ソフトウェア工学 113－1
（1997．3．7）

Martin-Löf の型理論に基づく プログラム支援システムの構築

縄田和裕　藤田憲悦†　石山明宏　中島隆

九州工業大学情報工学部

概要. Martin-Löf の型理論における構成的証明を利用して、構成的プログラミングに基づいたプログラム生成とその支援を行うシステムの構築について述べる。構成的プログラミングとは、プログラムの論理的仕様を構成的に証明し、その証明から正当性の保証されたプログラムを抽出する方法である。この手法に基づいて構成的証明からプログラムを抽出するシステムと、証明項から証明図を表示するシステムの実現について述べる。最後に未完成の証明と一般化証明との関係、およびそのプログラミングへの応用について考察する。

A program generation system based on Martin-Löf's type theory

Kazuhiro Nawata, Kenetsu Fujita,
Akihiro Ishiyama, Takashi Nakashima

Kyushu Institute of Technology
Faculty of Computer Science and Systems Engineering

Abstract. This paper gives an experimental implementation of a program generation system based on Martin-Löf's monomorphic type theory. The notion of constructive programming is known as a programming method to obtain correct programs in the sense of satisfying given specifications. Our system make it possible to extract programs from constructive proofs and display proof figures from proof terms. Finally, we observe a computational use of unfinished proofs and an interesting relation between unfinished proofs and generalized proofs.

1　はじめに

構成的数学の計算機への応用の１つに構成的証明からのプログラム抽出 [1, 3] がある。そこで用いられる理論の中で、特に重要と考えられるものとして構成的型理論が挙げられる。本論文では、代表的な構成的型理論である Martin-Löf の型理論 [4] を利用してプログラムの抽出を実現したシステムについて述べる。

まず最初に構成的証明を構築する論理体系として用いた Martin-Löf の型理論について述べる。本研究では構成的証明を計算機上で行うために、Chalmers 大学で作られたシステム (ALF[5, 6]) を利用する。これは、計算機で証明を構築するためのシステムであり、論理体系を定義することでそれに基づいた証明を構築することができる。次に、ALF で構築した構成的証明からプログラムを抽出するプログラム抽出システムについて述べ、ALF で構成した証明項を証明木の形で表示する証明図表示システムについて述べる。最後に、未完成の構成的証明の意味とそのプログラムへの応用について考察する。

†本研究の一部は文部省科学研究費補助金，及び大川情報通信基金による援助を受けている。

2 Martin-Löf の型理論

構成的証明を行う体系として Martin-Löf の型理論 (monomorphic theory)[4] を用いる。これは、Curry-Howard の原理を利用した論理的な推論と型の推論を融合した理論であり、構成的型理論と呼ばれるものである。Martin-Löf の型理論の骨子は、次のような形式をした judgement からなっている。

- $A \in Set$　　A は型である。
- $A = B \in Set$　　A と Bは等しい型である。
- $a \in A$　　a は型 A のオブジェクトである。
- $a = b \in A$　　a と b は等しいオブジェクトである。

judgement の意味は、形式的には推論規則により定められる。推論規則とは、型とオブジェクトがどのように形成されるかを定義した規則である。

プログラミングの立場においては、型を命題 (プログラムの論理的仕様)、オブジェクトをプログラムとみなす。まず示すべき命題を与え、それに推論規則を繰り返し適用することでその命題のオブジェクト、すなわち証明を得る。ここで、依存積Πを例に挙げて推論規則について説明する。

$$\frac{A \in Set \quad B \in (A)Set}{\Pi(A,B) \in Set}\ \Pi - form \qquad \frac{A \in Set \quad B \in (A)Set \quad b(x) \in B(x)\,[x \in A]}{\lambda(A,B,b) \in \Pi(A,B)}\ \Pi - intro$$

$$\frac{A \in Set \quad B \in (A)Set \quad f \in \Pi(A,B) \quad a \in A}{apply(A,B,f,a) \in B(a)}\ \Pi - elimination$$

$$\frac{A \in Set \quad B(x) \in Set\,[x:A] \quad b(x) \in B(x)\,[x:A] \quad a \in A}{apply(A,B,\lambda(A,B,b),a) = b(a) \in B(a)}\ \Pi - equality$$

