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

信学技報
TECHNICAL REPORT OF IEICE.

手続き型プログラムから書換え系への変換に基づく ソフトウェア検証の試み

古市　祐樹[†]　　西田　直樹[††]　　酒井　正彦[††]　　草刈　圭一朗[††]　　坂部　俊樹[††]

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

あらまし　項書換えの分野では帰納的定理の証明手法として潜在帰納法や書換え帰納法などが広く研究されている．２つの異なる関数が任意の入力に対して同様の出力を返すことは帰納的定理として捉えられるので，帰納的定理の証明法は関数型プログラムの等価性の検証に利用できる．本研究では，項書換えにおける帰納的定理の証明法を利用して手続き型プログラムの等価性の検証を試みる．具体的には，手続き型プログラムから書換え系への変換を与え，その変換により手続き型プログラムの等価性を項書換え系の関数の等価性に帰着できることを示す．
キーワード　項書換え系，帰納的定理，潜在帰納法，プレスブルガー文，完備化，プログラム変換

Approach to Software Verification Based on Transformation from Procedural Programs to Rewrite Systems

Yuki FURUICHI[†], Naoki NISHIDA[††], Masahiko SAKAI[††],

Keiichiro KUSAKARI[††], and Toshiki SAKABE[††]

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

Abstract　In the field of term rewriting, inductionless induction and rewriting induction have been widely studied as methods for proving inductive theorems. Since equivalence of functions (that is, the output values are the same for the same input) can be represented as an inductive theorem, methods for proving inductive theorems are useful to verify the equivalence of functions in functional programming. In this paper, we try to take advantage of methods for proving inductive theorems in verifying procedural programs written in a subset of the C language with integer type. More precisely, we propose a transformation from procedural programs to rewrite systems and show that the transformation reduces the equivalence of procedural programs to that of functions in rewrite systems.
Key words　term rewriting system, inductive theorem, inductionless induction, Presburger arithmetic, completion, program transformation

1. はじめに

項書換えの分野では帰納的定理の証明手法として潜在帰納法や書換え帰納法などが広く研究されている [9] [10] [11] [12]．帰納的定理とは任意の基底項上で成立する等式である．２つの異なる関数が任意の入力に対して同様の出力を返すことは帰納的定理として捉えられるので，帰納的定理の証明法は関数型プログラムで定義される関数の等価性の検証に利用できる．

一方，手続き型言語ではホーア論理を利用して関数の正当性を検証する方法がある [14]．しかし，ループ不変式や事前条件・事後条件を与えるなどのヒューリスティックな作業が必要である．

本研究では，項書換えにおける帰納的定理の証明法を利用して手続き型プログラムで定義される関数の検証を試みる．具体的には，手続き型プログラムを同等の計算を行う書換え系に変換し，帰納的定理の証明法を用いて等価性の検証を行う．検証の対称とする手続き型言語は while 文を記述できる int 型のみを扱う C 言語関数とする．この手続き型言語で書かれたプログ

ラムを検証するため，このプログラムを真偽判定可能な述語を条件にもつ決定可能条件付き項書換え系 (dTRS)，中でもプレスブルガー文を条件部にもつ項書換え系である pdTRS に変換する．pdTRS を用いる理由として，

- プレスブルガー文は真偽判定と充足可能判定が決定可能
- 完備化の暴走の抑制
- 完備化に適した負の数のコーディング

が挙げられる．実際には，直接項書換え系 (TRS) へ変換する手法も試みたが，比較演算子が完備化を暴走させたので pdTRS を導入した．変換した pdTRS と仕様で与える pdTRS の等価性を潜在帰納法で検証する．潜在帰納法のために，TRS の完備化手続きを pdTRS に拡張する．本手法では，関数型プログラムで書かれた正しい関数と新たに作成した手続き型関数の等価性の検証だけでなく，２つの手続き型関数の等価性の検証も行える．

本稿は次のように構成される．2. は書換え系に関する記法を説明する．3. では検証の対象とする手続き型言語の構文と意味論を与える．4. では検証に用いる dTRS の枠組，手続き型プログラムから pdTRS への変換アルゴリズムを提案する．また，その正当性を示す．5. では等価性の検証手順を説明し，dTRS での完備化を提案する．また，sum 関数での等価性検証例を挙げる．6. では今後の課題について議論する．

2. 準　　備

本論文では，項書換えの一般的な記法に従う [1].

抽象書換え系 (ARS) S とは，対象となる集合 A と A 上の**簡約化関係**と呼ばれる二項関係 $\rightarrow$ の組 $(A, \rightarrow)$ である．S における正規形の集合を $NF(S)$ で表す．$x \overset{*}{\rightarrow} z \overset{*}{\leftarrow} y$ となるような z が存在するとき，x と y は**会同関係にある**といい，$x \downarrow y$ と表記する．

