自動証明系と対話型証明支援系の連携によるポインタ操作プログラムの検証について

(Verification of Pointer-Manipulation Programs by Interactive and Automatic Theorem Proving)

湯浅 能史 † 武山 誠 †
Yoshifumi YUASA Makoto TAKEYAMA

関澤 俊弦 †‡ 田辺 良則 †§ 高橋 孝一 †
Toshifusa SEKIZAWA Yoshinori TANABE Koichi TAKAHASHI

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

‡ 大阪大学 大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University

§ 東京大学 大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo

対話型証明支援系と自動証明系を組合せたプログラム検証の一例を紹介する。証明支援系としては Martin-Löf 型理論をベースとした Agda を、自動証明系としては様相論理ベースの MLAT(Modal Logic Abstraction Tool) を用いる。ポインタ操作言語 PML で書かれたリスト反転プログラムを対象に、Hoare 論理に基いてその性質を検証する。プログラムは Agda によりサブプログラムに分割され、ループ不変式の確認等、人手で困難な個所は MLAT により自動証明される。

1 はじめに

ソフトウェアの検証とは、プログラムが要求された性質を持つ事を保証する事である。これはプログラムを対象とした定理証明の一種と捉えられる。ソフトウェア検証に現れる証明は、高度の厳密さが求められ、また一般には大規模なものとなるため、計算機の利用は欠かせない。

計算機による定理証明へのアプローチには、大きく分けて二つの考えかたがある。一つは、ユーザが示そうとする定理のみを与えて、あとは計算機が自動的に真偽を判定するというものである。この考えに基く定理証明器は**自動証明系**と呼ばれる。ここでは少し広い意味で捉えて、自動モデル検査器のような検証ツールもこの範疇に含めることにする。

もう一つは、定理だけでなく証明もユーザが与えて、計算機がその整合性を検査する、というものである。但し、完成した証明を与えて一度に検査させることよりも、ユーザが推論を積上げて徐々に証明を完成させる過程の各時点での、部分的な整合性確認を計算機で支援する、という方法がむしろ一般的である。そこで、この考えに基く定理証明器を**対話型証明支援系**と呼ぶ。

通常これらの証明器のいずれか一方のみにより、十分な検証を効率的に行うことは困難である。自動証明系はある意味手軽ではあるが、決定不可能性のような原理的な限界や、また処理時間や記憶領域などの計算資源的な限界に阻まれ、証明できる性質はかなり限定的である。一方、証明支援系を用いた証明では示せる性質に制約はないが、定理適用条件の確

*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」の一部として実施された。

認や繁雑な場合分け等、面倒な処理を人手により行う必要がある。また、自明と思われる補題を逐一示さねばならない点も煩わしい。そこで、実際の検証はこれらふたつの長所を生かし、また短所を補いつつ進めるのが妥当と思われる。

そのような協調の一例として、対話型証明支援系 Agda のプラグイン機構に遷移系自動抽象化器 MLAT (Modal Logic Abstraction Tool) を接続して行う検証法を提案する。検証の対象は、理想化された手続き型言語 PML (Pointer Manipulation Laguage) で書かれたポインタ操作プログラムである。枠組みとなる論理としては、良く知られた Hoare 論理 [5] を用いる。

本稿ではこの手法を開発するための第一歩として、検証の基本的な枠組である PML-Hoare 論理を Agda 上に定義し、MLAT との連携により簡単なプログラム検証の試行を行う。例題として取り上げた「リスト反転プログラム」は小さなものではあるが、より大きなプログラムを扱う際に得られるであろうメリットや、課題となる問題点など、今後の研究のために有益な情報が得られたと考えている。

本稿の構成について述べる。まず次節では Agda と MLAT について簡単な紹介をする。解説は最小限に留めたので、詳しい情報については参考文献にあたって欲しい。第 3 節では、プログラムの操作対象である「ポインタ構造」について述べる。数学的な定義を与え、これを様相論理の枠組でとらえる方法について解説した。続く第 4 節では、ポインタ操作言語 PML の定義と Hoare 論理について述べた。第 5 節では、3,4 節で提示した検証の枠組の Agda 上での表現について、また MLAT プラグインの扱いについて述べる。またこれらを利用した例題として、リスト反転プログラムの検証について解説する。第 6 節はまとめである。本試行結果の考察と今後の展望について述べる。

