ソフトウェア仕様とプログラムの導出

Software Specification and Program Derivation

石井 忠夫*

概要

情報化社会においてソフトウェアはその根幹の一端を担っており，その迅速で妥当なソフトウェア開発技術が求められている。ソフトウェアの利用形態や目的の変化等による仕様の追加や変更も含めることで，ソフトウェアの保守作業を「ソフトウェアが発展する過程」と捉えることができる。本稿では，構成的プログラミングの枠組みの中でこの発展過程を形式化するために，Dijkstra の Dutch national flag 問題を例として，ソフトウェア仕様の基本演算である合併，共通部分取り出しおよび差分演算の振る舞いについて議論する。

1 はじめに

情報化社会において，ソフトウェアの開発技術はその根幹の一端を担っているが，短期間に誤りの無いソフトウェア製品を提供する為には，迅速で妥当なソフトウェアの開発技術が求められている。ソフトウェアの開発は,「現実世界に存在する対象物やそれに関連した概念物を計算機上の操作対象物に写す（モデル化）ことで，現実世界に存在するいろんな課題を計算機上で解決することを可能にする」ものである。この時，課題の対象となる現実世界の振る舞いを何らかの形式言語を用いて書き下したものが仕様であり，また，この仕様を満たす計算機上のプログラムを作成するのが，ソフトウェアの開発技術である。しかしながら，仕様を満たすプログラムを開発する作業は，この仕様を論理式（定理）と考えると，数学者が行うその論理式（定理）を証明する行為に相当し，プログラム自身はその証明自身に対応すると云える。このようなプログラム開発の可能性は 1970 年代に R.L.Constable，後藤繁樹，佐藤雅彦らにより指摘され構成的プログラミングと呼ばれる。ここでの基本的な考えは，論理式の証明を直観的（または構成的）推論を用いて行うための各種データ型の体系を定義すると，型が論理式（仕様）また型の対象が証明（仕様を満すプログラム）に対応することであり，Curry-Howard 同型対応と呼ばれる。この対応関係を満す型理論として 1980 年代の初めに提案されたのが Martin-Löf の型理論 **MTT** [8, 9] である。

迅速で妥当なソフトウェア開発のためには，その１つの方法として既存のソフトウェア製品（仕様およびそのプログラム）を如何に有効利用して新しいソフトウェアを開発するかが課題となる。これは，保守作業にプログラム不良の修正の他に，ソフトウェアの利用形態や目的の変化等による仕様の追加や変更も含めることで，ソフトウェアの保守作業を如何に効率良く進めるかの問題と考えることができる。片山はこのようなソフトウェアの保守作業を「ソフトウェアが発展すること」と捉え，その発展原理の確立を目指した[6, 7]。先に，上述の構成的プログラミングの枠組みの中で片山が示した発展問題の形式化について検討を試みた[4, 5]。そこでは，構成的証明の中でプログラムの実行には不必要なプログラムの検証部分を削除する手法 [1, 2] の中で用いられた１つの対象からなる型 N_1 を利用して型（および対象）の縮退関係を定義することにより，Martin-Löf の型理論 **MTT** の中で型（および対象）間の順序関係$\sqsubset$を導入した。更に，それを基にして型（および対象）間での合併$\oplus$，共通部分取り出し$\otimes$および差分$\ominus$演算を定義し，発展問題の形式化を試みた。本稿では，構成的プログラミングの例として，Dijkstra の Dutch national flag 問題を取り上げ，ソフトウェア仕様の基本演

*ISHII, Tadao［情報システム学科］

算である合併⊕，共通部分取り出し⊗および差分⊖演算の振る舞いについて検討した。

第2節では構成的プログラミングを可能にする1つの型体系として，Martin-Löf の型理論 **MTT** を概説する。第3節では構成的プログラミングの具体例として，Dijkstra の Dutch national flag 問題を紹介し，また，構成的証明からのプログラム導出を説明する。第4節では縮退関係を逆から見て発展関係を定義することで，ソフトウェア仕様の基本演算である合併⊕，共通部分取り出し⊗および差分⊖演算を **MTT** 体系の中に導入する。第5節では Dijkstra の Dutch national flag 問題を例として，第4節で導入した合併，共通部分取り出しおよび差分演算の振る舞いについて議論する。