関数記号の集合 $\mathcal{F}$，変数の可算無限集合 $\mathcal{X}$，写像 arity:$\mathcal{F} \rightarrow \mathbb{N}$ (ただし，$\mathbb{N}$ は自然数の集合) から生成されるすべての**項**の集合を $\mathcal{T}(\mathcal{F}, \mathcal{X})$ とする．項 t に現れるすべての変数の集合を $\mathcal{V}ar(t)$ で表す．項 s と t が同一であるときは $s \equiv t$ と記述する．項 t における**位置**の集合を $\mathcal{O}(t)$ とする．位置 $p, q \in \mathcal{O}(t)$ に対して，$pp' = q$ を満たす $p' \in \mathcal{O}(t)$ が存在するとき，$p \leqq q$ と書く．特に $p' \neq \varepsilon$ とき，$p < q$ と記す．項 s の位置 p にある項を t に置き換えて得られる項を $s[t]_p$ と書く．**文脈** $C[\ ]$ において位置 p に出現するホール $\square$ を項 t で置き換えることによって得られる項を $C[t]_p$ と記す．なお，p を省略してもよい．また，位置 $p \in \mathcal{O}(t)$ における部分項 u を $t|_p$ と記す．

代入 σ の定義域，値域を $\mathcal{D}om(\sigma)$，$Ran(\sigma)$ で表す．値域に現れる変数の集合を $\mathcal{VR}an(\sigma) = \bigcup_{t \in \mathcal{R}an(\sigma)} \mathcal{V}ar(t)$ とする．$\mathcal{D}om(\sigma) = \{x_1, \cdots, x_n\}$ であり，かつ $\sigma(x_i) \equiv t_i$ のとき，σ を $\{x_1 \mapsto t_1, \cdots, x_n \mapsto t_n\}$ と記す．項 t に対して，$\sigma(t)$ を t の**インスタンス**と呼び，$t\sigma$ と略記する．代入 σ, σ' について，$\mathcal{D}om(\sigma) = \mathcal{D}om(\sigma')$ かつすべての $x \in \mathcal{D}om(\sigma)$ において $\sigma(x) \equiv \sigma'(x)$ のとき，$\sigma = \sigma'$ と記述する．σ と σ' の合成は $\sigma\sigma'$ と記述し，$x\sigma\sigma' \equiv \sigma'(\sigma(x))$ で定義する．

$$
\begin{array}{rcl}
D & ::= & \texttt{int}\ fn\texttt{(}P\texttt{)\{}I\ S\ \texttt{return(}E\texttt{);\}} \\
P & ::= & \epsilon \mid \texttt{int}\ x \mid \texttt{int}\ x\texttt{,}\ P \\
I & ::= & \epsilon \mid \texttt{int}\ x\ \texttt{=}\ E\texttt{;}\ I \\
S & ::= & \epsilon \\
 & & \mid x\ \texttt{=}\ E\texttt{;}\ S \\
 & & \mid \texttt{if(}B\texttt{)\{}S\texttt{\}else\{}S\texttt{\}}\ S \\
 & & \mid \texttt{while(}B\texttt{)\{}S\texttt{\}}\ S \\
 & & \mid \texttt{for(}x\ \texttt{=}\ E\texttt{;}B\texttt{;}x\ \texttt{=}\ E\texttt{)\{}S\texttt{\}}\ S \\
E & ::= & x \mid n \mid E\ \texttt{+}\ E \mid E\ \texttt{-}\ E \mid E\ \texttt{*}\ E \mid E\ \texttt{/}\ E \mid \texttt{(}\ E\ \texttt{)} \\
E_a & ::= & x \mid n \mid E\ \texttt{+}\ E \mid E\ \texttt{-}\ E \mid \texttt{(}\ E_a\ \texttt{)} \\
B & ::= & B_E \mid B\ \texttt{||}\ B \mid B\ \texttt{\&\&}\ B \mid \texttt{!}B \\
B_E & ::= & E_a\ \texttt{==}\ E_a \mid E_a\ \texttt{!=}\ E_a \mid E_a\ \texttt{<}\ E_a \mid E_a\ \texttt{<=}\ E_a
\end{array}
$$

図 1　手続き型言語の構文

3. 手続き型言語の構文と意味論

ここでは検証の対象とする手続き型言語の仕様を定める．具体的には構文，評価意味論，計算意味論を与える．

3.1 構　　文

本稿では，以下の限定した while プログラムのクラスで検証を試みる．

（１） 制御文として if 文と while 文を書くことが出来る．
（２） 型は整数 (int)(特に，自然数のみを扱う) に限る．
（３） 関数呼び出しはない．

対象とする手続き型言語を BNF 記法を用いて図 1 のように定義する．x，fn はそれぞれ変数名，関数名を表す記号列とし，C 言語の記法に従う．n は自然数とする．大文字から始まる単語は非終端記号を表す．ϵ は空文を表す．for 文は while 文のシンタックスシュガーとして扱う．また，図 1 で定めるプログラムに容易に変換できるようなシンタックスシュガー (インクリメント演算子++など) も許す．検証の対象とするプログラムは図 1 の構文に従う C プログラムとし，正常にコンパイルができるものとする．C 言語にはブール型はないが，ブール型の概念に矛盾しないプログラムを扱うこととする．

3.2 評価意味論

次に 3.1 で与えた手続き型言語の評価意味論 $\Rightarrow_A$，$\Rightarrow_B$ を定義する [7]．ここで提案する言語の基本操作は代入文 x = e; である．よって意味論を定義する上で変数への値を保持するための記憶が必要である．記憶を m とすると，$m[x]$ は m で記憶されている x の値を表す．また，m の定義域を代入と同じように $\mathcal{D}om(m)$ で表す．

評価関係 $\Rightarrow_A$ は一つの式 E と一つの記憶 m を取り，一つの値 (自然数) を返す．$\Rightarrow_A$ の型は $\langle Exp, Store\rangle \mapsto \mathbb{N}$ であり，その意味論は自然数上の四則演算に従う [7].

