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
SS2007-30(2007-10)

導出木からのループ検出による論理プログラムの非停止性証明法

水谷 知博† 西田 直樹†† 酒井 正彦†† 坂部 俊樹†† 草刈 圭一朗††

†,†† 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: †mizutani@sakabe.i.is.nagoya-u.ac.jp, ††{nishida,sakai,sakabe,kusakari}@is.nagoya-u.ac.jp

あらまし 本稿では，論理プログラムの非停止性を自動的に証明する手法を提案する．本手法では，与えられた質問パターンを代表する質問から起こり得る導出を木で構成し，その木の中からループを検出することにより質問パターンに対する非停止性を示す．具体的には，質問パターンの入力と出力を変数に置き換えた質問から導出木を構成するが，その導出やループ検出の際に入力に対応する変数を特別扱いすることにより健全性を確保する．実際に本手法を実現したツールを作成し，Termination Competition 2007 の論理プログラム部門で与えられているサンプルプログラムを対象とした実験を行い，既存の研究との比較を行う．

キーワード Prolog，停止性，ナローイング

Proving Non-termination of Logic Programs by Detecting Loops in Derivation Trees

Tomohiro MIZUTANI†, Naoki NISHIDA††, Masahiko SAKAI††,
Toshiki SAKABE††, and Keiichirou KUSAKARI††

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

Abstract In this paper, we present a method for automatically proving non-termination of logic programs. Given a program and a question pattern, the method proves non-termination by detecting a loop in the derivation tree constructed from the question that represents an arbitrary questions for the question pattern. For soundness, we distinguish variables given as arguments of predicates, and we keep the distinction in the derivation. We also report an implementation of the method, and compare it with other tools by applying it to programs in LP category of Termination Competition 2007.

Key words Prolog, termination, narrowing

1. はじめに

論理プログラムや関数型プログラムの停止性に関する研究は，これまでに盛んに行われている．停止性は一般には決定不能であるが，停止性を持つための十分条件に関しては多くの成果が存在する [8]．近年では，停止性を自動証明するツールの開発も盛んになっている [11] [6] [9]．一方，非停止性に関する研究も行われている [10]．プログラムの非停止性を自動的に判定することができれば，プログラムのデバッグの助けになる．例えば，停止性を持つプログラムを作成しているときに，非停止性を持つことが示されると，プログラムにバグがあることがわかる．非停止性の証明の多くは，計算したい質問から到達できるループ [1] を発見することで行われる．

論理プログラムには，関数型プログラムの場合とは異なり，プログラマが意図しない非停止性が存在する．例えば，自然数 0, 1, 2, ··· を 0, s(0), s(s(0)), ··· で，加算と乗算をそれぞれ add, mul で表現すると，自然数の乗算は以下の論理プログラムで計算できる．

```
add(0,Y,Y).
add(s(X),Y,s(Z)) :- add(X,Y,Z).
mul(0,Y,0).
mul(X,0,0).
mul(s(X),s(Y),s(Z)) :- mul(X,s(Y),W), add(W,Y,Z).
```

ここで，述語 mul は $\mathtt{mul}(m, n, m \times n)$ を満たす述語である．上記

の乗算プログラムは $m \times n$ を計算する，すなわち，$\texttt{mul}(m,n,X)$ を質問として計算するよう設計されている．mul の第一，第二引数に具体的な値 m,n を与えたときに，述語 mul を満たす第三引数の値を求めるために作られており，プログラムの最終行の本体のアトムの並びもそれに応じて決まっている．一方，n,$m \times n$ から m を計算するために $\texttt{mul}(X,n,n \times m)$ から計算した場合には，プログラムにより $\texttt{mul}(X',n,W)$，$\texttt{add}(W,n-1,n \times m-1)$ に変換される．しかし，$\texttt{mul}(X',n,W)$ を計算するには W の情報が未知であり，計算が停止しない．この際には，$n \times m-1$ の値は与えられているので，最終行を以下のように書くと $\texttt{mul}(X,n,n \times m)$ の計算は停止する．

```
mul(s(X),s(Y),s(Z)) :- add(W,Y,Z), mul(X,s(Y),W).
```

このような差が生じる原因は，論理プログラムが特定の質問パターンを意識して書かれている，すなわち，節のクエリ中のアトムの並び順が質問の形式に依存して定まっていることである．したがって，論理プログラムの停止性の議論は，質問パターンと呼ばれる，質問の入力と出力を定めるパターンを固定して行われている．ある質問パターンを意識して書かたれた論理プログラムは，それ以外の質問パターンに対しては停止性を持たないことも多い．このような非停止性は見落とされやすく，プログラマやユーザが停止性がないことに気付いていない場合もある．よって，論理プログラムでの非停止証明は，関数型プログラムのそれより有用である．

本稿では，指定された質問パターンに対し，導出中にループを生じる質問が存在することを示すことにより，論理プログラムの非停止性を証明する手法を提案する．本手法は，与えられたプログラムを解析することにより非停止性を示すのではなく，与えられたプログラムによって起こり得るすべての導出を求めながら，導出中のループを検出することで非停止性を示す．

与えれた質問パターンに対する論理プログラムの非停止性を示すためには，無限の導出を引き起こす質問パターンの入力の値（基礎項）を求める必要がある．一般には，質問パターンから考えられる具体的な質問は無限個あり，これらのクエリからのすべての導出を求めることは非常に困難である．そこで，本手法では，質問パターンを表現した一般的な質問から起こり得る導出を木で求める．そして，そのような導出を表現する木の中にループが存在するかを検査する．その導出やループ検出の際に質問パターンの入力に対応する変数を特別扱いすることにより健全性を確保する．