これらはいずれも型Πに関する推論規則であり、それぞれ型の形成規則・導入規則・除去規則・計算規則を表している。特に導入規則は、A が型であり、B が A に依存する型であり、b が B のオブジェクトであるならば、$\lambda(A,B,b)$ は$\Pi(A,B)$ のオブジェクトであることを示している。このように、型$\Pi(A,B)$ のオブジェクトを$\lambda(A,B,b)$ とすることで、その型 (命題) の意味を定めている。

この Martin-Löf の型理論を用いて構成的証明を ALF[5, 6] を利用して構築する。ALF とは、計算機上での構成的な証明の構築を支援する proof-editor である。Martin-Löf の型理論の推論規則を定義することで証明を構築することができる。証明はトップダウン方式で構築し、各推論のステップにおいてどの推論規則を適用するかの選択はユーザが指定する。

3 型付き関数型言語

ALF で構築した構成的証明をここで定義するプログラム言語に変換する。Martin-Löf の型理論から型の依存性を取り除いた体系を型システムとしてもつ関数型言語を考える。基本型となるものは unit と自然数の型 N、矛盾を表す型⊥の 3 つであり、それに直積型・直和型・関数型を加えたものを型として導入する。それらの型を持つ項に関する定義は以下に示す通りである。

- 型 $\tau ::= unit \mid N \mid \tau \times \tau \mid \tau + \tau \mid \tau \to \tau \mid \bot$
- 項 $e ::= () \mid 0 \mid succ(e) \mid x \mid < e,e > \mid e\ e \mid fst(e) \mid snd(e) \mid$
  $nrec(e,e,e) \mid inl(e) \mid inr(e) \mid case(e,e,e) \mid \lambda x.e \mid raise(e)$

[型推論規則]

$$():\mathrm{unit} \qquad 0 : \mathrm{N} \qquad \frac{n : N}{succ(n) : N} \qquad \frac{a : \bot}{raise(a) : C}$$

$$\frac{a:N \quad b:C \quad c(x,y):C[x:N,\ y(x):C]}{nrec(a,b,c):C} \qquad \frac{a:A+B \quad b(x):C\ [x:A] \quad c(y):C\ [y:B]}{case(a,b,c):C}$$

$$\frac{a:A \quad b:B}{<a,b>:A\times B} \qquad \frac{a:A\times B}{fst(a):A} \qquad \frac{b:A\times B}{snd(a):B}$$

$$\frac{f:A\to B \quad a:A}{f\ a:B} \qquad \frac{b(x):B\ [x:A]}{\lambda x.b:A\to B} \qquad \frac{a:A}{inl(a):A+B} \qquad \frac{b:B}{inr(b):A+B}$$

[計算規則の定義]

$(\lambda x.b)a = b(a)$
$case(inl(a),b,c) = b(a)$
$case(inr(a),b,c) = c(a)$
$fst(<a,b>) = a$
$snd(<a,b>) = b$
$nrec(0,b,c) = b$
$nrec(succ(a),b,c) = c(a,nrec(a,b,c))$

$raise(raise(a)) = raise(a)$
$raise(a)b = raise(a)$
$fst(raise(a)) = raise(a)$
$snd(raise(a)) = raise(a)$
$case(raise(a),b,c) = raise(a)$

4 証明からプログラムへの変換システム

ALF で構築した構成的証明からプログラム言語への変換 [7] について考える。証明からのプログラム抽出には 2 つの変換を考えなければならない。1 つはプログラムの論理的仕様からプログラムの型への変換であり、もう 1 つは証明項からプログラムへの変換である。それぞれの変換規則を定義し、その妥当性を示す。

4.1 型の変換

最初に基本型に関する変換を定義する。自然数型 Nはそのまま N に変換する。矛盾を表す型 Emp は $\perp$に変換する。型 A のオブジェクト x と y が等しいということを表す型 $Id(A,x,y)$ は、プログラムには不要な情報と解釈し、$unit$ 型に変換する。$Less(x,y)$ についても同様である。