2 Agda と MLAT について

対話型定理証明支援系 Agda は、スウェーデン Chalmers 工科大学で開発された、Martin-Löf 型理論に基礎を置く定理証明器である。Curry-Howard 対応に従って論理式を型と捉え、その型をもつ λ 項、即ち証明をユーザが定義する。Agda は定義式の型検査、つまり証明が正しいかどうか確認をしてくれる。証明の定義は、関数型プログラミング言語（特に Haskell）の関数定義と類似の構文で行う。以下に Agda による証明の一例を示す。

```
commCnj (A,B :: Set) :: A /\ B -> B /\ A
    = \(ab :: A /\ B) ->
          and (elimAndR ab)
              (elimAndL ab)
```

Agda はプラグイン機構を備えており、これを通じて外部の自動証明器を利用することもできる。定義式の右辺に、証明の代りに外部証明器を起動するキーワードを書いておくと、Agda は型検査の際に、これを呼出して真偽の判定を委ねる。現在までに HOL, NuSMV などとの接合実績がある。

証明支援系 Agda についての詳しい情報は “Agda official web page” [1] を、プラグイン機構については [2] の予稿を参照されたい。

一方、MLAT (Modal Logic Abstraction Tool) は、産総研システム検証研修センターで開発中の遷移系自動抽象化器である。ポインタを操作する手続き型プログラムを入力すると、対応する状態遷移系を作成し、ユーザの与えた述語セットを用いて述語抽象化を自動的に行うツールである。MLAT の原理の詳細および最近の開発の状況については [3] の予稿が詳しい。

MLAT では抽象状態間の遷移関係を発見する為に、プログラム断片に関する前条件と後条件の整合性チェックを行っている。本研究では、この機能を MLAT が外部提供する関数として新たに実装、Agda のプラグインとして提供した。

3 ポインタ構造

プログラムの操作対象はポインタである。まずこれに数学的な表現を与えよう。フィールド名セット Fld、変数名セット Var、および値表現セット Val をが与えられたとして、5 つ組 $(S, \{p_f\}_{f\in\mathsf{Fld}}, \alpha, \beta, \mathbf{0})$ で以下の条件を満足するものを考える。これを**ポインタ構造**と呼ぶ。

- S は集合。各 $f \in \mathsf{Fld}$ について $p_f : S \to S$.
- $\alpha : \mathsf{Var} \to S$ かつ $\beta : \mathsf{Val} \to \wp S$.
- $\mathbf{0} \in S$ であり、各 $f \in \mathsf{Fld}$ について $p_f\mathbf{0} = \mathbf{0}$.

関数 p_f はポインタの指示先を、α は変数の内容を、β はある値を保持するノードの範囲を示すものであ

る。ヌルポインタは（何も指し示さないというのではなく）予め指定された特別なノード **0** を指す事として表現する。ヌルノード **0** は全てのフィールドで自分自身を指すと決める。

自動抽象化器 MLAT では、ポインタ構造を Kripke 構造の一種とみて、その特性を記述するために多重様相論理式を用いる。具体的には、各 $f \in \mathsf{Fld}$ について $f, \bar{f}$ をそれぞれ順・逆の様相名とみなし、遷移関係 $\xrightarrow{f}$ を次のように決める。

$$s \xrightarrow{f} t \Longleftrightarrow t \xrightarrow{\bar{f}} s \overset{\triangle}{\Longleftrightarrow} p_f s = t$$

以下、順/逆様相名の集合を記号 M で表す。原子命題記号としては、変数 $x \in \mathsf{Var}$、値表現 $b \in \mathsf{Val}$ および定数 NULL を用意する。それぞれに対する付値は以下の通り。

$$\begin{aligned} s \models x &\overset{\triangle}{\Longleftrightarrow} s = \alpha x \\ s \models b &\overset{\triangle}{\Longleftrightarrow} s \in \beta b \\ s \models \mathrm{NULL} &\overset{\triangle}{\Longleftrightarrow} s = \mathbf{0} \end{aligned}$$

即ち、変数 x がノード s を指す時に原子命題 x は s で真であり、また値表現 b の意図する値をノード s が保持するとき命題 b は s で真とする。原子命題 NULL は s がヌルノードの時、かつその時のみ真である。

ポインタ構造の性質を記述する論理式は、多重様相計算木論理 (Multi-modal CTL) のそれを用いる。様相名の集合は上述の M だが、これは単なるラベル集合ではなく、方向を反転する演算 $(f \mapsto \bar{f})$ が作用している。そこで、特にこの体系を**２方向 CTL**、もしくは単に 2CTL と呼ぶ。詳細については文献 [4] を参照されたい。