本手法で用いる単一化には Occur-check の機能があることにする．Occur-check とはある項が自分自身を含むような項との単一化を防ぐための機能である．Occur-check を用いる理由は，ループや導出木の議論，および実装を単純にするためである．しかし，Occur-check のあるシステムで非停止性を示すことができたならば，Occur-check のないシステムでも非停止性が保証される．つまり，本手法で非停止性を示せたならば，Occur-check のない場合の非停止性も示される．

本稿は次のように構成される．2 節では，論理プログラムに関する記法と停止性について説明する．3 節では，論理プログラムの導出のループの概念を与え，4 節では，本稿で提案する手法のアイディアを説明する．5 節では，4 節で説明したアイディアを定式化し，非停止性の十分条件を示す．6 節では，本手法の実装を紹介し，他研究との比較を行う．7 節では，まとめと今後の課題を記す．

2. 準　　備

本稿では，論理プログラムの一般的な記法に従う [12].

関数記号の集合 Σ と**述語記号**の集合 Δ の組 (Σ,Δ) を**シグネチャ**と呼ぶ．関数記号 $f \in \Sigma \cup \Delta$ に対して，f の引数の個数を $arity(f)$ と記す．変数の可算無限集合を $\mathcal{V}$ とする．本稿では，先頭が小文字の文字列を関数・述語記号，大文字の文字列を変数とする．Σ と $\mathcal{V}$ から生成されるすべての（**有限**）**項**の集合を $\mathcal{T}(\Sigma,\mathcal{V})$ と記す．変数を持たない**基礎項**の集合 $\mathcal{T}(\Sigma,\emptyset)$ は $\mathcal{T}(\Sigma)$ と略記する．(Σ,Δ) 上の**アトム**は $p(t_1,\ldots,t_n)$ で表され，$p \in \Delta$（ただし，$arity(p)=n$），$t_1,\ldots,t_n \in \mathcal{T}(\Sigma,\mathcal{V})$ を満たす．アトムの集合を $\mathcal{A}(\Sigma,\Delta,\mathcal{V})$ と記す．

代入 σ の**定義域**，**値域**は，それぞれ $\mathcal{D}om(\sigma)$, $\mathcal{R}an(\sigma)$ で表す．値域に現れる変数の集合を $\mathcal{VR}an(\sigma)$ で表す．$\mathcal{D}om(\sigma)=\{x_1,\cdots,x_n\}$ であり，かつ $\sigma(x_i)=u_i$ のとき，σ を $\{x_1 \mapsto u_1,\cdots,x_n \mapsto u_n\}$ と記す．代入 σ の項 t への適用 $\sigma(t)$ を $t\sigma$ と記す．項 $t\sigma$ を t の**具体項**と呼ぶ．項 s,t が互いの具体項であるとき，s と t は**名前替え**であるといい，$s \sim t$ と記す．$\mathcal{V}$ から $\mathcal{V}$ への単射的な代入を**名前替え代入**と呼ぶ．すべての $x \in \mathcal{D}om(\sigma)$ について $\sigma(x) \in \mathcal{T}(\Sigma)$ のとき，σ を**基礎代入**と呼ぶ．代入 σ,θ の合成 $\sigma\theta$ は $t\sigma\theta=\theta(\sigma(t))$ と定義される．$\sigma\delta=\theta$ となる代入 δ が存在するとき，$\sigma \lesssim \theta$ と記す．$\sigma \lesssim \theta$ かつ $\sigma \gtrsim \theta$ のとき，$\sigma \sim \theta$ と記す．σ の定義域を $\mathcal{X} \subseteqq \mathcal{V}$ に制限した代入 $\sigma|_{\mathcal{X}}$ は $\{x \mapsto x\sigma \mid x \in \mathcal{D}om(\sigma) \cap \mathcal{X}\}$ である．本稿では，$\mathcal{R}an(\sigma) \subseteqq \mathcal{T}(\Sigma,\mathcal{V})$ とする．

項 s と t の**単一化子** σ は，$s\sigma \equiv t\sigma$ を満たす代入である．このとき，s と t は**単一化可能**であるという．項 s と t の**最汎単一化子** σ は，s と t の単一化子であり，s と t の任意の単一化子 θ に対して $\sigma \lesssim \theta$ を満たす．最汎単一化子は $\sim$ を法として唯一である．項 s と t の最汎単一化子を $\texttt{mgu}(s,t)$ と記す．

代入や名前替え，単一化はアトムへも自然に拡張する．

節 (clause) または**規則**は $H :- B_1,\ldots,B_k$ という形で表記され，$k \geqq 0$，$H,B_i \in \mathcal{A}(\Sigma,\Delta,\mathcal{V})$ である．H を節の**頭部** (head)，アトムの列 $B_1,\ldots,B_k$ を節の**本体** (body) と呼び，節の有限集合 $\mathcal{P}$ を**論理プログラム**とする．本体を持たない節（$k=0$ のとき）を**事実** (fact) と呼ぶ．頭部を持たない節は**クエリ** (query)（副目標の列）と呼ぶ．クエリは “:-” を省略して，節の本体と区別しやすいように [ ] を用いてリストのように $[B_1,\ldots,B_k]$ と表記し，これが空列である場合は ϵ と表記する．計算したいアトムを**質問**と呼ぶ．質問 A を $[A]$ というクエリとして扱い，クエリと区別しないで用いる場合もある．

$[A_1,\ldots,A_m]$ をクエリ，H :- B_1,...,B_k を節 c とする．A_1 と H が単一化可能で，その最汎単一化子を θ としたとき，クエリ $[B_1\theta,\ldots,B_k\theta,A_2\theta,\ldots,A_m\theta]$ を θ による**解法** (resolvent) といい，$[A_1,\ldots,A_m] \vdash_{c,\theta} [B_1\theta,\ldots,B_k\theta,A_2\theta,\ldots,A_m\theta]$ と表記する．論理プログラム $\mathcal{P}$ によるクエリ Q からの**導出**とは，

