$-**充足可能**であるという。

最後に次節以降で用いる概念/記法等を幾つか説明する。最外論理記号が時相演算子である 2CTL 式 $\varphi$ について、その**展開** $\exp\varphi$ を次で定める。

- $\exp \mathrm{E}_A \mathrm{X}\psi = \bigvee_{a\in A} \mathrm{E}_a \mathrm{X}\psi,$
  $\exp \mathrm{A}_A \mathrm{X}\psi = \bigwedge_{a\in A} \mathrm{A}_a \mathrm{X}\psi,$
- $\exp \mathrm{E}_A(\psi_1 \mathrm{U} \psi_0) = \psi_0 \vee (\psi_1 \wedge \mathrm{E}_A \mathrm{X} \mathrm{E}_A(\psi_1 \mathrm{U}\psi_0)),$
  $\exp \mathrm{A}_A(\psi_1 \mathrm{R} \psi_0) = \psi_0 \wedge (\psi_1 \vee \mathrm{A}_A \mathrm{X} \mathrm{A}_A(\psi_1 \mathrm{R}\psi_0)),$
- $\exp \mathrm{A}_A(\psi_1 \mathrm{U} \psi_0) =$
  $\psi_0 \vee (\psi_1 \wedge \mathrm{E}_A \mathrm{X}\top \wedge \mathrm{A}_A \mathrm{X} \mathrm{A}_A(\psi_1 \mathrm{U}\psi_0)),$
  $\exp \mathrm{E}_A(\psi_1 \mathrm{R} \psi_0) =$
  $\psi_0 \wedge (\psi_1 \vee \mathrm{A}_A \mathrm{X}\bot \vee \mathrm{E}_A \mathrm{X} \mathrm{E}_A(\psi_1 \mathrm{R}\psi_0)).$

展開式 $\exp\varphi$ は元の式 $\varphi$ と**同値**、即ち、任意のクリプキ構造の任意の状態において正否が一致している。

論理式 $\varphi_0, \cdots \varphi_n$ を含み、かつ部分式および展開で閉じた最小の集合を、これらの**閉包**と呼ぶ。記号では $cl(\varphi_0, \cdots \varphi_n)$ と書く。

2CTL 式の内、否定記号 "¬" が原子命題の前のみに現れるものを**正形式**論理式と呼ぶ。例えば $\mathrm{A}_A(p \mathrm{U} \mathrm{E}_B \mathrm{X} \neg q)$ は正形式だが、$\mathrm{A}_A(p \mathrm{U} \neg \mathrm{A}_B \mathrm{X} q)$ や $\neg \mathrm{E}_A(\neg p \mathrm{R} \mathrm{A}_B \mathrm{X} q)$ は正形式ではない。どの 2CTL 式にも、それと同値な正形式論理式が存在する。また正形式論理式の閉包に含まれる式は、すべて正形式である。

## 3 ポインタ構造

ポインタ構造とは、プログラム実行時のある瞬間のヒープを数学的に表現したものである。「フィールド名」「(ポインタ型の) プログラム変数」および「値」を表す有限集合を、それぞれ *Fld*, *Val* および $Var$ とする。但し *Val* と *Var* は共通元を持たないとする。以下を満す 4 つ組み $(N, p, d, \rho)$ を**ポインタ構造**と呼ぶ。

$$N : \text{集合}, \qquad p : \mathit{Fld} \times N \xrightarrow{p} N,$$
$$d : N \to \mathit{Val}, \qquad \rho : \mathit{Var} \xrightarrow{p} N.$$

ここで "$\xrightarrow{p}$" は部分関数であることを表している。

集合 $N$ はヒープ領域中のノードの集合を意図したものである。但しこれは無限集合であっても良いことにする。ヒープの各ノード $n$ について、その保持する値は $d(n)$ で、フィールド $a$ の指示先は $p(a, n)$ で表現している。また、プログラム変数 $v$ が指すノードを $\rho(v)$ で示す。