評価関係 $\Rightarrow_B$ は一つの論理式 B と一つの記憶 m を取り，T(真) か F(偽) を返す．$\Rightarrow_B$ の型は $\langle BExp, Store\rangle \mapsto \{T, F\}$ であり，意味論は一般のものに従う [7].

3.3 計算意味論

最後に手続き型言語の計算意味論 $\rightarrow_C$ を与える．$\rightarrow_C$ はステートメントの系列 (ここでは変数の宣言文を含む) と一つの記憶を取り，ステートメントの系列と記憶を返す．正確に

入力：`int` f`(int` $x_1,\ldots,$`int` x_n`){`xs`}`，出力：R．ただし，$\mathbb{T}(f(x_1,\ldots,x_n),xs,\emptyset,1)=(t,R,i)$

- $\mathbb{T}(t,\epsilon,R,i)=(t,R,i)$.
- $\mathbb{T}(t,$`int` x `=` e`;`$xs,R,i)=\mathbb{T}(U_i(\overrightarrow{\mathcal{V}ar(t)},x),xs,R\cup\{t\to U_i(\overrightarrow{\mathcal{V}ar(t)},\mathbb{E}(e))\},i+1)$.
- $\mathbb{T}(t,x$ `=` e`;`$xs,R,i)=\mathbb{T}(U_i(\overrightarrow{\mathcal{V}ar(t)}),xs,R\cup\{t\to U_i(\overrightarrow{\mathcal{V}ar(t)})\{x\mapsto\mathbb{E}(e)\},i+1)$.
- $\mathbb{T}(t,$`if(`be`){`s_1`}else{`s_2`}`$xs,R,i)=\mathbb{T}(U_{i''}(\overrightarrow{\mathcal{V}ar(t)}),xs,R\cup R'\cup R''\cup\left\{\begin{array}{rcl} t & \to & U_i(\overrightarrow{\mathcal{V}ar(t)})\Leftarrow be,\\ t & \to & U_{i'}(\overrightarrow{\mathcal{V}ar(t)})\Leftarrow\neg be,\\ t' & \to & U_{i''}(\overrightarrow{\mathcal{V}ar(t)}),\\ t'' & \to & U_{i''}(\overrightarrow{\mathcal{V}ar(t)})\end{array}\right\},i'')$,

  ただし，$\mathbb{T}(U_i(\overrightarrow{\mathcal{V}ar(t)}),s_1,\emptyset,i+1)=(t',R',i')$ かつ $\mathbb{T}(U_{i'}(\overrightarrow{\mathcal{V}ar(t)}),s_2,\emptyset,i'+1)=(t'',R'',i'')$.
- $\mathbb{T}(t,$`while(`be`){`s_1`}`$xs,R,i)=\mathbb{T}(U_{i'}(\overrightarrow{\mathcal{V}ar(t)}),xs,R\cup R'\cup\left\{\begin{array}{rcl} t & \to & U_i(\overrightarrow{\mathcal{V}ar(t)})\Leftarrow be,\\ t & \to & U_{i'}(\overrightarrow{\mathcal{V}ar(t)})\Leftarrow\neg be,\\ t' & \to & t\end{array}\right\},i')$,

  ただし，$\mathbb{T}(U_i(\overrightarrow{\mathcal{V}ar(t)}),s_1,\emptyset,i+1)=(t',R',i')$.
- $\mathbb{T}(t,$`for(`x `=` e_1`;`be`;`x `=` e_2`){`s_1`}`$xs,R,i)=\mathbb{T}(U_{i'}(\overrightarrow{\mathcal{V}ar(t)}),xs,R\cup R'\cup\left\{\begin{array}{rcl} t & \to & U_i(\overrightarrow{\mathcal{V}ar(t)})\{x\mapsto\mathbb{E}(e_1)\},\\ U_i(\overrightarrow{\mathcal{V}ar(t)}) & \to & U_{i+1}(\overrightarrow{\mathcal{V}ar(t)})\Leftarrow be,\\ U_i(\overrightarrow{\mathcal{V}ar(t)}) & \to & U_{i'}(\overrightarrow{\mathcal{V}ar(t)})\Leftarrow\neg be,\\ t' & \to & U_i(\overrightarrow{\mathcal{V}ar(t)})\end{array}\right\},i')$,

  ただし，$\mathbb{T}(U_{i+1}(\overrightarrow{\mathcal{V}ar(t)}),s_1,\emptyset,i+2)=(t',R',i')$.
- $\mathbb{T}(t,$`return(`e`);`$,R,i)=(y,R\cup\{t\to U_i(\mathbb{E}(e)),U_i(y)\to y\},i+1)$，ただし，$y\notin\mathcal{V}ar(t)$.
- $\mathbb{E}(x)=x$，$\mathbb{E}(n)=S^n(0)$，$\mathbb{E}(e_1$`+`$e_2)=Add(\mathbb{E}(e_1),\mathbb{E}(e_2))$．(`-`，`*`，`/`の場合は Add をそれぞれ Sub，Mul，Div に置き換える)

図 2　変換アルゴリズム $\mathbb{TR}$

は，先頭のステートメントが実行 (評価) され，残りのステートメントの系列と，更新された記憶が返される．$\to_C$ の型は $\langle Statements,Store\rangle\mapsto\langle Statements,Store\rangle$ であり，その意味論は [7] に従う．