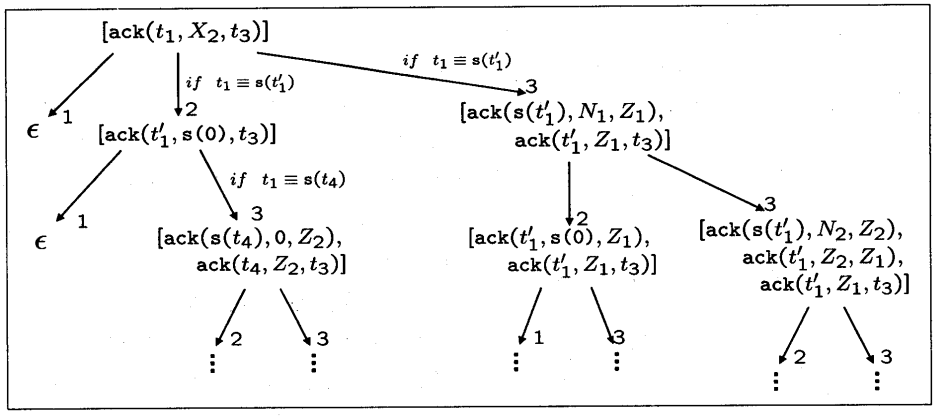

図 1　$\mathcal{P}_1$ による ack(t_1, X_2, t_3) からのすべての導出を表現した木

$Q_0 = Q$ とし，すべての i について $Q_i \vdash_{c_{i+1},\theta_{i+1}} Q_{i+1}$ としたときのクエリの（有限あるいは無限の）系列 $Q_0, Q_1, \ldots$ をいう．ただし，$\mathcal{P}$ の節 c_{i+1} に出現する変数はそれまでに使われていない変数に名前替えされているものとする．有限の導出 $Q_0, \ldots, Q_n$ を $Q_0 \vdash^n_{\mathcal{P},\theta_1 \ldots \theta_n} Q_n$ または $Q_0 \vdash^n_{\mathcal{P}} Q_n$ と表記する．$n \geq 1, n \geq 0$ のときは，$\vdash^n_{\mathcal{P}}$ をそれぞれ $\vdash^+_{\mathcal{P}}, \vdash^*_{\mathcal{P}}$ と表記する．

$p \in \Delta$，$arity(p) = n$，$\alpha_1, \ldots, \alpha_n \in \{i,o\}$ （ただし，i,o $\notin \Sigma \cup \delta \cup \mathcal{V}$）としたとき，$p(\alpha_1, \ldots, \alpha_n)$ を**質問パターン**という．これは，i の位置には入力として具体的な値，つまり，基礎項を，o の位置には出力として値を求めるために変数を含む項を与えることを意味する(注1)．$p(\alpha_1, \ldots, \alpha_n)$ は質問の集合ともみなせる：$p(\alpha_1, \ldots, \alpha_n) = \{\, p(t_1, \ldots, t_n) \mid (\alpha_i = \mathtt{i} \wedge t_i \in \mathcal{T}(\Sigma)) \vee (\alpha_i = \mathtt{o} \wedge t_i \in \mathcal{T}(\Sigma, \mathcal{V})) \,\}$．

論理プログラム $\mathcal{P}$ によるクエリ Q からのすべての導出が有限であるとき，Q が $\mathcal{P}$ について**停止する**といい，それ以外のとき，Q が $\mathcal{P}$ について**停止しない**という．質問パターン P が停止性を持たない質問を含むとき，$\mathcal{P}$ は P について**停止性を持たない**（**非停止性**を持つ）という．

3. ループに起因する非停止性

本節では，論理プログラムの導出に対するループの概念を与え，ループを検出することで非停止性を証明できることを示す．

具体的な例として，次のアッカーマン関数を表現した論理プログラム $\mathcal{P}_1$ について考える．

```
1: ack(0,N,s(N)).
2: ack(s(M),0,V) :- ack(M,s(0),V).
3: ack(s(M),s(N),V) :- ack(s(M),N,Z), ack(M,Z,V).
```

アッカーマン関数を $Ack(m,n)$ で表すと，ack は ack($m, n, Ack(m,n)$) を満たすときに真になる述語である．$\mathcal{P}_1$ は質問パターン ack(i,i,o) を意識して作られているので，$\mathcal{P}_1$ は ack(i,i,o) については停止性を持つ．しかし，ack(i,o,i) については停止性を持たない．例えば，ack(s(0), X, s(s(s(0)))) からの導出には次のような無限な導出が存在する．

(注1)：変数を含まなくてもよい．

ack(s(0), X, s(s(s(0))))]
$\vdash_{\mathcal{P}_1}$ [ack(s(0), N_1, Z_1), ack(0, Z_1, s(s(s(0))))]
$\vdash_{\mathcal{P}_1}$ [ack(s(0), N_2, Z_2), ack(0, Z_2, Z_1), ack(0, Z_1, s(s(s(0))))]
$\vdash_{\mathcal{P}_1} \cdots$

このとき，2,3 番目のクエリの先頭を比較すると，名前替えの関係にある．第一引数は全く同じ基礎項であり，第二，第三引数は共に変数であるので，ack(s(0), N_2, Z_2) からの計算は ack(s(0), N_1, Z_1) からの計算と同じになると考えられる．実際，ack(s(0), N_2, Z_2) からは ack(s(0), N_3, Z_3) が計算される．つまり，同じ計算を無限に繰り返している．このような導出の繰り返しがループである．このループを項書換えのループの概念 [1] をもとに，次のように定義する．