二つの 2CTL 式 φ と ψ に対して「全てのノードで、式 φ が真なら式 ψ も真となる」ことを "$\varphi \vdash \psi$" という式で表す。特に φ が変数の時に、これを「基本 P-式」と呼び、そのブール結合を **P-式**と呼ぶ。

本稿末尾の図 2 に、式 "$\varphi \vdash \psi$" の成否を判定する推論体系を示した。ただし、これが完全かどうかは現時点では不明である。（健全性はポインタ構造の定義により容易に確認できる。）

4 ポインタ操作言語 PML

本研究での検証の対象は PML という手続き型言語で書かれたプログラムである。これは制御構造として条件分岐と反復のみを含む、理想化された抽象言語（これを **While 言語**と呼ぶ）に、ポインタ操作の基本命令をはめこんで具体化したものである。そこで、まずこの抽象言語とそれを対象とした一般的な Hoare 論理について簡単に解説する。今、基本条件文セット AS と基本命令セット AC が予め与えられているとしよう。基本条件文のブール結合を**条件文**と呼び、その集合を S と記す。すなわち：

$$\mathsf{S} \quad ::= \quad \mathsf{AS} \mid (\,!\,\mathsf{S}) \mid (\mathsf{S}\,\&\&\,\mathsf{S}) \mid (\mathsf{S}\,||\,\mathsf{S})$$

である。While 言語の命令 C およびプログラム P を以下の BNF で定義する。

$$\begin{aligned} \mathsf{C} \quad &::= \quad \mathsf{AC} \mid \mathtt{if}\,\mathsf{S}\,\mathtt{then}\,\{\mathsf{P}\}\,\mathtt{else}\,\{\mathsf{P}\} \\ &\qquad\quad \mid \mathtt{while}\,\mathsf{S}\,\mathtt{do}\,\{\mathsf{P}\} \\ \mathsf{P} \quad &::= \quad \lambda \mid \mathsf{C};\mathsf{P} \end{aligned}$$

ここで λ は空の文字列を表す記号である。

次に Hoare 論理について述べる。まず、操作対象の状態を記述する適当な形式言語 L を一つ固定する。これは命題論理の拡張でなくてはならない。また基本原子条件文 s のそれぞれに、言語 L での解釈として論理式 $\bar{s}$ を与える。一般の条件文の解釈は、これをブール結合に自然に拡張して定義する。

While 言語の命令 c および言語 L の論理式 P, Q に対し、形式 "$P\,\{c\}\,Q$" を Hoare の三つ組、または **Hoare 式**と呼ぶ。これは「操作対象が P を満たす時、命令 c を行えば、実行後は Q を満たす状態となる」ことを表す式である。プログラム cs についての Hoare 式 "$P\,\{cs\}\,Q$" も同様の意味である。Hoare 式の成否を推論する論理体系は、Hoare 論理と呼ばれており、様々な表現が可能だが、ここでは以下のものを用いることにする。

$$\frac{P \Rightarrow \mathrm{wp}_c\, Q}{P\,\{c\}\,Q}\,(\mathrm{atm}) \qquad (\text{for } c \in \mathsf{AC})$$

$$\frac{P \wedge \bar{s}\,\{cs_0\}\,Q \quad P \wedge \neg\bar{s}\,\{cs_1\}\,Q}{P\,\{\mathtt{if}\,s\,\mathtt{then}\,\{cs_0\}\,\mathtt{else}\,\{cs_1\}\}\,Q}\,(\mathrm{ite})$$

$$\frac{P \Rightarrow R \quad R \wedge \bar{s}\,\{cs\}\,R \quad R \wedge \neg\bar{s} \Rightarrow Q}{P\,\{\mathtt{while}\,s\,\mathtt{do}\,\{cs\}\}\,Q}\,(\mathrm{whl})$$

$$\frac{P \Rightarrow Q}{P\,\{\lambda\}\,Q}\,(\mathrm{nil}) \qquad \frac{P\,\{c\}\,R \quad R\,\{cs\}\,Q}{P\,\{c;cs\}\,Q}\,(\mathrm{cns})$$