ポインタ構造が一つ与えられると、以下のようにしてクリプキ構造を定義することができる。

**定義 3.1** ポインタ構造 $\mathcal{P} = (N, p, d, \rho)$ に対して、クリプキ構造 $\mathcal{M}(\mathcal{P}) = (S, \{R_m\}_{m\in M}, \lambda)$ を次で定める。

- 様相の集合 $M$ を *Fld* とする。原子命題の集合 $AP$ を $\mathit{Val} \cup \mathit{Var}$ とする。
- 状態の集合 $S$ を $N$ として、遷移 $sR_ms'$ を $p(m, s) = s'$ で定める。
- 付値関数 $\lambda$ を以下で定める。

$$k \in \lambda(n) \quad \Leftrightarrow \quad d(n) = k \quad (k \in \mathit{Val})$$
$$v \in \lambda(n) \quad \Leftrightarrow \quad \rho(v) = n \quad (v \in \mathit{Var})$$

クリプキ構造 $\mathcal{M}(\mathcal{P})$ は、本来「ポインタ構造 $\mathcal{P}$ から導出されたクリプキ構造」と呼ぶべきものだが、文脈から分かる場合には敢えて混同して、これもポインタ構造ということにする。

プログラムヒープの状態を表す多くの重要な概念が、ポインタ構造上で 2CTL 論理式を用いることで表現できる。いくつか例を挙げよう。

**例 3.2**
- 性質「変数 $v$ の指すノードから (フィールド $f$ を辿って) 変数 $u$ の指すノードに到達可能である」は 2CTL 式で次のように表現できる。

$$v \Rightarrow \mathrm{E}_f \mathrm{F} u$$

- フィールド $f$ を辿って一巡するループを $f$-**ループ**と呼ぶことにする。性質「変数 $v$ の指すノードが $f$-ループの中にある」は次で表現できる。

$$v \Rightarrow \mathrm{E}_f \mathrm{F} \mathrm{E}_f \mathrm{X} v$$

- 性質「変数 $v$ の指すノードと変数 $u$ の指すノードは、フィールド $f$ を辿って共通のノードに到達できる」は以下で表現される。

$$v \Rightarrow \mathrm{E}_f\mathrm{F}\mathrm{E}_{\bar{f}}\mathrm{F}u$$

当然のことながら、ポインタ構造ではないクリプキ構造も存在する。ポインタ構造は、以下の 2 つの性質を持つ。

($A$) 任意の $s \in S$ および $m \in M$ について、$sR_ms'$ となる $s'$ は高々一つである。

($B$) 任意の $v \in Var$ について、$v \in \lambda(s)$ となる $s$ は高々一つである。

性質 ($A$) を**非分岐性**という。また性質 ($B$) の $v$ のような原子命題は、一般に**ノミナル** (*nominal*) と呼ばれている [4]。

## 4 決定手続き

正形式 2CTL 式 $\alpha_0, \cdots, \alpha_n$ $(= \vec{\alpha})$ および $\chi$ が任意に与えられたとする。本節ではある制約を満す $\vec{\alpha}$-構造で式 $\chi$ を充足し得るか否か、即ち「制限下での $\vec{\alpha}$ 充足可能性」を判定する手続きを示す。ここでいう制約とは、前節の ($A$)、および ($B$) の弱形である以下の ($B'$) である。ただし $V$ は予め指定された $AP$ の部分集合である。

($B'$) 任意の $v \in V$ について、もし $p \in \lambda s, \lambda s'$ ならば、次が成り立つ。

$$\forall \varphi \in cl(\vec{\alpha}, \chi)\ (s \models \varphi \Leftrightarrow s' \models \varphi)$$