定義 1　$\mathcal{P}$ を論理プログラム，$[A_1, \ldots, A_l] \vdash^+_{\mathcal{P}} [B_1, \ldots, B_m]$ を導出とする．$[A_1] \vdash^+_{\mathcal{P},\theta} [B_1, \ldots, B_n]$ $(n \leq m)$ かつ $A_1 \sim B_1$ ならば，導出 $[A_1, \ldots, A_l] \vdash^+_{\mathcal{P}} [B_1, \ldots, B_m]$ を**ループ**と呼ぶ．□

このループに関して次の性質が明らかに成り立つ．

定理 1　論理プログラムを $\mathcal{P}$，クエリを Q とする．$\mathcal{P}$ による Q からの導出にループが存在するならば，$\mathcal{P}$ は Q について停止性を持たない．□

質問パターン P に Q が含まれているとき，$\mathcal{P}$ は P について停止性を持たない．よって，$\mathcal{P}$ について停止性をもたない Q を発見することで P の非停止性を証明できる．

4. ループ検出のアイディア

3 節の ack(s(0), X, s(s(s(0)))) のように，導出にループを含む質問を発見することで，質問パターン ack(i,o,i) の非停止性を証明できる．以降では，ループを生じさせる質問をどのようにして発見するかを説明する．初めに，質問パターン ack(i,o,i) に含まれる質問を一般的な表現 ack(t_1, X_2, t_3) で考える．第一，第三引数は i なので，それぞれ基礎項 t_1, t_3 で表している．第二引数は o なので，変数にしている．

次に，$\mathcal{P}_1$ による質問 ack(t_1, X_2, t_3) からのすべての導出を考えてみる．$\mathcal{P}_1$ のどの規則をどの順番で適用するかによって，導出は複数存在する（図 1）．ここで，$\mathcal{P}_1$ の 3 番目の規則を適用する場合に注目する．$t_1 \equiv$ s(t'_1) のとき，質問 ack(t_1, X_2, t_3) と規則の頭部の名前替え ack(s(M_1), s(N_1), V_1) との最汎単一

化子を考えると { $M_1 \mapsto t_1'$, $X_2 \mapsto \mathtt{s}(N_1)$,$V_1 \mapsto t_3$ } が得られる．その結果，クエリ [$\mathtt{ack}(\mathtt{s}(t_1'), N_1, Z_1)$, $\mathtt{ack}(t_1', Z_1, t_3)$] が得られる．ただし，$t_1'$ は基礎項である．続けて同様に導出すると，次のような系列が得られる．

$[\mathtt{ack}(t_1, X_2, t_3)]$
$\vdash_{\mathcal{P}_1} [\mathtt{ack}(\mathtt{s}(t_1'), N_1, Z_1), \mathtt{ack}(t_1', Z_1, t_3)]$
$\vdash_{\mathcal{P}_1} [\mathtt{ack}(\mathtt{s}(t_1'), N_2, Z_2), \mathtt{ack}(t_1', Z_2, Z_1), \mathtt{ack}(t_1', Z_1, t_3)]$
$\vdash_{\mathcal{P}_1} \cdots$

このとき，[$\mathtt{ack}(\mathtt{s}(t_1'),N_1,Z_1)$,$\mathtt{ack}(t_1',Z_2,Z_1)$] からの導出の過程に [$\mathtt{ack}(\mathtt{s}(t_1'),N_2,Z_2)$,$\mathtt{ack}(t_1',Z_2,Z_1)$, $\mathtt{ack}(t_1',Z_1,t_3)$] が出現している．よって，$\mathtt{ack}(t_1, X_2, t_3)$ からの導出にループが検出された．以上から，$\mathcal{P}_1$ は `ack(i,o,i)` において停止性を持たないと判断できる

図1の t_i や t_i' を変数とみなすと，図1は質問からのすべての導出を木で表現している．$t_1 \equiv \mathtt{s}(t_1')$ の場合でループを発見する際には，クエリの t_1 を $\mathtt{s}(t_1')$ で置き換えた状況で，導出の中でループを発見する．このように，質問パターン $p(\alpha_1, \ldots, \alpha_n)$ に対し質問 $p(X_1, \ldots, X_n)$ からの導出を考えればいいように考えられる．しかし，図1の t_i や t_i' は導出では変数のように振舞っているが，実際は任意の基礎項を表すものであるので，変数と全く同じように扱ってはならない．全く同じように扱った場合には，停止性を持つ質問パターンの導出においてもループが誤って検出されてしまうことがある．この問題を避けるために，以下のことが必要である．

(1) クエリからクエリへの導出がループかどうかの判定の際に，基礎項を表している変数は基礎項とみなす．

(2) 導出の際に，変数が基礎項を表していることを，その変数に代入される項に含まれる変数に引き継がせる．

実際，上記の (1)，(2) がなければ，停止性のある質問パターン `ack(i,i,o)` でもループを検出してしまう．

5. 質問パターンからの導出からのループ検出

4節のアイディアを実現するために，変数を2種類に区別する．本節では，その区別に基づいて，質問パターンを代表する質問の構成法を示す．次に，その区別を維持する代入を用いた導出でのループの概念を与える．最後に，そのようなループが質問パターンに対する非停止性を導くことを示す．

5.1 変数の区別

任意の基礎項を表すために導入される変数を区別するために，変数を2種類に分ける．基礎項を表すために導入する変数の集合を $\mathcal{V}^{\mathtt{i}}$ ($\subseteqq \mathcal{V}$) とする．$\mathcal{V}^{\mathtt{i}}$ の変数を $X^{\mathtt{i}}$ のように表記し，その変数は **i の情報を持つ**という．