ここで $P \Rightarrow Q$ は $\neg P \vee Q$ の略記である。また、推論図式 atm の上段にある "$\mathrm{wp}_c\, Q$" は、命令 c に関する Q の**最弱前条件**と呼ばれるもので、以下の条件を満す L の式 $\bar{Q}$ の内で最も弱いものをいう。

> 「操作の対象が $\bar{Q}$ を満すなら、命令 c を行った後には Q を満すようになる。」

最弱前条件の存在は一般には保証されない。

以上が While 言語とその Hoare 論理についての一般論である。本稿で扱う PML では、操作対象は前節で導入したポインタ構造であり、状態の表現は P-式で行う。基本条件文は以下のようにとる。括弧内は P-式での解釈である。

`x`	`==Null`	($x \vdash$ NULL)
`x.val`	`==b`	($x \vdash b$)
`x`	`==y`	($x \vdash y$)

ここで x, y は任意の変数を、b は任意の値表現を表している。また基本命令は以下の 6 つである（f はフィールド名。）

`x :=Null`, `x.val:=b`, `x :=x`,
`x :=x.f`, `x.f :=y`, `x :=new`

それぞれの基本命令の直観的な意味は明らかであろう。正確な意味は最弱前条件で与えられるが、その計算法は複雑である。また、現時点では Agda 上に実装をしていないため、ここでは最弱前条件の計算法の詳細は述べない。文献 [3] を参照せよ。

ポインタ構造に関する推論体系（図 2）と、上述の PML に関する Hoare 論理を併せて **PML-Hoare 論理**と呼ぶことにする。

5 Agda-MLAT 連携による検証

まず、PML-Hoare 論理の Agda での表現について述べる。ポインタ構造に関する推論体系と Hoare 論理の推論系の二つがあるが、それぞれを異なる符号化方式に則って表現する。[1]

ポインタ構造の推論系は浅い符号化 (shallow embedding) によりコードした。これが、通常良く行われる符号化方式であり、第 2 節で紹介した証明例もこの方法に依っている。この符号化法では 2CTL 式や P-式を Agda の既存型により解釈して、推論規則は上式から下式への関数として定義する。不動点演算子に関する部分は、永山-武山-池上による μ-計算ライブラリを利用した。

一方、プログラムに関する Hoare 論理は深い符号化法 (deep embedding) で表現する。まず、Hoare 式に対応するデータ型を新たに導入し、推論規則をこの型のデータ構成子とする。

MLAT プラグインの呼び出しは、Hoare 論理の証明作成中に利用する事ができる。示そうとする Hoare 式の証明定義式の右辺に、MLAT 起動のためのキーワード「`external "mlat"`」を書いておけばよい。但し現状では、MLAT による自動化を利用できるのは、基本命令もしくはそれらの列についての Hoare 式の場合だけである。また特別な場合として、空プログラムについての Hoare 式を考えると、P-式の含意関係の判定にもこのプラグインを利用できる。

検証例： 次のプログラム “`reverse`” を対象とする。これはポインタで繋がれたリストを反転するプログラムである。

```
y := Null ;
while (not (x == Null)) do {
  t      := y ;
  y      := x ;
  x      := x.next ;
  y.next := t ;
} ;
```

フィールドは “`next`” のみで、リストの次の要素へのポインタを表す。変数 x がリストの先頭を指した状態でこのプログラムを実行すると、終了時にはリストは逆順となり、その先頭を変数 y が指す。

以下この性質を検証する。まず略号を幾つか導入しよう。略号の導入は、Agda では関数の定義である。

$$
\begin{aligned}
\mathrm{Equal}(x,y) &:= x \vdash y \\
\mathrm{Next}(x,y) &:= x \vdash \mathrm{E}_{\mathrm{next}}\mathrm{X}\, y \\
\mathrm{Nullp}(x) &:= x \vdash \mathrm{NULL} \\
\mathrm{Reach}(x,y) &:= x \vdash \mathrm{E}_{\mathrm{next}}\mathrm{F}\, y \\
\mathrm{Listp}(x) &:= x \vdash \mathrm{E}_{\mathrm{next}}\mathrm{F}\,\mathrm{NULL}
\end{aligned}
$$

それぞれ「変数 x と y は同じノードを指す」「変数 x の指すノードが `next` で指すノードを、変数 y が指す」「変数 x はヌルノードを指す」「変数 x の指すノードから `next` フィールドを辿って、変数 y の指すノードに到達する」「変数 x はあるリストの先頭のノードを指す」を意味する。