4. 決定可能条件付き項書換え系への変換

本節では 3. で定まる手続き型言語プログラムを検証するための関数型言語のモデルとして決定可能条件付き項書換え系 (dTRS) の枠組を与え，dTRS に属するプレスブルガー文条件付き TRS (pdTRS) への変換アルゴリズムを提案する．また，変換アルゴリズムの正当性を示す．

4.1 決定可能条件付き項書換え系

$\mathcal{F}$ 上の**決定可能条件付き書換え規則** (l,r,c) は，$l\notin\mathcal{X}$，$l,r\in\mathcal{T}(\mathcal{F},\mathcal{X})$，$\mathcal{V}ar(l)\supseteq\mathcal{V}ar(r)$ を満たす項 l,r と，真偽判定と充足可能性判定が決定可能な $\mathcal{T}(\mathcal{F},\mathcal{X})$ 上の一階論理述語 c の組であり，$l\to r\Leftarrow c$ と記す．項 l,r をそれぞれ**左辺**，**右辺**，条件 c を**条件部**と呼ぶ．c が T (真) のときは条件部を省略して $l\to r$ と書くこともある．

R を決定可能条件付き書換え規則の集合とする．R で定まる書換え関係 $\to_R$ を，$\to_R=\{(C[l\sigma]_p,C[r\sigma]_p)\mid l\to r\Leftarrow c\in R,C\in\mathcal{T}(\mathcal{F}\cup\{\Box\},\mathcal{X}),c\sigma$が真 $\}$ と定義する．**決定可能条件付き項書換え系 (dTRS)** は項の集合 $\mathcal{T}(\mathcal{F},\mathcal{X})$ と書換え関係 $\to_R$ で定められる抽象書換え系 $(\mathcal{T}(\mathcal{F},\mathcal{X}),\to_R)$ であり，書換え規則の集合 R で表す．条件部が真の規則のみである dTRS は項書換え系 (TRS) である．

本稿では，条件部がプレスブルガー文 [6] である dTRS を扱う．**プレスブルガー算術**とは加算を持つ整数の理論 (整数の集合 $\mathbb{Z}$ 上の変数，定数，$+$，$-$，$=$，$\geqq$，$\leqq$，$\vee$，$\wedge$，$\neg$，$\forall$，$\exists$からなる理論) であり，プレスブルガー算術上の閉論理式を**プレスブルガー文 (P 文)** (注1)という．P 文の真偽判定と充足可能判定は決定可能である．条件部を P 文に制限した dTRS を pdTRS と呼ぶ．本稿では限量子を持たない pdTRS を扱う．

P 文の条件部の意味論を次のように与える．c を P 文，θ を代入とする．以下の条件が成り立つとき $c\theta$ は真．

- $\mathcal{R}an(\theta)\subseteq\mathcal{T}(\{Add,Sub,S,0\},\mathcal{X})$，かつ
- P 文 $c\theta'$ が真．ただし，θ' は $\mathcal{D}om(\theta)=\mathcal{D}om(\theta')$ かつ $\forall x\in\mathcal{D}om(\theta').\ x\theta'=\mathbb{E}^{-1}(x\theta)$ を満たす代入とする ($\mathbb{E}^{-1}$ は 4.2 で与える変換 $\mathbb{E}$ の逆変換とする).

4.2 pdTRS への変換アルゴリズム

ここでは図 1 で定義された手続き型言語で書かれたプログラムを pdTRS に変換するアルゴリズムを提案する．

図 2 の変換アルゴリズム $\mathbb{TR}$ は図 1 で定義された手続き型言語で書かれたプログラムを入力とし，それを変換した pdTRS を出力する．$\mathbb{TR}$ 自体は，再帰的変換 $\mathbb{T}$ で構成される．$\mathbb{TR}$ で生成された規則は元のプログラムの制御の流れを表現している．

再帰的変換 $\mathbb{T}$ の詳細について説明する．変換 $\mathbb{T}$ はステートメントを一つ取り出して，その種類で場合分けして変換する．$\mathbb{T}(t,xs,R,i)=(t',R',i')$ とする．t は計算したい項であり，次に生成する規則の左辺となる．xs は変数宣言とステートメントの系列で，これから pdTRS へと変換するプログラムである．R はすでに得られた規則である．R' は t を計算するための変換してできた規則の (R を含む) 集合である．i はプログラムの内部状態を表す関数記号 U を識別するためにラベルとして利用する．$U_i(x_1,\ldots,x_m)$ は，ラベル i が指す (ラベル i を U に付ける際に読んでいる) 命令文を評価する際のメモリの状態を表していると言える．U_i の導入は CTRS から TRS

(注1)：自由変数は $\forall$ で束縛されているとして真偽判定を行う．

```
int sum1(int x){
  int i=0; int z=0;
  while(i<x){
    i=i+1; z=z+i;
  }
  return z;
}
```

図 3　手続き型プログラム sum1

への変換 [2] で用いる記号の考え方を参考とした．$\overrightarrow{\mathcal{V}ar(t)}$ は t に含まれる変数を特定の順序で並べた列である．プログラム $P =$ `int` f`(int` x_1`,...,int` x_n`){`xs`}` を変換したいときは，$\mathbb{T}(f(x_1,\ldots,x_n), xs, \emptyset, 1)$ とすることで $f(t_1,\ldots,t_n)$ を計算する pdTRS R' が得られる．