5.2 質問パターンを代表する質問

次に，与えられた質問パターンから，それを代表する質問をどのようなアトムで表現するかを定める．質問パターン P ($= p(\alpha_1, \ldots, \alpha_n)$) に対して，質問 A ($= p(t_1, \ldots, t_n)$) が $P = \{A\theta \mid t_i\theta \in \mathcal{T}(\Sigma) \wedge \alpha_i = \mathtt{i}\}$ を満たすとき，Q は P **を代表する**という．すなわち，A は P のあらゆる質問を作り出せるような質問である．

本手法では，質問パターンの o の位置には $(\mathcal{V} \setminus \mathcal{V}^{\mathtt{i}})$ の変数を割り当て，i の位置には i の情報を持つ変数を割り当てることによって，質問パターンを代表する質問を構成する．ただし，引数に割り当てる変数はすべて異なる変数であるとする．例えば，$\mathtt{ack}(X_1^{\mathtt{i}}, X_2, X_3^{\mathtt{i}})$ は質問パターン `ack(i,o,i)` を代表する質問の一つである．このように構成した質問が質問パターンを代表していることは明らかである．

5.3 変数の区別を維持するための代入

4節で述べたように，i の情報を持つ変数は，導出の際にその情報を引き継がなくてはならない．すなわち，i の情報を持つ変数に i の情報を持たない変数を含む項を代入してはならない．i の情報は起こりえないループの検出を防ぐために必要であるが，導出の際に用いるすべての最汎単一化子が正確にこの情報を引き継ぐとは限らない．そこで，本手法では $\mathcal{V}^{\mathtt{i}}$ の変数への代入の値域は $\mathcal{T}(\Sigma, \mathcal{V}^{\mathtt{i}})$ に含まれる項のみとする．よって，ここでは $\mathtt{ack}(X_1^{\mathtt{i}}, X_2, X_3^{\mathtt{i}})$ と $\mathtt{ack}(\mathtt{s}(M_1), \mathtt{s}(N_1), V_1)$ の最汎単一化子として { $X_1^{\mathtt{i}} \mapsto \mathtt{s}(M_1), X_2 \mapsto \mathtt{s}(N_1), V_1 \mapsto X_3^{\mathtt{i}}$ } などは用いずに，{ $X_1^{\mathtt{i}} \mapsto \mathtt{s}(M_1^{\mathtt{i}}), X_2 \mapsto \mathtt{s}(N_1), V_1 \mapsto X_3^{\mathtt{i}}, M_1 \mapsto M_1^{\mathtt{i}}$ } を用いる．これにより，4節で述べた問題の1つは解決される．本節では，導出の際，i の情報の引継ぎを維持するように代入に制限を与える．

導出の際に変数の区別を保つために，次のような代入のみを扱う．

定義 2 σ を代入とする．すべての $x \in \mathcal{D}om(\sigma)$ について，$x \in \mathcal{V}^{\mathtt{i}}$ ならば $\mathcal{V}ar(x\sigma) \subseteqq \mathcal{V}^{\mathtt{i}}$ を満たすとき，σ を **I/O 代入**という．I/O 代入である単一化子を **I/O 単一化子**という． □

例 1 次の代入 $\sigma_1, \sigma_2, \sigma_3$ はアトム $p(X_1, s(X_2))$ と $p(Y_1, Y_2^{\mathtt{i}}))$ の単一化子である．

$\sigma_1 = \{X_1 \mapsto Y_1, Y_2^{\mathtt{i}} \mapsto s(X_2)\}$,
$\sigma_2 = \{X_1 \mapsto Y_1, Y_2^{\mathtt{i}} \mapsto s(X_2^{\mathtt{i}}), X_2 \mapsto X_2^{\mathtt{i}}\}$,
$\sigma_3 = \{X_1 \mapsto Y_1^{\mathtt{i}}, Y_1 \mapsto Y_1^{\mathtt{i}}, Y_2^{\mathtt{i}} \mapsto s(X_2^{\mathtt{i}}), X_2 \mapsto X_2^{\mathtt{i}}\}$

σ_1 は I/O 代入ではないが，σ_2, σ_3 は I/O 代入である． □

σ_3 も I/O 代入ではあるが，通常の変数を不必要に i の情報を持つ変数に置き換えている．i の情報を持たせても，停止性がある場合にループを検出する問題は起こらない．しかし，これでは検出できるはずのループが検出できなくなる可能性があるで，このような代入は扱わないことにする．そのために，I/O 単一化子にも最汎性を導入する．

定義 3 アトム s と t の I/O 単一化子を σ とする．s と t の任意の I/O 単一化子 θ に対して $\sigma\delta = \theta$ となる I/O 代入 δ が存在するとき，σ を **最汎 I/O 単一化子**という． □

最汎 I/O 単一化子は最汎単一化子と同様の性質を持ち，I/O 名前替えを法とし一意に決定できる．

i の情報を持つ変数に代入される項は，i の情報を持つ変数しか含まないことを I/O 代入は意味している．よって，導出の際に用いる最汎単一化子が最汎 I/O 単一化子であるならば，変数が持つ情報を適切に引き継ぐことができる．

例 2 以下は，$\mathcal{P}_1$ の3番目の規則による質問 $\mathtt{ack}(X_1^{\mathtt{i}}, X_2, X_3^{\mathtt{i}})$ からの導出である．ただし，導出で用いる最汎単一化子は最汎