2 構成的プログラミング

構成的数学の計算機科学への応用として，プログラムの仕様記述，検証およびプログラムの抽出がある。構成的数学では何かの存在を証明する時に，その対象を構成する具体的な手段を示す必要があり，これが通常の古典的な数学と異なる点である。従って，存在しないと仮定して矛盾を導いても，その対象の存在を証明したことにはならない。即ち，構成的数学では背理法を使用できず，また，これと同等な二重否定の除去$\neg\neg A \to A$や排中立$A \vee \neg A$が認められない。古典論理から背理法（排中立）を除いた論理は直観主義論理と呼ばれ，構成的数学はこの論理の上で展開される。Martin-Löf の型理論 **MTT** は構成的数学が展開可能な構成的型理論の1つであり，Curry-Howard 同型対応を利用して，論理的な推論と型の推論を融合した理論を構成している。

型理論	$\Longleftrightarrow$	構成的論理
型	$\Longleftrightarrow$	論理式
対象	$\Longleftrightarrow$	証明
対象の評価	$\Longleftrightarrow$	証明の正規化

今，入力データの型（論理式）を A，出力データの型（論理式）を $B(x)[x \in A]$ 及び入出力関係の型（論理式）を $C(x, y)[x \in A, y \in B(x)]$ とすると，プログラムの仕様は論理式 $(\forall x \in A)(\exists y \in B(x))C(x, y)$ で表せる。この時，構成的論理を用いてこの論理式の証明を行うと，その証明は対象の具体的な構成手段を含み元の仕様を満たす1つのプログラムと見なせる。

MTT の形式体系は，表現，型，判定および推論規則から構成される。表現とは型の対象であり，表現 a が型 A を持つことを $a \in A$ で表す。型に属する表現間の等号関係＝は次の評価規則により規定される。n 変数を持つ任意の表現 $b(x_1, x_2, ..., x_n)$ において，各変数への別表現 $a_1, a_2, ..., a_n$ の同時代入 $b[a_1, a_2, ..., a_n/x_1, x_2, ..., x_n]$ を表現 b の評価と定める。この時，各型に対して正規形表現と非正規形表現が定義できる。正規形表現は評価しても値が変らない形式であり定数データに対応し，また，非正規形表現は評価により値が変る形式でありプログラムデータに対応する。**MTT** は実際に表3.1に示す型と表現から構成される。但し，$List(A)$ は集合 A の要素から構成されるリストの集合を表し，N と同様に帰納的に定義される集合の型である。有限集合の型 $N_n(n = 0, 1, ..., n)$ を見ると，N_0 は空集合の型であり正規形表現を持たない。また，非正規形表現 $R_0(c)$ は $c \in N_0$ に対して，c が値を持たないので評価が停止しこれは abort 文に対応する。N_1（または T と表す）は1要素集合の型であり唯一の正規形表現 0_1（または t と表す）を持つ。また，非正規形表現 $R_1(c, c_0)$ は $c \in N_1$ が常に t を値として持つので必ず c_0 が評価される。これは nop 文（またはプログラムの連接）に対応する。N_2 は2要素集合の型であり正規形表現 0_2，1_2（または *true*，*false*）を持ち，非正規形表現 $R_2(c, c_0, c_1)$ は $c \in N_2$ の2つの値に応じて c_0，c_1 のいずれかが評価されるので if 文に対応する。また，一般の N_n の非正規形表現は n 分岐の switch 文に対応する。

判定は型理論における基本的な言明であり，**MTT** は(1) A type, (2) $A = B$, (3) $a \in A$, (4) $a = b \in A$ の4つで構成される。(1) A は型であるや A は問題の仕様である，(2) A と B は同じ型であるや A と B は同じ問題である，(3) a は型 A の対象であるや a は問題 A のプログラムである，(4) a と b は型 A の等しい対象であるや a と b は問題 A の等しいプログラムである等に解釈できる。

表 3.1：**MTT** の型と表現

型 名 称	型 形 式	正規形表現	非正規形表現
有限集合（列挙型）	$N_n(n = 0, 1, ..., n)$	$0_n, 1_n, ..., (n-1)_n$	$R_n(c, c_0, c_1, ..., c_{n-1})$
自然数（帰納型）	N	$0, succ(n)$	$R(c, d, e(x, y))$
集合族（依存積型）	$(\Pi x \in A)B(x)$	$(\lambda x)b$	$Ap(c,a)$
直積（レコード型）	$(\sum x \in A)B(x)$	$<a,b>$	$E(c,d(x, y))$
直和（バリアント型）	$A+B$	$inl(a), inr(b)$	$D(c, d(x), e(y))$
等号	$I(A, a, b)$	r	$J(c, d)$
リスト	$List(A)$	$nil, cons(a, b)$	$LR(c, d, e(x, y, z))$
超限帰納型	$(Wx \in A)B(x)$	$sup(a, b)$	$T(c, d(x, y, z))$