ポインタ構造は（$Var$ を $V$ として）これらの制約を満す。従って、もし本節の方法で式 $\chi$ が $\vec{\alpha}$-充足不能と判定されれば、ポインタ構造でもこれを $\vec{\alpha}$-充足できないことになる。

ここで述べる充足可能性判定手続きは、大筋において [3] のそれと同様である。最初に全ての状態を含む大きなタブロを用意して、適当な縮小写像を繰り返し適用することで、実現不可能な状態を削除していく。大きな相違点は、複数のタブロを同時並行的に処理する点である。個々のタブロには、集合 $V$ の各要素がどの状態を指定するかを表す写像が付随しており、これによる制約を受けている。

以下では、閉包 $cl(\vec{\alpha}, \chi)$ を単に $cl$ と書くことにする。閉包に属する、原子命題、および $\mathrm{E}_a\mathrm{X}\psi$ または $\mathrm{A}_a\mathrm{X}\psi$ の形 ($a \in Mod$) の式を全てあつめた集合を**リーン** (*lean*) と呼ぶ。ここでは記号で ln と書こう。リーンの部分集合を**タイプ**と呼ぶ。閉包の要素は全て ln の要素のブール結合で表現できるので、タイプ $\Phi \subseteq \mathrm{ln}$ によりリーン上の付値を指定する（$\Phi$ に属す式が真）と、閉包の各要素の真偽値が確定する。この付値で $\varphi$ が真であることを “$\Phi\Vdash\varphi$” と書く。

原子命題の集合 $V$ が予め与えらえたとして、ここから $\wp(\mathrm{ln}\setminus V)$ への写像を**命名写像**と呼ぶ。命名写像の全体を $\mathcal{F}$ と記する。各 $f \in \mathcal{F}$ に対して：

$$T_f = \{\Phi \subseteq \mathrm{ln} \mid \forall p \in V\ \ p \in \Phi \Rightarrow f(p) = \Phi \setminus V\}$$

と定め、この集合上に遷移関係を導入する。各様相 $m \in M$ について、遷移 $\Phi \xrightarrow{m}_f \Psi$ が成立するのは、任意の $\varphi \in cl$ について、以下の 3 条件が全て満された時と決める。

(i) $\varphi$ が $\mathrm{A}_m\mathrm{X}\psi$ または $\mathrm{E}_m\mathrm{X}\psi$ の形の時、もし $\Phi\Vdash\varphi$ ならば $\Psi\Vdash\psi$ である。

(ii) $\varphi$ が $\mathrm{A}_{\bar{m}}\mathrm{X}\psi$ の形の時、もし $\Psi\Vdash\varphi$ ならば $\Phi\Vdash\psi$ である。

(iii) $\varphi$ が $\mathrm{A}_A(\psi_1\mathrm{U}\psi_0)$ の形で、かつ $m, \bar{m} \in A$ の時、次の 2 命題のいずれかが成り立つ。

- $\Phi\Vdash\varphi$ ならば $\Psi\Vdash\psi_0$
- $\Psi\Vdash\varphi$ ならば $\Phi\Vdash\psi_0$

以下では各 $f$ についてのタブロを一斉に処理する。そこで、これらの直和集合を考えよう。

$$\tilde{T} = \sum_{f \in \mathcal{F}} T_f\ (= \{\langle f, \Phi\rangle \mid f \in \mathcal{F}, \Phi \in T_f\})$$

遷移 $\langle f, \Phi\rangle \xrightarrow{a} \langle g, \Psi\rangle$ は個々の $T_f$ から自然に引き継ぐ。即ち「$f = g$ かつ $\Phi \xrightarrow{a}_f \Psi$」のこととする。また $T \subseteq \tilde{T}$ と $\varphi \in cl$ に対して、次の記法を導入する。

$$[\![\varphi]\!]_T = \{\langle f, \Phi\rangle \in T \mid \Phi\Vdash\varphi\}$$

各 $\varphi \in cl$ に対して、直和タブロ $\tilde{T}$ の冪集合上に写像 $\mathrm{Con}^\varphi$ を定める。これは、指定された集合 $T \subseteq \tilde{T}$ に属す元で「式 $\varphi$ について整合的でない」要素を除去する写像である。以下で集合演算 $X \supset Y$ は $X^c \cup Y$ を意味する。