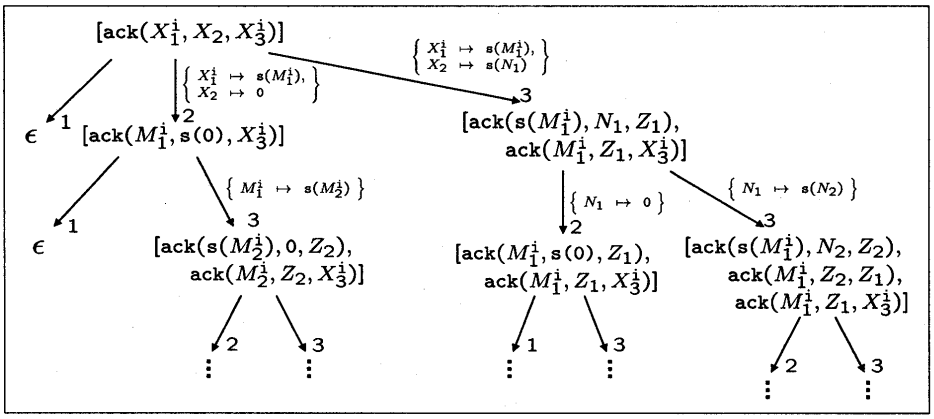

図 2　$\mathcal{P}_1$ による質問 $\mathtt{ack}(X_1^{\mathtt{i}}, X_2, X_3^{\mathtt{i}})$ の導出木

I/O 単一化子であるとする．

$[\mathtt{ack}(X_1^{\mathtt{i}}, X_2, X_3^{\mathtt{i}})]$
$\vdash_{\mathcal{P}_1} [\mathtt{ack}(\mathtt{s}(M_1^{\mathtt{i}}), N_1, Z_1), \mathtt{ack}(M_1^{\mathtt{i}}, Z_1, X_3^{\mathtt{i}})]$
$\vdash_{\mathcal{P}_1} [\mathtt{ack}(\mathtt{s}(M_1^{\mathtt{i}}), N_2, Z_2), \mathtt{ack}(M_1^{\mathtt{i}}, Z_2, Z_1), \mathtt{ack}(M_1^{\mathtt{i}}, Z_1, X_3^{\mathtt{i}})]$
$\vdash_{\mathcal{P}_1} \cdots$ □

以降では，導出で用いる最汎単一化子はすべて最汎 I/O 単一化子であるとする．

5.4 I/O ループ

本手法では，導出の中にループが存在するかどうかを判定することにより非停止性を示したい．論理プログラム $\mathcal{P}$ が質問パターン P について停止性を持たないことを示すためには，P を代表する質問を A とすると，$\mathcal{P}$ による A からの導出の中にループが存在するかを調べればよい．そのループかどうかの判定の際に $\mathtt{i}$ の情報を持つ変数は基礎項を表現しているので，変数とはみなしてはならない．また，導出の際の代入は $\mathtt{i}$ の情報を持つ変数に関してはクエリに反映させなければならない．そこで，$\mathcal{V}^{\mathtt{i}}$ を含むクエリの導出に関するループを定義する．アトム A, B が変数集合 $\mathcal{X}$ に関して名前替えであるとき，すなわち，$\mathcal{D}om(\delta) \subseteq \mathcal{X}$ かつ $\mathcal{VR}an(\delta) \cap \mathcal{V}ar(A, B) = \emptyset$ を満たす名前替え代入 δ が存在するとき，$A \sim^{\mathcal{X}} B$ と記す．これは，$\mathcal{X}$ の変数以外は基礎項とみなした名前替えであることを意味する．

定義 4　$\mathcal{P}$ を論理プログラム，$[A_1,\ldots,A_l] \vdash^{+}_{\mathcal{P}} [B_1,\ldots,B_m]$ を導出とする．$[A_1] \vdash^{+}_{\mathcal{P},\theta} [B_1,\ldots,B_n]$ $(n \leq m)$ かつ $A_1(\theta|_{\mathcal{V}^{\mathtt{i}}}) \sim^{\mathcal{V}\setminus\mathcal{V}^{\mathtt{i}}} B_1$ ならば，導出 $[A_1,\ldots,A_l] \vdash^{+}_{\mathcal{P}} [B_1,\ldots,B_m]$ を **I/O ループ**と呼ぶ．□

I/O ループに関して次の性質が成り立つ．

定理 2　論理プログラムを $\mathcal{P}$，質問パターンを P，P を代表する質問を Q とする．$\mathcal{P}$ による Q からの導出に I/O ループが存在するならば，$\mathcal{P}$ による $A \in P$ からの導出にループが存在する，すなわち，$\mathcal{P}$ は P について停止性を持たない．

［略証］　$(Q \vdash^{*}_{\mathcal{P},\theta}) [A_1,\ldots,A_m] \vdash^{+}_{\mathcal{P},\theta'} [B_1,\ldots,B_n,A_2\theta',\ldots,A_m\theta']$ を $\mathcal{P}$ による Q からの導出の I/O ループとすると，$A_1(\theta'|_{\mathcal{V}^{\mathtt{i}}}) \sim^{\mathcal{V}\setminus\mathcal{V}^{\mathtt{i}}} B_1$ を満たす．このとき，$\mathcal{D}om(\delta) = \mathcal{V}ar(Q(\theta|_{\mathcal{V}^{\mathtt{i}}})(\theta'|_{\mathcal{V}^{\mathtt{i}}})) \cap \mathcal{V}^{\mathtt{i}}$ を満たす基礎代入 δ を考える．θ,θ' は I/O 代入であるので，$Q(\theta|_{\mathcal{V}^{\mathtt{i}}})(\theta'|_{\mathcal{V}^{\mathtt{i}}})\delta \in P$ である．導出と代入には次の性質が成り立つ：$Q_1 \vdash^{*}_{\mathcal{P},\sigma} Q_1'$ とする．

- $\sigma = \sigma_1 \cup \sigma_2$ かつ $\mathcal{VR}an(\sigma_1) \cap \mathcal{D}om(\sigma_2) = \emptyset$ ならば，$Q_1\sigma_1 \vdash^{*}_{\mathcal{P},\sigma_2} Q_1'$.
- $\mathcal{D}om(\delta) \cap \mathcal{D}om(\sigma) = \emptyset$ かつ $\mathcal{VR}an(\delta) \cap \mathcal{D}om(\sigma) = \emptyset$

を満たす代入 δ について，$Q_1\delta \vdash^{*}_{\mathcal{P},\sigma} Q_1'\delta$.