(1) $N^* = N$

(2) $Emp^* = \perp$

(3) $Id(A,x,y)^* = unit$

(4) $Less(x,y)^* = unit$

次に Martin-Löf の型理論における型の依存性を取り除く変換について考える。Π型・Σ型・+型はそれぞれ関数型・直積型・直和型に変換する。さらに、$unit$ 型を除去する変換についても考える。

(5) $+(A,B)^* = A^* + B^*$

(6) $$\Pi(A,B)^* = \begin{cases} A^*\to B^* & [A^*\neq unit; B^*\neq unit \text{ のとき}] \\ B^* & [A^* = unit \text{ のとき}] \\ unit & [B^* = unit \text{ のとき}] \end{cases}$$

$$(7)\Sigma(A,B)^* = \begin{cases} A^* \times B^* & [A^* \neq unit; B^* \neq unit \text{ のとき}] \\ B^* & [A^* = unit \text{ のとき}] \\ A^* & [B^* = unit \text{ のとき}] \end{cases}$$

ただし A^*は、型 A を変換規則にしたがって再帰的に変換した型を表す。

4.2 証明項の変換

次に、オブジェクト (証明項) からプログラムへの変換について考える。Martin-Löf の型理論に基づいて行った解の存在定理の構成的証明の中には、主に型の情報と対象の構成に関する情報とその対象が存在することの証明の情報の 3 種類の情報が含まれている。その中の対象の構成に関する情報のみを保存し、さらに型の情報にしたがって unit 型を持つオブジェクトの部分を除去するように変換規則を定義する。例えば$\lambda(A,B,[x]b)$ と $pair(A,B,a,b)$ は、次のように変換規則を定義する。

$$\underline{\lambda(A,B,[x]b)} = \begin{cases} \lambda x.\underline{b} & [A^* \neq unit; B^* \neq unit \text{ のとき}] \\ \underline{b}[x := ()] & [A^* = unit \text{ のとき}] \\ () & [B^* = unit \text{ のとき}] \end{cases}$$

$$\underline{pair(A,B,a,b)} = \begin{cases} (\underline{a},\underline{b}) & [A^* \neq unit; B^* \neq unit \text{ のとき}] \\ \underline{b} & [A^* = unit \text{ のとき}] \\ \underline{a} & [B^* = unit \text{ のとき}] \end{cases}$$

ただし、A^*とは型 A を上記の型変換規則に従って変換した型、$\underline{a}$とはオブジェクト a を再帰的に変換したものである。$pair(A,B,a,b)$ の引数 A,Bはそれぞれ a,b の型を表しているので除去し、再帰的に a,b を変換させる。

ここで定義した変換規則を用いて得られるものは、ALF の型とその証明の関係を保存している。つまり、型 A を型変換規則で変換したものは、その型 A の証明を証明変換規則で変換したものの型になっている。このことを次に示す。

命題 1.(変換規則の妥当性) ： ALF における型を A、その証明を a とする。型 A を変換したものを A^*、証明 a を変換したものを$\underline{a}$とすると、任意の ALF における型とその証明に対して、$\Gamma \vdash a : A$ ならば、定義した関数型言語において$\Gamma^* \vdash \underline{a} : A^*$である。

ただし、Γは ALF における前提を表し、Γ^*はそれを変換したものである。この命題は ALF における証明の構成に関する帰納法で証明することができる。

□ **base-step** ： less1(m) : Less(zero,succ(m)) についてのみ示す。変換規則より、less1(m) は () に変換され、Less(zero,succ(m)) は unit 型に変換される。定義した型付き関数型言語の型推論規則より、():unit である。したがって、明らかに成り立つ。

□ **induction-step** ： $\lambda(A,B,[x]b) : \Pi(A,[x]B)$ についてのみ示す。Martin-Löf の型理論の推論規則

$$\frac{A \in Set \quad B \in (A)Set \quad b(x) \in B(x)\,[x \in A]}{\lambda(A,B,[x]b) \in \Pi(A,[x]B)}\ \Pi - introduction$$