式 $\varphi$ が $\mathrm{E}_a\mathrm{X}\psi$ の時、写像 $\mathrm{Con}^\varphi$ は：

$$\mathrm{Con}^\varphi(T) = [\![\varphi]\!]_{\tilde{T}} \supset (\xrightarrow{a})^{-1}[[\![\psi]\!]_T]$$

と定める。式 $\varphi$ が $\mathrm{E}_A(\psi_1 \mathrm{U} \psi_0)$ または $\mathrm{A}_A(\psi_1 \mathrm{U} \psi_0)$ の時には、まず $U_0 = [\![\psi_0]\!]_T$ として、以下の拡大写像 $F_T$ の、$U_0$ 上の最小不動点 $U_*$ を求める。

$$F_T(U) = U \cup ([\![\psi_1]\!]_T \cap \mathrm{Step}_T^{\varphi}(U))$$

即ち $F_T^n(U_0) = F_T^{n+1}(U_0)$ となる $F_T^n(U_0)$ を $U_*$ と置く。ただし $\mathrm{Step}_T^{\varphi}$ は以下のものである。

$$\mathrm{Step}_T^{\mathrm{E}_A(\psi_1 \mathrm{U} \psi_0)}(U) = \bigcup_{a \in A} (\stackrel{a}{\longrightarrow})^{-1}[U]$$

$$\mathrm{Step}_T^{\mathrm{A}_A(\psi_1 \mathrm{U} \psi_0)}(U) = \bigcap_{a \in A} \bigcap_{\mathrm{E}_a \mathrm{X} \upsilon \in cl} \mathrm{Con}^{\mathrm{E}_a \mathrm{X} \upsilon}(U)$$

そして集合 $\mathrm{Con}^{\varphi}(T)$ を：

$$\mathrm{Con}^{\varphi}(T) = [\![\varphi]\!]_T \supset U_*$$

と定める。この他の $\varphi$ に対しては、$\mathrm{Con}^{\varphi}$ を恒等写像とする。以上で $\mathrm{Con}^{\varphi}$ の定義を終わる。これらを用いて縮小写像 $G$ を次のように定義する。

$$G(T) = T \cap \bigcap_{\varphi \in cl} \mathrm{Con}^{\varphi}(T)$$

最後に $T_0 = [\![\bigwedge \vec{\alpha}]\!]_{\tilde{T}}$ として、これに含まれる $G$ の最大不動点、即ち $G^n(T_0) = G^{n+1}(T_0)$ となる $G^n(T_0)$ を $T_*$ と決める。このとき次の事実が成り立つ。

**定理 4.1** 式 $\chi$ を充足する $\vec{\alpha}$-構造で、性質 $(A)$ と $(B')$ を満たすものが存在するのは、集合 $[\![\chi]\!]_{T_*}$ が空でない時、かつその時に限る。

**系 4.2** 集合 $[\![\chi]\!]_{T_*}$ が空なら、式 $\chi$ を充足する $\vec{\alpha}$-ポインタ構造は存在しない。ここで $\vec{\alpha}$-ポインタ構造とは、ポインタ構造でかつ $\vec{\alpha}$-構造であるものをいう。

## 5 実装および実行例

ここでは前節で述べた手順を BDD を用いて実装する方法について述べる。本稿で新たに導入した、命名写像に関する部分を除けば、基本的なアイデアは [3] と同じである。