この性質と，θ,θ' が I/O 代入であること，δ が基礎代入であることより，$A_1(\theta'|_{\mathcal{V}\setminus\mathcal{V}^{\mathtt{i}}})\delta \sim B_1\delta$ であり，次の導出が存在する：$Q(\theta|_{\mathcal{V}^{\mathtt{i}}})(\theta'|_{\mathcal{V}^{\mathtt{i}}})\delta \vdash^{*}_{\mathcal{P},\theta|_{\mathcal{V}\setminus\mathcal{V}^{\mathtt{i}}}} [A_1(\theta'|_{\mathcal{V}^{\mathtt{i}}})\delta,\ldots,A_m(\theta'|_{\mathcal{V}^{\mathtt{i}}})\delta] \vdash^{+}_{\mathcal{P},\theta'|_{\mathcal{V}\setminus\mathcal{V}^{\mathtt{i}}}} [B_1\delta,\ldots,B_n\delta, A_2\theta'\delta,\ldots,A_m\theta'\delta] = [B_1\delta,\ldots,B_n\delta, A_2(\theta'|_{\mathcal{V}^{\mathtt{i}}})\delta(\theta'|_{\mathcal{V}\setminus\mathcal{V}^{\mathtt{i}}}),\ldots,A_m(\theta'|_{\mathcal{V}^{\mathtt{i}}})\delta(\theta'|_{\mathcal{V}\setminus\mathcal{V}^{\mathtt{i}}})]$．よって，$\mathcal{P}$ による $Q(\theta|_{\mathcal{V}^{\mathtt{i}}})(\theta'|_{\mathcal{V}^{\mathtt{i}}})\delta$ からの導出にループが存在する．□

本手法では，質問パターンを代表する質問からのあらゆる導出を木で表現し（図 2），その木の中から I/O ループを検出することで非停止性を証明する．

例 3　図 2 より，$\mathcal{P}_1$ による質問 $\mathtt{ack}(X_1^{\mathtt{i}}, X_2, X_3^{\mathtt{i}})$ からの導出には次の導出が存在する．$[\mathtt{ack}(\mathtt{s}(M_1^{\mathtt{i}}), N_1, Z_1), \mathtt{ack}(M_1^{\mathtt{i}}, Z_1, X_3^{\mathtt{i}})] \vdash_{\mathcal{P}_1} [\mathtt{ack}(\mathtt{s}(M_1^{\mathtt{i}}), N_2, Z_2), \mathtt{ack}(M_1^{\mathtt{i}}, Z_2, Z_1), \mathtt{ack}(M_1^{\mathtt{i}}, Z_1, X_3^{\mathtt{i}})]$．$[\mathtt{ack}(\mathtt{s}(M_1^{\mathtt{i}}), N_1, Z_1)] \vdash_{\mathcal{P}_1} [\mathtt{ack}(\mathtt{s}(M_1^{\mathtt{i}}), N_2, Z_2), \mathtt{ack}(M_1^{\mathtt{i}}, Z_2, Z_1)]$ かつ $\mathtt{ack}(\mathtt{s}(M_1^{\mathtt{i}}), N_1, Z_1) \sim^{\mathcal{V}\setminus\mathcal{V}^{\mathtt{i}}} \mathtt{ack}(\mathtt{s}(M_1^{\mathtt{i}}), N_2, Z_2)$ であるので，I/O ループを持つ．よって，$\mathcal{P}_1$ は $\mathtt{ack(i,o,i)}$ について停止性を持たない．□

6. 本手法の実現と既存の研究との比較

本稿で提案した手法を Standard ML of New Jersey で実装した．導出木は一般には無限になるので，導出木を構成しながら，深さ優先探索で I/O ループを探すことにより実現している．有限の範囲で探索を打ち切るために，本実装ではユーザが探索する導出木の高さを指定する．入力は，Prolog の構文に従ったプログラムと質問パターンとし，ループを検出したときは，“停止性なし”，それ以外は“不明”を出力する．本実装では，プリミティブな述語やカット演算子を含むプログラムは対象としていない（NIT [10] と同様の制約である）．この場合も“不明”を出力する．

実装したツールを用いて Termination Competition 2007 [5] (注2) の論理プログラム部門（LP category）で扱っているプログラム（TPDB 4.0 ; The Termination Problems Data Base 4.0）

(注2)：http://www.lri.fr/ marche/termination-competition/2007/

表 1　TPDB 4.0 の例に対する各ツールの実験結果（325 個）

ツール / 応答	AProVE	Polytool	TALP	NTI	本手法
停止性あり	248	210	170	0	0
停止性なし	0	0	0	42	52
不明	77	115	155	283	273

を対象とした実験を行った．Termination Competition 2007 の結果に本手法を加えた結果を表 1 に示す．TPDB 4.0 のサンプルプログラム数は 325 個である．Termination Competition 2007 の論理プログラム部門に参加しているツールの内，非停止性を証明しているのは NTI のみである．本手法と NTI の両方で成功した例は 40 個で，本手法のみが成功した例は 12 個，NTI のみが成功した例は 2 個存在した．