より、b(x):B(x) [x:A] から導入されている。ただし $A^* \neq unit, B^* \neq unit$ の場合を考える。帰納法の仮定より、B(x) を変換したものはその型の証明 b(x) を変換したものの型になっている。よって定義した関数型言語の推論規則から$\lambda x.\underline{b}$ ： $A^* \to B^*$を導入できる。したがって、変換規則

$$\Pi(A,B)^* = A^* \to B^*$$

$$\underline{\lambda(A, B, [x]b)} = \lambda x.\underline{b}$$

は、妥当な変換である。他の変換についても同様に示すことができる。よって、定義した型と証明の変換規則は妥当な変換である。

4.3 変換例

任意の自然数 n を与えると $\lfloor\sqrt{n}\rfloor$ を計算するプログラムを考える。プログラムの論理的仕様は、

$$\forall n : N.\exists h : N.h^2 \leq n < (h+1)^2$$

と記述することができ、これを ALF で表現すると

$$(n : N; D(n))\Sigma(N, [r]\Sigma(Lesseq(square(r), n), [h]Less(n, square(succ(r)))))$$

となる。ここで n:N と D(n) は前提であり、D(n) は次のように再帰的に定義した型 [3] である。

- dom1 : D(0)
- n:N,h:D(div4 n) ならば dom2(n,h):D(n)

ただし、(div4 x) は x を 4 で割った商を返す関数である。D(n) をプログラムには不要な型と解釈し unit に変換すると考えると、この仕様は型変換規則に従って、次のような平方根を求めるプログラムの型を得ることができる。

$$N \rightarrow N$$

次に、ALF で平方根の解の存在定理を D(n) に関する帰納法で証明する。(証明の詳細は省略)

root(0,dom1) = pair(N, [h1]Σ(. . .), 0, . . .)

root(n,dom2(n,h)) = when(Less(. . .), Lesseq(. . .), lemma1(. . .), [h1]Σ(. . .), . . .)

これを証明項変換規則に従って変換すると次のプログラムを抽出することができる。

root 0 = 0

root n = case(lemma1(n,square(succ(mult(succ(succ(0)),root (div4 n))))),
mult(succ(succ(0)),root (div4 n)),succ(mult(succ(succ(0)),root (div4 n))))

ただし lemma1(x,y) は x < y であるか x≥y であるかを判定する関数である。平方根を計算するこのプログラムは、直観的に表現すると次のようになる。

$root\ 0 = 0$

$root\ n = if\ n < (2 * (root(div4\ n)) + 1)^2\ then\ 2 * root(div4\ n)$
$else\ 2 * (root(div4\ n)) + 1$

この例の証明は型 D(n) に関する帰納法で証明している。したがって D(n) が任意の自然数で成り立つことを示さなければ得られるプログラムが停止することが保証されない。その意味では、この例は仕様を停止性と部分妥当性に分割して示した証明である。しかし、D(n) はプログラムの情報を含まないので、プログラムを抽出することは可能である。

実現したシステムは ALF で構築した証明を入力として受け取る。用意した型変換規則と証明変換規則にしたがって型と証明を変換し、抽出したプログラムとその型を出力する。ここで定義した以外の型やオブジェクトを入力した場合、システムはユーザに変換規則を要求し、規則入力後処理を再開する。システムとユーザとの対話機能により、ユーザが自由に型とオブジェクトを定義できるオープンな ALF のシステムの特徴を生かしている。ただし、ユーザが新たに定義した型とオブジェクトに対する変換規則については、別途、その変換の妥当性を証明しなければならない。また、変換規則のデータベースはシステムとは独立させているので、証明の内容によって変換規則を容易に変更することができ、ユーザの意図を反映した変換を行うことができる。さらにこのシステムは未完成の証明からプログラムを抽出することができる。そのプログラムの意味と利用方法については第 6 章で詳しく述べる。

5 証明図表示システム

ALF では、証明の過程は証明木としてではなく証明項で構成されている。証明に対する直観的な理解を支援するために証明項が表す証明図を表示させるシステムについて考える。これは、ALF で構築した証明項から型を抽出し、それを証明木の形に構成することで実現することができる。証明図を表示させる利点は、証明の過程を分かりやすくするだけでなく、証明から得られるプログラムと証明との対応関係を直観的に理解でき、プログラムを構築する上で有益である。