最後に変換 $\mathbb{E}$ について説明する．$\mathbb{E}$ は式から項への変換であり，$\mathbb{E}$ の型は $E \mapsto \mathcal{T}(\{Add, Sub, Mul, S, 0\}, \mathcal{X})$ である．式を評価する pdTRS R_c は $\{Add(x,0) \to x, Add(x, S(y)) \to S(Add(x,y)), \cdots\}$ のようにあらかじめ与えておく除算は例外処理があるので今回は用いないこととする．また，R_c は合流性と十分完全性(注2)を持つように与える．

例 4.1　図 3 のプログラム $sum1$ を $\mathbb{TR}$ で変換して得られる pdTRS は以下のようになる．

$$R_{sum1} = \left\{ \begin{array}{rcl} sum1(x) & \to & U_1(x,0) \\ U_1(x,i) & \to & U_2(x,i,0) \\ U_2(x,i,z) & \to & U_3(x,i,z) \Leftarrow i < x \\ U_3(x,i,z) & \to & U_4(x, Add(i, S(0)), z) \\ U_4(x,i,z) & \to & U_5(x,i,Add(z,i)) \\ U_5(x,i,z) & \to & U_2(x,i,z) \\ U_2(x,i,z) & \to & U_6(x,i,z) \Leftarrow i \geqq x \\ U_6(x,i,z) & \to & U_7(z) \\ U_7(y) & \to & y \end{array} \right.$$

命題 4.2　$\mathbb{TR}(P)$ は合流性を持つ左線形 pdTRS である．また，P が停止性を持つならば，$\mathbb{TR}(P) \cup R_c$ は十分完全性を持つ．

4.3　変換アルゴリズムの正当性

ここでは図 2 で提案した変換アルゴリズムの正当性を示す．ここでいう正当性とは手続き型プログラムで計算された計算とその手続き型プログラムを変換した pdTRS での計算は一致することである．

式 e を項に変換したものを $t_e(= \mathbb{E}(e))$ と書く．記憶 m に対して代入 θ_m を $\mathcal{D}om(\theta_m) = \mathcal{D}om(m)$ かつ，すべての $x \in \mathcal{D}om(\theta_m)$ に対して $x\theta_m = t_{m[x]}$ と定義する．

以下の定理は変換の正当性を保証する．

定理 4.3　プログラム P を `int` f`(int` x_1`,...,int` x_n`){`xs `return(`e`);}` (is は変数宣言の系列，xs はステートメントの系列を表す)，m，m' を記憶，$\mathbb{T}(f(x_1,\ldots,x_n), xs$ `return(`e`);`$, \emptyset, 1) = (t, R, k)$ とする．$(xs$ `return(`e`);`$, m) \xrightarrow{*}_C$ (`return(`e`);`$, m')$ かつ $(e, m') \Rightarrow_A a$ のとき，かつこのときに限り，$f(x_1,\ldots,x_n)\theta_m \xrightarrow{*}_{R \cup R_c} t_a$．

(注2)：TRS R が十分完全性を持つとは，すべての基底項 s に対して $s \xrightarrow{*}_R t$. ただし，t は構成子のみで構成される項とする．

5.　手続き型プログラムの検証手順

本節では本稿で提案する手続き型プログラムの検証手順を説明する．検証する性質は 3. で提案した手続き型言語で書かれたプログラムと pdTRS で書かれた仕様を満たす関数の等価性とする．等価性判定には潜在帰納法 [9] [10] [11] [12] を利用する．

5.1　検証対象プログラムの制限

検証の対象とするプログラムは以下を満たすと仮定する．

- プログラムは停止性を持つ．
- オーバーフローは起こらない．

5.2　提案する検証手順

検証手順は以下の順に従う．3. で提案した手続き型言語で書かれたプログラムを $P =$ `int` f`(int` x_1`,...,int` x_n`){`xs`}`，pdTRS で書かれた仕様 (関数 $f'(x_1,\ldots,x_n)$) を R_s とする．

（1）　$\mathbb{TR}(P) = R'$．$R_p = R' \cup R_c$ とする．

（2）　等式 $f(x_1,\ldots,x_n) = f'(x_1,\ldots,x_n)$ が $R_p \cup R_s$ 上で帰納的定理であるかを潜在帰納法により検証する．

なお，手続き型プログラム P_1 と P_2 の等価性についても $\mathbb{TR}$ で得られる pdTRS の U 記号の重複がないと仮定すれば，上記と同様に行える．

5.3　潜在帰納法

等式 $e = e'$ が項書換え系 R における**帰納的定理**であるとは，e と e' に対する任意の基底代入 σ_g について $e\sigma_g =_R e'\sigma_g$ となることである．

2 つの抽象書換え系を $R_1 = (A, \to_1)$，$R_2 = (A, \to_2)$ とする．$\overset{*}{\leftrightarrow}_1 = \overset{*}{\leftrightarrow}_2$ のとき R_1 と R_2 は**等価**であるという．**潜在帰納法**とは適当な集合上における 2 つの抽象書換え系の等価性を判定する手法である．

定理 5.1 (潜在帰納法)　以下の条件を満たすとき，$\overset{*}{\leftrightarrow}_1 = \overset{*}{\leftrightarrow}_2$.

- $\to_1 \subseteqq \to_2$.
- R_1 が弱正規性を持つ．
- R_2 が合流性を持つ．
- $NF(R_1) = NF(R_2)$.