次に，証明にかかる時間について報告する．NTI は与えられたプログラムを静的に解析することにより非停止性を示しており，Termination Competition 2007 での結果では，平均すると 0.01 秒以内に出力を返している．これに対し，本手法は導出を求めながらループを検出する手法である．早い段階でループが検出できれば速いが，そうでない場合，プログラム中の本体を持つ節の数に比例して指数的に計算量が増大する．以下に，本手法で TPDB の非停止性を検証するのにかかる時間を示す．ただし，探索する木の高さは 10 とした．利用した計算機の仕様は，CPU は Intel Pentium III-S 1.26GHz Dual，メモリは 2GB，OS は FreeBSD 4.8-RELEASE である．証明に成功する 52 個の平均時間は 0.074 秒，証明できない 273 個の平均時間は 0.191 秒であった．成功する場合は途中で探索を終えるため，証明できない場合より早く結果が得られた．今回，実験の対象とした TPDB では木の高さが 10 程度で十分だったため，ほとんどのプログラムでは 1 秒以内に結果が得られた．しかし，節の数が多いプログラムは，そうでないプログラムに比べると相当な時間が必要だった．これに対して，NTI を同計算機で動作させると，どのプログラムでも 0.01 秒以下で結果が得られた．理論的な比較は行っていないが，NTI の方が本手法より計算時間は少ないと考えられる．

また，1 節で挙げた `mul` のプログラムが `mul(o,o,i)` について停止しないことを NTI では証明できないが，本手法では証明できた．

7. ま と め

本稿では，論理プログラムの非停止性証明法を提案し，この手法の正当性を示した．また，実験により，Termination Competition 2007 の論理プログラム部門で与えられているサンプルプログラムに対しては，本手法の方が既存の手法より多くのプログラムの非停止性を示すことに成功した．本稿では，深さ優先探索で実装したが，幅優先探索で実装し比較することは今後の課題である．

論理プログラムにはカット演算という特殊な制御が存在する．本手法では，カット演算がない論理プログラムを対象としているが，カット演算を持つ論理プログラムを対象とした手法へ簡単に拡張できると考えている．それは，本手法は与えられたプログラム自身を解析するわけではなく，与えられたプログラムによる導出を解析することにより非停止性を示しているため，カット演算に簡単に対応できるからである．具体的には，導出木の探索範囲を工夫することで実現できると考えている．

本手法は TRS のループ検出による非停止性の証明手法 [1] [2] と同様な考えに基づいている．しかし，変数を 2 種類に区別して節のまま扱うことで，論理プログラムでの非停止性の問題を直接的にわかりやすく捉えた．本手法でのループの定義では，名前替えを用いている．しかし，準単一化を利用したループの検出の手法 [3] を用いることで，よりループの判定が強力になると考えている．

Prolog などでは，節には優先順位があるとし，先に書かれる節を優先している．Prolog では少なくとも 1 つ解が求まれば十分な場合もある．しかし，このような停止性の証明は非常に難しい．

謝辞　本研究は一部，科研費#16650005，#17700009，#18500011 の補助を受けている．

文　献

[1] N. Dershowits. Termination of rewriting. *Journal of Symbolic Computation*, Vol. 3, No. 1/2, pp. 69–115, 1987.

[2] A. Geser and H. Zantema. Non-looping string rewriting. *Theoretical Informatic and Applications*, Vol. 33, pp. 279–302, 1999.

[3] J. Giesl, R. Thiemann, and P. Schneider-Kamp. Proving and disproving termination of higher-order functions. Vol. 3717 of LNCS, pp. 216–231, 2005.

[4] J.-M. Hullot. Canonical forms and unification. *Proceedings of the 5th International Conference on Automated Deduction*, Vol. 87 of LNCS, pp. 318–334, 1980.

[5] C. Marche, J. Waldmann and H. Zantema. Termination Competition 2007. *In Proceedings of the 9th International Workshop on Termination*, pp. 72–76, 2007.

[6] M. T. Nguyen and D. De Schreye. Polytool: Proving Termination Automatically Based on Polynomial Interpretations. *In Proceedings of the 16th International Symposium on Logic-Based Program Synthesis and Transformation*, Vol. 4407 of LNCS, pp. 210–218, 2007.

[7] N. Nishida, M. Sakai, and T. Sakabe. Partial inversion of constructor term rewriting systems. *In Proceedings of the 16th International Conference on Rewriting Techniques and Applications*, Vol. 3467 of LNCS, pp. 264–278, 2005.

[8] E. Ohlebusch. *Advanced Topic in Term Rewiting*. Springer, 2002.

[9] E. Ohlebusch, C. Claves and C. Marche. TALP: A Tool for the Termination Analysis of Logic Programs. *In the 11th International Conference on Rewriting Techniques and Applications*, Vol. 1833 of LNCS, pp. 270–273, 2000.

[10] E. Payet and F. Mesnard. Nontermination inference of logic programs. *ACM Transactions on Programming Languages and Systems*, Vol. 28, No. 2, pp. 256–289, 2006.

[11] P. Schneider-Kamp, J. Giesl, A. Serebrenik and R. Thiemann. Automated Termination Analysis for Logic Programs by Term Rewriting. *In Proceedings of the 16th International Symposium on Logic-Based Program Synthesis and Transformation*, Vol. 4407 of LNCS, pp. 177–193, 2007.

[12] J.W. ロイド ／ 佐藤雅彦，森下真一　訳．論理プログラミングの基礎．産業図書，1987.

[13] 西田直樹，酒井正彦，坂部俊樹．構成子項書換え系の逆計算プログラムの生成．電子情報通信学会論文誌 D–I，Vol. J88–D–I，No. 8，pp. 1171–1183，2005.