リーンの各元 $\varphi$ に対して、BDD 変数 $x^{\varphi}$ を一つづつ割り当てる。これを**リーン変数**と呼ぶことにする。閉包 $cl$ に属す式 $\varphi$ の BDD 表現 $e(\varphi)$ は、この式をリーンの要素のブール結合で表し、同じ形式で対応するリーン変数を結合してつくる。タイプ $\Phi$ については、これに属す式のリーン変数全てと、属さない式のリーン変数の否定全て、の連言を BDD 表現 $e(\Phi)$ と定める。この時、次が成り立つ。

$$\Phi \| \!\!- \phi \Leftrightarrow e(\Phi) \to e(\varphi) \equiv \mathbf{1}$$

タブロ $T_f$ は $\wp(\ln)$ の部分集合である。その BDD 表現 $e(T_f)$ は、以下の関係を満すように定める。

$$\Phi \in T_f \Leftrightarrow \text{“}e(\Phi) \to e(T_f) \equiv \mathbf{1}\text{”}$$

遷移関係 $\stackrel{a}{\longrightarrow}_f$ は、直積 $\wp(\ln) \times \wp(\ln)$ の部分集合である。これを表現するために、リーン変数 $x$ 各々につき、そのコピー $x'$ を用意する。また一般の BDD 式 $e$ についても、そこに現れるリーン変数を全てコピーで置き換えたものを $e'$ と表記することする。そして遷移関係の BDD 表現を、以下の関係を満すように定める。

$$\Phi \stackrel{a}{\longrightarrow}_f \Psi \Leftrightarrow \text{“}e(\Phi) \wedge e'(\Psi) \to e(\stackrel{a}{\longrightarrow}_f) \equiv \mathbf{1}\text{”}$$

最後に、命名写像と直和タブロの BDD 表現を定めよう。各 $y \in V$ および式 $\varphi \in \ln \backslash V$ の対につき、BDD 変数を一つずつ割当てる。これを $x_y^{\varphi}$ と書く。命名写像 $f$ の BDD 表現は次のものである。

$$e(f) = \bigwedge_{y \in V} \left( \bigwedge_{\varphi \in f(y)} x_y^{\varphi} \wedge \bigwedge_{\varphi \in f(y)^c} \neg x_y^{\varphi} \right)$$

この時、直和タブロ $\tilde{T}$ および遷移 $\stackrel{a}{\longrightarrow}$ に関して以下が成り立つ。

$$\langle f, \Phi \rangle \in \tilde{T} \Leftrightarrow \text{“}e(\Phi) \to e(T_f) \equiv \mathbf{1}\text{”}$$

$$\langle f, \Phi \rangle \stackrel{a}{\longrightarrow} \langle f, \Phi \rangle \Leftrightarrow$$

$$\text{“}e(f) \wedge e(g) \wedge (e(\Phi) \wedge e'(\Psi) \to e(\stackrel{a}{\longrightarrow}_f)) \equiv \mathbf{1}\text{”}$$

これらの表現を用いると、前節の充足可能性判定手続きを BDD 式の計算に翻訳することが出来る。その際に、以下の関係式が有用である。

$$e([\![\varphi]\!]_T) = e(T) \wedge e(\varphi)$$

翻訳結果を擬似コードとして記述したものを図 1 に掲げる。

図 1 の擬似コードを元に作成した Java プログラムの実行例をいくつか示そう。実行時間は、2CTL 式の構文解析にかかる時間も含んでいる。実行環境は以下の通りである。