これらによりプログラム `reverse` の前条件 Pre と後条件 Pos を以下で定める。検証すべき Hoare 式は

[1] このような形になったのは、作業上の理由であり、理論上/実用上の深い意味はない。当初、全てを深い埋め込みで行う予定だったが、途中で方針を変更し一部を既存のライブラリに依存することした。ライブラリが浅い符号化に依っていたため、併用の形となった。

“Pre {`reverse`} Pos” である。

$$\begin{aligned} \mathrm{Pre} &:= \mathrm{Listp}(x) \wedge \mathrm{Reach}(x,u) \wedge \\ &\qquad \mathrm{Next}(u,v) \wedge \neg\mathrm{Nullp}(v) \\ \mathrm{Pos} &:= \mathrm{Next}(v,u) \end{aligned}$$

即ち、任意の u, v が x から到達可能でかつこの順に並んでいれば、実行後には逆に v, u の順に並んでいる、という事を示したい。

図 1 に Agda による証明を示した。証明中の：

```
...  = external "mlat"
```

とある個所で MLAT が呼ばれ、左辺の型（= Hoare 式）が正しいことを確認している。ちなみに、この証明に含まれる補題 3 つを示すのに MLAT が要した時間は、総計 10 秒であった。

図 1 の式 Inv は、いわゆる「ループ不変式」である。ループでは変数 x がリストの先頭から末尾に向って動いていく。式 Inv の選言を構成する 3 状態：

$$\begin{aligned} &\mathrm{Reach}(x,u) \ \wedge\ \mathrm{Next}(u,v), \\ &\neg\mathrm{Reach}(x,u) \ \wedge\ \mathrm{Equal}(x,v) \wedge \mathrm{Equal}(yu), \\ &\neg\mathrm{Reach}(x,u) \ \wedge\ \mathrm{Next}(v,u) \end{aligned}$$

は、それぞれ「x が u に到るまで」「x が u を通過したとき（直後）」「それ以降」を表す。

これを得るには、およその形を予想した上で、Agda での対話的証明を進めながら、試行錯誤で条件の細部を確定していった。これに際しても、MLAT プラグインの提供する自動恒真性判定が有効であった。例えば、上での $\neg\mathrm{Reach}(x,u)$ の必要性は、そのようにして発見されたものである。プラグインがなければ、しばらくは条件の足りないまま不可能な証明に努力を費す場面であったと考えられる。

6 まとめ

本稿では、対話型証明支援系 Agda に自動抽象化器 MLAT が提供する自動証明器を接続して、ポインタ操作プログラムを検証する方法を導入し、簡単なプログラムの検証を行った。大きなプログラムや実用的なプログラムの検証は今後の課題である。

今回、検証例として用いたポインタ反転プログラムは、実際には MLAT 単独でも検証可能である。しかし、多くの記憶領域を要し時間もかなり (40 秒程度) かかる。これは、このような短いプログラムであっても、全体の完全な自動化を行おうとすると、考慮すべき状態遷移の数も、また個々の遷移可能性を判定するのに必要な計算資源も、ともに膨大なものとなるためである。今回の試行では、検証を 3 分割し更にループ不変式を明示的に与えることで、自動証明部分が要した時間の総計は 1/4 となった。この時間短縮がどの要因によるものかは不明であるが、何れにしても大きなプログラムの検証では、より顕著になる公算は高い。

別の課題として、ポインタ構造の性質記述言語の強化がある。現在開発中の MLAT の次期バージョンでは記述言語を 2CTL ではなく、無交代 2 方向 μ 計算とする予定である。本稿で与えた PML-Hoare 論理は、実際には既に μ 計算に対応したものとのとなっているため、これにより多様な検証項目を扱うことが可能となる。

この他、最弱前条件の Agda 上での実装や、MLAT の（部分機能ではなく）本来の機能そのものをプラグインとし利用する方法の模索なども重要な課題である。また、MLAT から Agda への返値として、単なる真偽値に加え付加的な情報（真の場合には証明のヒント、偽なら判例）を渡せるようにする、という改良も有意義である。ここで受け取った情報を Agda での証明に役立てることができるであろう。

謝辞

産業技術総合研究所 池上大介氏には、本研究で用いた Agda μ 計算ライブラリの改良に際してご協力頂いた。氏に感謝する。

参考文献

[1] Agda Official Web Site. `"http://agda.sourceforge.net/"`

[2] 池上大介: Agda プラグイン機構. 日本ソフトウェア科学会 第 22 回大会 一般講演（2005）

[3] 田辺 良則, 関澤 俊弦, 湯浅 能史, 高橋 孝一: 様相論理を使用したヒープ検証方式. 第 3 回ディペンダブルソフトウェアワークショップ (DSW'06).