証明項から証明図への変換を $pair(A,B,a,b)$ を例に説明する。$pair(A,B,a,b)$ の型は推論規則から $\Sigma(A,B)$ である。同様に a と b もそれぞれ型 A と B のオブジェクトである。このとき、$pair(A,B,a,b)$ は Martin-Löf の型理論の推論規則

$$\frac{A \in Set \quad B \in (A)Set \quad a \in A \quad b \in B(a)}{pair(A,B,a,b) \in \Sigma(A,B)}\ \Sigma - introduction$$

により導出される。したがって、$pair(A,B,a,b)$ は次のように変換することでそのオブジェクトが表す証明図を構築することができる。

$$\frac{\tilde{a} \quad \tilde{b}}{\Sigma(A,B)}$$

ただし、$\tilde{a}$は a を再帰的に変換したものである。他のオブジェクトに対しても同様の変換をすることで、ALF で構築した証明項が表す証明図を生成することができる。

この変換にしたがって証明項から証明図へ変換する証明図表示システムを実現した。本システムは、オブジェクトの変換規則が定義されていない場合、その変換をユーザに問い合わせる機能を持たせている。これにより新たに ALF で定義したオブジェクトにも対応することができる。

6 部分証明と一般化証明

本章では、途中段階の証明 (部分証明) と一般化証明との関係、および高階単一化を使って部分証明と具体例に対する証明から完成した証明を構築する方法について考察する。

6.1 一般化証明

文献 [2] では、次の 2 つの一般化証明について考察されている。変数化による一般化と数の一般化である。変数化による一般化とは、定数等の項を単純に変数で置き換えることによって証明を一般化することである。項の列$\vec{a}$を含む命題 $B[\vec{a}]$ の証明を変数化により一般化した命題の証明は $\forall \vec{x} : \vec{D}.\vec{A}[\vec{x}] \rightarrow B[\vec{x}]$ という型を持つ。ただし、$\vec{A}[\vec{x}]$ は変数の列$\vec{x}$に関する適当な条件の列を表している。数の一般化とは、具体的な数に対する命題 (例えば B(3)) の証明を一般化して任意の自然数 n に対する命題 B(n) の証明を得ることである。

6.2 部分証明

ALF では、証明はトップダウン方式 (全体から部分へと進めていく方式) で構築する。したがって証明の途中段階における未完成の証明は、全体の構造は完成しているが葉の部分の証明が一部欠落している証明である。

任意の自然数 n に対するプログラムの論理的仕様 A(n) の証明において、欠落している部分？がある型 B(n):Set を持つとき、この未完成の証明を A(n) に対する部分証明と定義する。証明の未完成の部分の型は ALF が自動的に推論するので、未完成の証明が部分証明かどうか容易に判断することができる。ここで、部分証明において欠落している部分？に変数化による一般化を適用することで、一般化証明を得るこ

とができる。この一般化証明の型は、一般化された命題 $\forall n : N.B(n) \rightarrow A(n)$ である。この一般化証明から得られるプログラムとその利用法について次の節で考察する。

6.3 仕様の分割と部分証明

証明からプログラムを抽出するという立場において部分証明の意味を考えると、それには次の2通りの場合がある。証明の未完成の部分 B(n) にプログラムに必要な情報が含まれていない場合と含まれている場合である。

前者の場合、部分証明の中にはプログラムに必要な情報が全て含まれているのでプログラムを抽出することができるが、その型は一般化された命題 $\forall n : N.B(n) \rightarrow A(n)$ である。しかし、未完成の部分の命題 B(n) が任意の n について成り立つことを示すことで、得られたプログラムは仕様 A(n) を満たすことを示すことができる。未完成の証明の部分は計算機上で行う必要は無い。このことを利用して証明の手間を簡略化することができる。

次に、証明の未完成の部分にプログラムに必要な情報が含まれている場合を考える。この場合、部分証明から抽出されるプログラムは、未完成の部分を変数で置き換えることで一般化されたプログラムである。一般化された変数部分を補いその証明を与えれば、本来の仕様を満たすプログラムとその正当性を得ることができる。その一つの方法として、具体例に対する証明と一般化証明との単一化による完全な証明の構築について考察する。

6.4 一般化証明と具体例に対する証明

単一化とは変数を含む項の間の等式系を解く問題のことで、等式系の解のことを単一化代入という。ここでは、Martin-Löf の型理論におけるオブジェクトを単一化の対象とする。