| | |
|---|---|
| CPU | PentiumM 1.40GHz |
| メモリ | 512MB |
| OS | Microsoft Windows XP |
| JVM | 1.5.0_03 |
| JavaBDD | version 1.0b2 |

```
BDD type, trans[], tabl[], u[];
Boolean satisfiable(α⃗, χ){
   type := ⋀{x^v → ⋀{x^v_φ ↔ e(φ) | φ ∈ ln(α⃗, χ)\V} | v ∈ ln(α⃗, χ) ∩ V}
   foreach(m ∈ M){
      trans[m]:=   ⋀{e(φ) → e'(ψ) | φ = E_m Xψ ∈ cl(α⃗, χ)}
                 ∧⋀{e(φ) → e'(ψ) | φ = A_m Xψ ∈ cl(α⃗, χ)}
                 ∧⋀{e'(ψ) → e(φ) | φ = A_m̄ Xψ ∈ cl(α⃗, χ)}
                 ∧⋀{(e(φ) → e'(ψ0)) ∨ (e'(φ) → e(ψ0)) | φ = A_A(ψ1 U ψ0) ∈ cl(α⃗, χ)}
   }
   int k := 0;
   tabl[0] := type ∧ e(⋀α⃗);
   repeat{
      tabl[k+1] := consis(tabl[k]);
      k:= k+1;
   }until (tabl[k] == tabl[k-1])
   if (tabl[k] ∧ e(χ) != 0) { return true; } else { return false;}
}
BDD consis(T) { return T ∧ ⋀{consisEX(φ, T) | φ = E_a Xψ       ∈ cl(α⃗, χ)}
                         ∧ ⋀{consisEU(φ, T) | φ = E_A(ψ1 U ψ0) ∈ cl(α⃗, χ)}
                         ∧ ⋀{consisAU(φ, T) | φ = E_A(ψ1 U ψ0) ∈ cl(α⃗, χ)};
}
BDD consisEX(E_a Xψ,       T) { return e(E_a Xψ) → ∃x⃗' (T' ∧ e'(ψ) ∧ trans[a]); }
BDD consisEU(E_A(ψ1 U ψ0), T) { return e(E_A(ψ1 U ψ0)) → fulfill(A, ψ1, ψ0, T, stepEU); }
BDD consisAU(A_A(ψ1 U ψ0), T) { return e(A_A(ψ1 U ψ0)) → fulfill(A, ψ1, ψ0, T, stepAU); }
BDD fulfill(A, ψ1, ψ0, T, step) {
   int j := 0;
   u[0]:= T ∧ e(ψ0);
   repeat {
      u[j+1] := u[j] ∨(T ∧ e(ψ1) ∧ step(A, T, u[j]));
      j := j+1;
   } until (u[j] == u[j-1])
   return u[j] ;
}
BDD stepEU(A, T, U){ return ⋁{∃x⃗' (U' ∧ trans[a]) | a ∈ A};}
BDD stepAU(A, T, U){ return ⋀{consisEX(ψ, U) | ψ = E_a Xv ∈ cl(α⃗, χ), a ∈ A} ;}
```

図 1: 擬似コード「BDD を用いた充足性判定手続き」

**例 5.1** 2CTL 論理式 $\alpha_0$ および $\chi$ を以下で定める。また $V = \{v\}$ とする。

$$\begin{aligned} \alpha_0 &= v \Rightarrow \mathrm{E}_f\mathrm{X}k \wedge \mathrm{E}_f\mathrm{X}\neg k \\ \chi &= \mathrm{E}_g\mathrm{X}v \end{aligned}$$

式 $\alpha_0$ をヒープの状態として解釈するなら、「変数 $v$ が指すノードでは、そのフィールド $f$ の指示先として、値が $k$ のノードも $k$ でないノードもある」という意味になる。式 $\chi$ は変数 $v$ が Null でない、つまりどこかのノードを指すことを保証している。

一般のクリプキ構造で $\chi$ が $\bar{\alpha}$-充足可能であるのは明かである。しかしポインタ構造では、性質 $(A)$ があるため、充足は不可能である。我々の判定器は、これを約 281 ミリ秒で “false” と判定した。

**例 5.2** 2CTL 論理式 $\alpha_0, \alpha_1$ と $\chi$ を以下で定める。また $V = \{v\}$ とする。

$$\begin{aligned} \alpha_0 &= v \Rightarrow \mathrm{A}_{\bar{f}}\mathrm{F}\bot \\ \alpha_1 &= v \Rightarrow \mathrm{E}_f\mathrm{F}(\mathrm{E}_f\mathrm{X}v) \\ \chi &= \mathrm{E}_g\mathrm{X}v \end{aligned}$$