[4] 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, Vol.22, No.3 (2005), pp.154-166

[5] C.A. R. Hoare, An Axomatic Basis for Computer Programming, CACM 12 (10), 1969, 576-580.

```
loopBody :: Program = [  t       :=  y
                      ,  y       :=  x
                      ,  x       :=.(x,next)      --   "x := x.next"
                      , (y,next).:= t ]           --   "y.next := t"
reverse  :: Program = [ setNull y                 --   "y := NULL"
                      , while_do (not (null x))
                          loopBody
                       ]

Equal :: (x, y :: Var) -> PFormula
Equal x y = x ===> nom y                          --   "x |- y"
Next :: (x, y :: Var) -> PFormula
Next x y = x ===> EX (nom y)                      --   "x |- EX y"
Nullp :: Var -> PFormula
Nullp x = x ===> Null                             --   "x |- Null"
Reach :: (x, y :: Var) -> PFormula
Reach x y = x ===> EF (nom y)                     --   "x |- EF y"
ListP :: Var -> PFormula
ListP x = x ===> EF Null                          --   "x |- EF Null"

Pre :: PFormula = ListP x && Reach u x && Next u v && not (Nullp v)
Pos :: PFormula = Next v u

Inv :: PFormula = ListP x && not (Nullp v) &&
                      (     Reach x u  && Next u v ||
                       not (Reach x u) && Equal x v && Equal y u ||
                       not (Reach x u) && Next v u)

rev :: HT Pre reverse Pos
    = let lem1 :: HTc Pre (setNull y) Inv  = external "mlat"
          lem2 :: HTp (Inv && not (Nullp x)) loopBody Inv = external "mlat"
          lem3 :: PFImp (Inv && not (not (Nullp x))) Pos  = external "mlat"
      in cns_ lem1 (cns_ (whl_ id_ lem2 lem3) (nil_ id_))
```

図 1: リスト反転プログラムの検証

命題論理の公理：

$$\varphi \vdash \varphi \ (\text{id}) \qquad \bot \vdash \psi \ (\text{leftBot}) \qquad \varphi \vdash \top \ (\text{rghtTop})$$

命題論理の推論規則：

$$\frac{\varphi \vdash \chi \quad \chi \vdash \psi}{\varphi \vdash \psi}\ (\text{sequencial})$$

$$\frac{\varphi_1 \vdash \psi}{\varphi_0 \wedge \varphi_1 \vdash \psi}\ (\text{leftCnj0}) \qquad \frac{\varphi_0 \vdash \psi}{\varphi_0 \wedge \varphi_1 \vdash \psi}\ (\text{leftCnj1}) \qquad \frac{\varphi \vdash \psi_0 \quad \varphi \vdash \psi_1}{\varphi \vdash \psi_0 \wedge \psi_1}\ (\text{rghtCnj})$$

$$\frac{\varphi \vdash \psi_1}{\varphi \vdash \psi_0 \vee \psi_1}\ (\text{rghtDsj0}) \qquad \frac{\varphi \vdash \psi_0}{\varphi \vdash \psi_0 \vee \psi_1}\ (\text{rghtDsj1}) \qquad \frac{\varphi_0 \vdash \psi \quad \varphi_1 \vdash \psi}{\varphi_0 \vee \varphi_1 \vdash \psi}\ (\text{leftDsj})$$

$$\frac{\psi_0 \wedge \varphi \vdash \psi_1}{\psi_0 \vdash \neg\varphi \vee \psi_1}\ (\text{rghtNot}) \qquad \frac{\psi_0 \vdash \varphi \vee \psi_1}{\psi_0 \wedge \neg\varphi \vdash \psi_1}\ (\text{leftNot})$$