具体例に対する証明とは、命題の変数部分を具体的な値に置き換えた命題の証明である。ここで述べる部分証明を利用した完全な証明の構築の方法は、まず部分証明を変数化により一般化し、その一般化された証明と具体例に対する証明を高階単一化することで、任意の自然数で成り立つ証明を構築するものである。ただし、単一化の対象は帰納法の仮定が欠落していない部分証明のみを考え、単純型理論に対する高階単一化 [8] を拡張した単一化手続きは与えられていると仮定する。ここで、単一化の例として足し算の交換則について考える。ただし、具体例の証明は命題 Id(N,plus(2,4),plus(4,2)) の証明であり、証明中の各 lemma は算術に関する補題を表している。

[足し算の交換則] `plus_sym : (m,n:N)Id(N,plus(m,n),plus(n,m))`

■ 一般化証明 ■

```
plus_sym(0,n) =
        lemma2(plus(n,0),plus(0,n),lemma3(plus(0,n)))
plus_sym(succ(h),n) =
        lemma4(plus(succ(h),n),
               succ(plus(n,h)),
               plus(n,succ(h)),
               lemma5(plus(h,n),plus(n,h),succ,plus_sym(h,n)),
               X(h,n))
```

■ 具体例に対する証明 ■

```
plus_24 =
        lemma4(plus(succ(succ(0)),succ(succ(succ(succ(0))))),
```

```
succ(plus(succ(succ(succ(succ(0)))),succ(0))),
plus(succ(succ(succ(succ(0)))),succ(succ(0))),
lemma5(plus(succ(0),succ(succ(succ(succ(0))))),...),
lemma2(plus(succ(succ(succ(succ(0)))),succ(succ(0))),
       succ(plus(succ(succ(succ(succ(0)))),succ(0))),
       lemma6(succ(succ(succ(succ(0)))),succ(0))))
```

最初の証明は、第一引数の自然数に関する帰納法による証明の未完成の部分を関数X(h,n)と置き換えた一般化証明である。これと具体例に対する証明とを単一化すると、単一化代入の一つとして

$$X = \lambda h.\lambda n.lemma2(plus(n, succ(h)), succ(plus(n, h)), lemma6(n, h))$$

という代入を得ることができる。

具体例に対する証明と一般化証明との高階単一化により得られる証明は、一般にその正しさは保証されない。したがって、得られるプログラムはその正当性が保証されないが、計算機上での構成的証明の構築を支援する1つの機能として利用することが考えられる。

7 まとめ

本研究で、Martin-Löf の型理論を論理体系として構築した構成的証明からプログラムを抽出する変換規則を定義し、その妥当性を示した。その変換規則に基づいて実現したプログラム抽出システムについて述べた。次に部分証明が一般化された命題の証明であることを示し、その証明から得られるプログラムは一般化されたプログラムであることを述べた。最後に単一化を使って部分証明と具体例に対する証明から完成した証明を構築する一方法について考察を与えた。

参考文献

[1] 小林聡：Program Extraction from Constructive Proofs、コンピュータソフトウェア Vol.7 No.4、1990 Oct.

[2] 萩谷昌己：Higher-order Unification and Generalization of Proofs、人工知能学会誌 Vol.6 No.3、1991 May.

[3] Hayashi S. and Nakano H.：PX A Computational Logic、The MIT Press、1988.

[4] Bengt Nordström, Kent Petersson, Jan M. Smith：Programming in Martin-Löf type theory An introduction、Clarendon press、1990.

[5] Bengt Nordström, Jan M. Smith, Thierry Coquand：Type Theory and Programming、BULLETIN of European Association for theoretical Computer Science Number 52、1994 Feb.

[6] Bengt Nordström：The ALF proof editor、Type for Proofs and Programs、at Båstad、1993 June.

[7] Björn von Sydow：From ALF to LML、proceedings of El Wintermöte 、University of Göteborg and Chalmers University of Technology report 73、1993 June.

[8] Masami Hagiya : From Programming-by-Example to Proving-by-Example、Lecture Notes in Computer Science 526、September 1991.