式 $\alpha_0$ は、$f$ を逆にたどる無限列がない、つまり整礎性の主張である。自然数のような整礎集合に、全状態で $v$ が真であるよう付値した、クリプキ構造を考えれば、式 $\chi$ が $\bar{\alpha}$-充足可能であることがわかる。しかし、ポインタ構造では式 $\alpha_1$ が $f$-ループの存在を導くため、充足不可能である。これは性質 $(B)$ によるものである。弱形の $(B')$ でも十分かどうかは、あまり自明ではないが、判定器は約 219 ミリ秒で “false” と判定した。

## 6 おわりに

本稿では、プログラムヒープの数学的定式化であるポインタ構造について、二つの特徴的な性質を取り出し、その制約のもとで 2CTL 式の充足可能性を判定する手続きを提案した。また、この手続きを BDD を用いて実装し、ヒープ解析の例題を解かせることで、その有効性を確認した。

今後の研究課題は大きく分けてふたつあると思われる。一つはより精度の高い解析を追求することである。ここで示した方法は、ノミナルに関する条件 $(B)$ をより弱い条件 $(B')$ で代用することで手続きの健全性を保証している。筆者らは、このような条件の弱化を行っても、実用上十分な応用範囲を持つと考えるが、可能ならノミナルによる制約そのものを課した充足正判定法が好もしいことは言うまでもない。

もう一つの課題は実行速度の向上である。本稿で提示した充足可能性判定アルゴリズムは、主にポインタ操作手続きのモデル検査、特に抽象化技法の自動化に応用することを目指して、開発されたものである。具体的で比較的小さな事例として、「ポインタ反転手続き」の述語抽象化 [5] に現れるな 2CTL 式の充足可能性問題を、ここで実装したプログラムに適用してみたところ、所要時間はおよそ 3 秒程度であった。

この事例で、自動述語抽象を完遂するには、これと同レベルの問題を数十回解くことになるため、決して早いとはいえないが現実的な範囲ではある。ただし、中程度以上のサイズのプログラムを扱う場合や、抽象化に使う述語がふえたときなどには、現状では対応できない可能性が高い。速度改善の為の更なる工夫が必須と思われる。

### 参考文献


[1] K. Takahashi and M. Hagiya : Abstraction of Graph Transformation Using Temporal Formulas, *Supplemental Volume of the 2003 International Conference on Dependable Systems and Networks (DSN 2003)*, pp. W-65 to W-66, 2003.

[2] M. Hagiya, K. Takahashi, M. Yamamoto and T. Sato : Analysis of Synchronous and Asynchronous Cellular Automata using Abstraction by Temporal Logic, *FLOPS2004: The Seventh Functional and Logic Programming Symposium*, LNCS volume 2998, pp 7-21, 2004.

[3] 田辺良則, 高橋孝一, 山本光晴, 佐藤貴洋, 萩谷昌己: 「BDD を用いた 2 方向 CTL 論理式充足可能性決定手続きの実装」コンピュータソフトウェア volume 22, pp 154-166. 日本ソフトウェア科学会, 2005.

[4] U. Sattler and M. Y. Vardi.: The Hybrid μ-Calculus. In *Proceedings of the International Joint Conference on Automated Reasoning*, volume 2083 of LNAI, pages 76-91. Springer Verlag, 2001.

[5] Yoshinori Tanabe, Toshinori Takai, Toshifusa Sekizawa and Koichi Takahashi: Preconditions of properties described in CTL for statements manipulating pointers, Supplemental Volume of *the 2005 International Conference on Dependable Systems and Networks (DSN-2005)*, pp.228-234. 2005.