R 上の等式 $e = e'$ の証明は以下の手順で行える．

（1）　$R \subseteqq \succ$ となる簡約化順序を与える．($e \succ e'$ とする)

（2）　($\{e = e'\}, R$) と $\succ$ で完備化 (5.4 参照) を行う．

（3）　完備化が成功して R' が得られたとする．$NF(R \cup \{e \to e'\}) = NF(R')$ を判定する．真ならば $e = e'$ は R の帰納的定理である．

5.4　dTRS における完備化手続き

TRS における完備化 [1] [13] を dTRS に拡張する．まずは dTRS の危険対の概念を導入し，危険対定理が dTRS で成立する条件を示す．

5.4.1 危険対定理

dTRS の危険対の概念を CTRS の場合と同様に与える [8]. $l_1 \to r_1 \Leftarrow c_1$，$l_2 \to r_2 \Leftarrow c_2$ を $\mathcal{V}ar(l_1, r_1) \cap \mathcal{V}ar(l_2, r_2) = \emptyset$ となるように変数を名前替えした規則とする．$p \in \mathcal{O}(l_1)$ を $l_1|_p$ が変数でない位置，σ を $l_1|_p\sigma \equiv l_2\sigma$ となるような最汎単一化子としたとき，$\langle r_1\sigma, (l_1[r_2]_p)\sigma\rangle \Leftarrow c_1\sigma \wedge c_2\sigma$ を**条件付き危険対**と呼ぶ．このとき，2つの規則は**重なる**という．dTRS R のすべての危険対の集合を $CP(R)$ で表す．さらに，R の規則と $l \to r \Leftarrow c$ の間で形成される危険対の集合を $CP(R, l \to r)$ と書く．ある条件付き危険対 $\langle s, t\rangle \Leftarrow c$ に対して，条件 c を充足するような代入 σ が存在するときその危険対は**可能 (feasible)** であるという．また，そうでないときその危険対は**不能 (infeasible)** であるという．

R を dTRS とする．項 s，t が**条件 c の元で会同関係にある**とは，任意の σ に対して，$c\sigma$ が真ならば $s\sigma \downarrow_R t\sigma$ が成り立つことである．また，条件付き項対 $\langle s, t\rangle \Leftarrow c$ が会同関係にあるとは，s と t が c の元で会同関係にあることである．

TRS の合流性に関する危険対補題は，dTRS 上ではそのまま成り立たない．書換えの分岐が重なりのない上下の位置での場合に問題が生じる．そこで，条件部により強い制限を与える必要がある．

定義 5.2 (強決定可能) 条件 c が**二項関係 $\approx$ の元で強決定可能**とは，すべての代入 θ に対して，$c\theta$ が真ならばすべての $\theta_\approx$ に対して $c\theta_\approx$ も真となることである．ただし，$\theta_\approx$ は $\forall x \in \mathcal{X}.\ x\theta \approx x\theta_\approx$ を満たすとする．dTRS R が**二項関係 $\approx$ の元で強決定可能**であるとはすべての規則 $l \to r \Leftarrow c \in R$ の条件 c が二項関係 $\approx$ の元で強決定可能であることである.,

強決定可能な dTRS において以下の条件付き危険対補題・定理が成り立つ.

補題 5.3 (条件付き危険対補題) R を強決定可能な dTRS, s, t_0, t_1 を項とする．$s \to_R t_i$ $(i = 0, 1)$ ならば，以下のどちらかを満たす.

- $t_0 \downarrow_R t_1$.
- $t_i = s[u_i]_p$ $(i = 0, 1)$．ただし，$\langle v_0, v_1\rangle \Leftarrow c \in CP(R)$ と代入 σ が存在して， $c\sigma$は真 かつ，$u_i = v_i\sigma$ $(i = 0, 1)$ または $u_{i-1} = v_i\sigma$ $(i = 0, 1)$.

定理 5.4 (条件付き危険対定理) R を強決定可能な dTRS とする．R が局所合流性を持つとき，かつそのときに限り，R から作られるすべての可能な R の条件付き危険対は会同関係にある．

5.4.2 dTRS の完備化

dTRS の完備化手続きを TRS の完備化手続き [1] [13] を元に提案する．条件部の考慮以外は，基本的に推論規則もアルゴリズムも同じである．

図 4 で与える推論規則は (E, R) に作用する．ただし，E は等式の有限集合であり，R は書換え規則の有限集合である．直観的に，E には入力の等式か，まだ書換え規則へと変換していない危険対が含まれる．R は停止性を持つ書換え規則の集合である．基本的な完備化として，R の停止性は完備化手続きに入力として与えられる簡約化順序 $\succ$ によって保証される．目標は (E_0, R) から R' が完備で $E_0 \cup R$ に等価であるような $(\emptyset, R')$ に変換することである．しかし，完備化手続きは一般に停止するとは限らない.