MTT の推論規則はこれらの判定を用いて Gentzen の自然演繹体系で与えられる。各型に対して構成的解釈を自然な形で与えるように4つの推論規則が定義される。例えば，リストおよびレコード型の推論規則は次で与えられる。但し，推論規則内の角括弧は前提の仮定を示している。

(1) *List*-formation :
$$\frac{A \text{ type}}{List(A) \text{ type}}$$

(2) *List*-introduction :
$$\frac{}{nil \in List(A)} \qquad \frac{a \in A \quad b \in List(A)}{cons(a, b) \in List(A)}$$

(3) *List*-elimination :
$$\frac{c \in List(A) \quad d \in C(nil) \quad \begin{matrix}[x \in A, y \in List(A), z \in C(y)] \\ e(x, y, z) \in C(cons(x, y))\end{matrix} \quad \begin{matrix}[w \in List(A)] \\ C(w) \text{ type}\end{matrix}}{LR(c, d, e(x, y, z)) \in C(c)}$$

(4) *List*-equality :
$$\frac{d \in C(nil) \quad \begin{matrix}[x \in A, y \in List(A), z \in C(y)] \\ e(x, y, z) \in C(cons(x, y))\end{matrix} \quad \begin{matrix}[w \in List(A)] \\ C(w) \text{ type}\end{matrix}}{LR(nil, d, e(x, y, z)) = d \in C(nil)}$$

$$\frac{a \in A \quad b \in List(A) \quad \begin{matrix}[x \in A, y \in List(A), z \in C(y)] \\ e(x, y, z) \in C(cons(x, y))\end{matrix} \quad \begin{matrix}[w \in List(A)] \\ C(w) \text{ type}\end{matrix}}{LR(cons(a, b), d, e(x, y, z)) = e(a, b, LR(b, d, e(x, y, z))) \in C(cons(a,b))}$$

(5) $\sum$-formation :
$$\frac{A \text{ type} \quad \begin{matrix}[x \in A] \\ B(x) \text{ type}\end{matrix}}{(\sum x \in A)\, B(x) \text{ type}}$$

(6) $\sum$-introduction :
$$\frac{A \text{ type} \quad \begin{matrix}[x \in A] \\ b(x) \in B(x)\end{matrix}}{<a,b> \in (\sum x \in A)\, B(x)}$$

(7) $\sum$-elimination :
$$\frac{c \in (\sum x \in A)B(x) \quad \begin{matrix}[x \in A, y \in B(x)] \\ d(x, y) \in C(<x, y>)\end{matrix} \quad \begin{matrix}[z \in (\sum x \in A)B(x)] \\ C(z) \text{ type}\end{matrix}}{E(c, d(x, y)) \in C(c)}$$

(8) $\sum$-equality :
$$\frac{\begin{matrix}[x \in A] \\ B(x) \text{ type}\end{matrix} \quad a \in A \quad b \in B(a) \quad \begin{matrix}[x \in A, y \in B(x)] \\ d(x, y) \in C(<x, y>)\end{matrix} \quad \begin{matrix}[z \in (\sum x \in A)B(x)] \\ C(z) \text{ type}\end{matrix}}{E(<a, b>, d(x, y)) = d(a,b) \in C(<a,b>)}$$

これらの推論規則を適用して証明を構成するには，各推論規則の結果をゴール，仮定をサブゴールに分解して，そこで得られた小問題をまた証明対象として推論規則を適用する。例えば，(3)の *List*-除去規則を適用するには，次のサブゴールが生成される。

01) $\downarrow C(c)$ の証明（ゴール）(*List*-除去規則)より
02) $\downarrow List(A)$ の証明（サブゴール1）
03) $\uparrow \ni c$：サブゴール1の結果
04) $\downarrow C(nil)$ の証明（サブゴール2）
05) $\uparrow \ni d$：サブゴール2の結果
06) $[x \in A, y \in List(A), z \in C(y)]$
07) $\downarrow C(cons(x, y))$の証明（サブゴール3）
08) $\uparrow \ni e(x, y, z)$：サブゴール3の結果
09) $[w \in List(A)]$
10) $C(w)$ type の証明（サブゴール4）
11) $\uparrow \ni LR(c, d, e(x, y, z))$：ゴールの結果