2 方向 CTL の推論規則：

$$\frac{\varphi \vdash \psi}{\mathrm{A}_m\mathrm{X}\varphi \vdash \mathrm{A}_m\mathrm{X}\psi}\ (\text{monoAX}) \qquad \frac{\varphi \vdash \psi}{\mathrm{E}_m\mathrm{X}\varphi \vdash \mathrm{E}_m\mathrm{X}\psi}\ (\text{monoEX})$$

$$\frac{\mathrm{E}_m\mathrm{X}\varphi \vdash \psi}{\varphi \vdash \mathrm{A}_{\bar{m}}\mathrm{X}\psi}\ (\text{adjAX}) \qquad \frac{\varphi \vdash \mathrm{A}_m\mathrm{X}\psi}{\mathrm{E}_{\bar{m}}\mathrm{X}\varphi \vdash \psi}\ (\text{adjEX})$$

$$\frac{\varphi \vdash \neg\mathrm{E}_m\mathrm{X}\ \psi}{\varphi \vdash \mathrm{A}_m\mathrm{X}\neg\psi}\ (\text{notAX}) \qquad \frac{\neg\mathrm{A}_m\mathrm{X}\ \varphi \vdash \psi}{\mathrm{E}_m\mathrm{X}\neg\varphi \vdash \psi}\ (\text{notEX})$$

$$\frac{\psi \vdash \varphi_0 \vee (\varphi_1 \wedge \mathrm{E}_m\mathrm{X}\,\mathrm{E}_m(\varphi_1\mathrm{U}\varphi_0))}{\psi \vdash \mathrm{E}_m(\varphi_1\mathrm{U}\varphi_0)}\ (\text{fixEU}) \qquad \frac{\varphi_0 \vee (\varphi_1 \wedge \mathrm{E}_m\mathrm{X}\psi) \vdash \psi}{\mathrm{E}_m(\varphi_1\mathrm{U}\varphi_0) \vdash \psi}\ (\text{lstEU})$$

$$\frac{\psi \vdash \varphi_0 \vee (\varphi_1 \wedge \mathrm{A}_m\mathrm{X}\,\mathrm{A}_m(\varphi_1\mathrm{U}\varphi_0))}{\psi \vdash \mathrm{A}_m(\varphi_1\mathrm{U}\varphi_0)}\ (\text{fixAU}) \qquad \frac{\varphi_0 \vee (\varphi_1 \wedge \mathrm{A}_m\mathrm{X}\psi) \vdash \psi}{\mathrm{A}_m(\varphi_1\mathrm{U}\varphi_0) \vdash \psi}\ (\text{lstAU})$$

$$\frac{(\mathrm{A}_m\mathrm{X}\,\mathrm{A}_m(\varphi_1\mathrm{R}\varphi_0) \vee \varphi_1) \wedge \varphi_0 \vdash \psi}{\mathrm{A}_m(\varphi_1\mathrm{R}\varphi_0) \vdash \psi}\ (\text{fixAR}) \qquad \frac{\psi \vdash (\mathrm{A}_m\mathrm{X}\psi \vee \varphi_1) \wedge \varphi_0}{\psi \vdash \mathrm{A}_m(\varphi_1\mathrm{R}\varphi_0)}\ (\text{greAR})$$

$$\frac{(\mathrm{E}_m\mathrm{X}\,\mathrm{E}_m(\varphi_1\mathrm{R}\varphi_0) \vee \varphi_1) \wedge \varphi_0 \vdash \psi}{\mathrm{E}_m(\varphi_1\mathrm{R}\varphi_0) \vdash \psi}\ (\text{fixER}) \qquad \frac{\psi \vdash (\mathrm{E}_m\mathrm{X}\psi \vee \varphi_1) \wedge \varphi_0}{\psi \vdash \mathrm{E}_m(\varphi_1\mathrm{R}\varphi_0)}\ (\text{greER})$$

Null の公理 (nullnext)： $\mathrm{NULL} \vdash \mathrm{E}_f\mathrm{X}\,\mathrm{NULL}$

非分岐性の公理 (unique)： $\mathrm{E}_f\mathrm{X}\varphi \vdash \mathrm{A}_f\mathrm{X}\varphi$ 　トータル性の公理 (total)： $\mathrm{A}_f\mathrm{X}\varphi \vdash \mathrm{E}_f\mathrm{X}\varphi$

Nominal 推論 1 (nom1)： $\dfrac{x \not\vdash \neg\varphi}{x \vdash \varphi}$ 　Nominal 推論 2 (nom2)： $\dfrac{x \vdash \varphi}{x \not\vdash \neg\varphi}$

上記規則において、m は任意の様相名、f は任意のフィールド名、x は任意の変数記号、$\varphi, \varphi_0, \varphi_1, \psi, \psi_0, \psi_1$ は任意の 2CTL 式。

図 2: ポインタ構造の推論規則