orientaion

$$\frac{(E \cup \{s = t \Leftarrow c\}, R)}{(E, R \cup \{s \to t \Leftarrow c\})} \quad \text{ただし，} s \succ t$$

composition

$$\frac{(E, R \cup \{s \to C[v\rho] \Leftarrow c\})}{(E, R \cup \{s \to C[w\rho] \Leftarrow c\})} \text{ ただし，} v \to w \Leftarrow c' \in R \text{ かつ } c \vdash c'\rho$$

deduction

$$\frac{(E, R)}{(E \cup \{s' = t' \Leftarrow c'\}, R)}$$

ただし，$s \to t \Leftarrow c \in R$ かつ，$\langle s', t'\rangle \Leftarrow c' \in CP(R, s \to t \Leftarrow c)$

collapse

$$\frac{(E, R \cup \{C[s\theta] \to t' \Leftarrow c'\})}{(E \cup \{C[t\theta] = t' \Leftarrow c \wedge c'\}, R \cup \{C[s\theta] \to t' \Leftarrow c' \wedge \neg c\})}$$

ただし，$s \to t \Leftarrow c \in R$

simplification

$$\frac{(E \cup \{s = C[v\rho] \Leftarrow c\}, R)}{(E \cup \{s = C[w\rho] \Leftarrow c\}, R)} \text{ ただし，} v \to w \Leftarrow c' \in R \text{ かつ } c \vdash c'\rho$$

E-deletion

$$\frac{(E \cup \{s = t \Leftarrow c\}, R)}{(E, R)} \quad \text{ただし，} s \equiv t \text{ または } c \text{ は恒偽}$$

R-deletion

$$\frac{(E, R \cup \{s \to t \Leftarrow c\})}{(E, R)} \quad \text{ただし，} c \text{ は恒偽}$$

図 4　dTRS 完備化のための推論規則

入力：R，E．出力：R_i.
$R_0 := R$;　$E_0 := E$;　$i := 0$; (R_0 に (5) を適用)
$E_i = \emptyset$ となるまで以下を繰り返す.
（1） orientation を使い，等式集合 E の中から両辺に順序が付くものを一つ選ぶ．順序の付く等式が存在しないとき，完備化失敗. 等式集合 $E = \emptyset$ なら完備化成功.
（2） composition で R のすべての規則の右辺を正規形にする.
（3） deduction を使い，$s \to t \Leftarrow c$ と $\{s \to t \Leftarrow c\} \cup R$ の間にできる全ての危険対を E に加える.
（4） collapse を使い，R の規則のうちで左辺が s の例 (instance) となる部分項を持つ規則の条件部に $\neg c$ を加える (option).
（5） simplification で E のすべての等式の両辺を正規形にする.
（6） E-deletion を使い，冗長な等式をすべて取り除く.
（7） R-deletion を使い，R の不要な規則をすべて取り除く.
（8） $R_{i+1} := R_i$;　$E_{i+1} := E_i$;　$i := i + 1$;

図 5　dTRS 完備化手続き

図 4 の推論規則を 1 回適用することを $(E, R) \vdash_{\mathcal{C}} (E', R')$ と書く．また，任意の代入 σ に対して $c\sigma$ が真ならば $c'\sigma$ も真 (すなわち，$\neg c\sigma \vee c'\sigma$ が真) であることを $c \vdash c'$ と書く.

KB 完備化手続き [1] を dTRS へ拡張した **dTRS 完備化手続き**を図 5 に示す．条件 c 中の自由変数は任意 (すなわち $\forall$) として恒偽，恒真判定を行う．入力 R が TRS であるときは，図 5 の手続きは TRS の KB 完備化手続きと一致する.

dTRS 完備化手続きの (1) から (8) までの手続きを (E, R) に順に適用することを $(E, R) \vdash_{\mathcal{K}} (E', R')$ と書く．

定理 5.5 E_0 を等式の集合，R_0 を完備な強決定可能な dTRS とする．dTRS 完備化手続きで，$(E_0, R_0) \vdash_{\mathcal{K}} \cdots \vdash_{\mathcal{K}} (\emptyset, R_n)$ を得たとする．このとき，

（1） R_n は $E_0 \cup R_0$ に等価である．

（2） R_n は完備である．

（3） $\to_{R_0} \subseteqq \overset{*}{\to}_{R_n}$ $(R_0 \subseteqq R_n)$ ならば，R_n は強決定可能．

5.5 sum の検証例

最後に本手法による検証例を挙げる．手続き型で書かれた関数 sum1 (図 3) と仕様として pdTRS で書いた関数 sum0 (R_{sum0}) を与え，それらの等価性を検証する．以降で $Add(,)$ は + を用いた中置記法で表す．手続き型で書かれた関数 sum1 を pdTRS に変換したものが例 4.1 の R_{sum1} である．R_{sum1} に composition を施して簡単化した物が以下の R'_{sum1} である．

$$R'_{sum1} =$$

$$\begin{cases} sum1(x) & \to & U_2(x, 0, 0) & (1) \\ U_2(x, i, z) & \to & U_2(x, S(i), z + S(i)) \Leftarrow i < x & (2) \\ U_2(x, i, z) & \to & z \Leftarrow i \geqq x & (3) \end{cases}$$

$$R_{sum0} =$$

$$\begin{cases} sum0(0) & \to & 0 & (4) \\ sum0(S(x)) & \to & sum0(x) + S(x) & (5) \end{cases}$$

$R_0 = R'_{sum1} \cup R_{sum0}$ 上で等式 $sum1(x) = sum0(x)$ が帰納的定理であることは図 6 のように証明できる．よって関数 sum1 と関数 sum0 は等価であるといえる．なお，検証には補題 $U_2(S(x), k, y) = U_2(x, k, y) + S(x) \Leftarrow k \leqq x$ を導入することが必要であったため，補題から証明した．

補題 $U_2(S(x), k, y) \to U_2(x, k, y) + S(x) \Leftarrow k \leqq x$ (6) の証明.
$(\{(6)\}, R_0) \vdash_{\mathsf{ori}} (\emptyset, R_0 \cup \{(6)\})$
$\vdash_{\mathsf{com}}$ $\vdash_{\mathsf{ded}} (\{(7), (8)\}, R_0 \cup \{(6)\})$
$\vdash_{\mathsf{sim}} (\{(8), (9)\}, R_0 \cup \{(6)\})$
$\vdash_{\mathsf{sim}} (\{(8), (10)\}, R_0 \cup \{(6)\})$
$\vdash_{\mathsf{Ede}} (\emptyset, R_0 \cup \{(6)\})$: 成功

証明に出現する等式・規則.
$U_2(S(x), S(k), y + S(k))$
$\to U_2(x, k, y) + S(x) \Leftarrow k < x + 1 \wedge k \leqq x$ (7)
$y = U_2(x, k, y) + S(x)\rangle \Leftarrow k \geqq x + 1 \wedge k \leqq x$ (8)
$U_2(x, S(k), y + S(k)) + S(x) = U_2(x, k, y) + S(x) \Leftarrow k \leqq x$ (9)
$U_2(x, S(k), y + S(k)) + S(x)$
$= U_2(x, S(k), y + S(k)) + S(x) \Leftarrow k \leqq x$ (10)

帰納的定理 $sum1(x) \to sum0(x)$ (11) の証明.
$(\{(11)\}, R \cup \{(6)\}) \vdash_{\mathsf{sim}} (\{(12)\}, R \cup \{(6)\})$
$\vdash_{\mathsf{ori}} (\emptyset, R \cup \{(6), (12)\})$
$\vdash_{\mathsf{ded}} (\{(13), (14)\}, R \cup \{(6), (12)\})$
$\vdash_{\mathsf{sim}} (\{(14), (15)\}, R \cup \{(6), (12)\})$
$\vdash_{\mathsf{sim}} (\{(14), (16)\}, R \cup \{(6), (12)\})$
$\vdash_{\mathsf{sim}} (\{(14), (17)\}, R \cup \{(6), (12)\})$
$\vdash_{\mathsf{Ede}} (\emptyset, R \cup \{(6), (12)\})$: 成功

証明に出現する等式・規則.
$U_2(x, 0, 0) \leftarrow sum0(x)$ (12)
$0 = U_2(0, 0, 0)$ (13)
$sum0(0) + S(0) = U_2(S(0), 0, 0)$ (14)
$0 = 0$ (15)
$U_2(0, 0, 0) + S(0) = U_2(S(0), 0, 0)$ (16)
$U_2(0, 0, 0) + S(0) = U_2(0, 0, 0) + S(0)$ (17)

図 6 完備化のによる検証例

6. おわりに

本手法で対象とする手続き型言語は配列、ポインタが扱えないなど制限強いが，初心者を対象とした C プログラム演習での採点などへの利用が期待できる．完備化には簡約化順序 $\succ$ が必要であるが，今回の sum 関数の検証では人間の直観にたよった順序に基づいて完備化を行った．規則の順序付けを経路順序で自動的に行うと規則 (2) には順序が付かない．どのように順序を与えるかが今後の課題である．pdTRS の停止性の証明法も今後の課題である．今回は自然数のみを扱ったが，整数への拡張は容易に行える．

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] F. Baader and T. Nipkow : "Term Rewriting and All That," Cambridge University Press, 1998.
[2] E. Ohlebusch : "Advanced Topics in Term Rewriting", Springer, 2002.
[3] H. Ganzinger : "A Completion Procedure for Conditional Equations," Journal of Symbolic Computation, vol. 11, pp. 51–81, 1991.
[4] L. Bachmair : "Canonical Equational Proofs," Progress in Theoretical Computer Science, 1991.
[5] S. Kaplan : "Simplifying conditional term rewriting systems: unification, termination and confluence," Journal of Symbolic Computation, vol. 4, pp. 295–334, 1987.
[6] D. C. Cooper : "Theorem Proving in Arithmetic without Multiplication," Machine Intelligence, no. 7, pp. 91–99, Edinburgh University Press, 1972.
[7] M. ヘネシー (著)，荒木 啓二郎，程 京徳 (共著) : "プログラミング言語の意味論入門," サイエンス社，1993.
[8] 高橋 宣孝，酒井 正彦，外山 芳人 : "条件付き項書換え系の合流性について," 信学技報 COMP94-65, pp. 105–111, 1994.
[9] A. Bouhoula : "Automated theorem proving by test set induction," Journal of Symbolic Computation, vol. 23, pp. 47–77, 1997.
[10] R. S. Boyer and J. S. Moore : "A computational logic," Academic Press, 1979.
[11] G. Huet and J. M. Hullot : "Proof by induction in equational theories with constructors," Journal of Computer and System Science, vol. 25, pp. 239–266, 1982.
[12] D. R. Musser : "On proving induction roperties of abstract data types," In Proc. 7th ACM Symp. Principles of Programming Languages, pp. 157–162, 1980.
[13] L. Backmair : "Canonical Equational Proofs," Progress in Theoretical Computer Science. Birkhäuser, 1991.
[14] M. Huth, M. Ryan : "Logic in Computer Science," Cambridge University Press, 2000.