3 Dutch national flag 問題

Dutch national flag 問題は，最初に構造化プログラミングの演習課題として Dijkstra により提案された。この問題は，赤，白，青の三色のいずれかの色を持つ複数の物が色に関して任意の順序で一列に並んでいる時，ドイツ国旗のように赤，白，青の3つの領域にこの順番で分割して並び換えるプログラムの導出を求めている。この問題を構成的型理論を用いて導出した説明が，例えば Smith の本に載っている[10]。以下ではこの本の内容を基に，構成的プログラミングの枠組みでのプログラム導出について説明する。

問題の前提として，仕様を記述するためにいくつかの仮定を設ける。

(1) 三色のいずれかの色を持つ物の集合を A で表す。また，三色の集合は列挙型 N_3 を用いて，$Color = \{red, white, blue\}$で表す。この時，関数$color : A \to Color$が存在し，$color(x) \in Color[x \in A]$を満たす。

(2) $colored(s) = List(\{x \in A \mid color(x) = s\})$ を用いて，$Red = colored(red)$, $White = colored(white)$および $Blue = colored(blue)$ と定義する。

(3) A は決定可能な等号を持つ。即ち，$N_2 = \{true, false\}$ に対して，
$eq(A, x, y) \in \{z \in N_2 \mid z = true \Longleftrightarrow x =_A y\}[x, y \in A]$

(4) 2つのリストが交換可能 ($l_1 \approx l_2$) を次で定める。$l_1 \approx l_2 = (\forall x \in A) I(N, occin(x, l_1), occin(x, l_2))$
但し，$occin(x, l) = LR(l, 0, (u, v, w)\ if\ eq(A, x, u)\ then\ succ(w)\ else\ w)$ この時，eq の決定可能性より，$\approx$も決定可能となる。

(5) 2つのリストの連結 ($l_1 @ l_2$) を次で定める。
$l_1 @ l_2 = LR(l_1, l_2, (x, y, z)\ cons(x, z))$

(6) $\{<x, y, z> \in A \times B \times C \mid P(x, y, z)\}$ は3つの集合 A, B, C の直積の型で，論理式 $P(x, y, z)$を満たすものである。この時$<a, b, c> \in \{<l_1, l_2, l_3> \in A \times B \times C \mid P(l_1, l_2, l_3)\}[a \in A, b \in B, c \in C, P(a, b, c)\ \text{true}]$ が成り立つ。また，これは $\{z \in A \times (B \times C) \mid P(Fst(z), Fst(Snd(z)), Snd(Snd(z)))\}$ の省略形である。

この時，本問題の仕様は，$S = (\Pi\, l \in List(A))\{<l_1, l_2, l_3> \in Red \times White \times Blue \mid l \approx l_1 @ l_2 @ l_3\}$

と表せる。ここで，$Flag_3(l) = \{<l_1, l_2, l_3> \in Red \times White \times Blue \mid l \approx l_1 @ l_2 @ l_3\}$ とする。この仕様型の対象を証明により導出すると，それが本問題を満たす1つのプログラムとなる。仕様 S は依存積型を持つので，次のΠ-導入規則により，ゴールとサブゴールに分割する。

01) $\downarrow S = (\Pi\, l \in List(A))Flag_3(l)$（ゴール）（Π-導入規則）より
02) $[l \in List(A)]$：仮定
03) $\downarrow Flag_3(l)$（サブゴール1）（*List*-除去規則）より
04) $\downarrow List(A)$（サブゴール2）（仮定）より
05) $\uparrow \ni l$：サブゴール2の結果
06) $\downarrow Flag_3(nil)$（サブゴール3）
07) $\uparrow \ni d$：サブゴール3の結果
08) $[x \in A, y \in List(A), z \in Flag_3(y)]$：仮定
09) $\downarrow Flag_3(cons(x, y))$（サブゴール4）
10) $\uparrow \ni e(x, y, z)$：サブゴール4の結果
11) $[w \in List(A)]$：仮定
12) $Flag_3(w)$ type（サブゴール5）
13) $\uparrow \ni LR(l, d, e(x, y, z))$：サブゴール1の結果
14) $\uparrow \ni (\lambda l)LR(l, d, e(x, y, z))$：ゴールの結果

ここで，4)のサブゴール2は2)の仮定より明らかである。6)のサブゴール3と9)のサブゴール4は，それぞれリスト構成についての基底ケースおよび帰納ケースの証明である。また，12)のサブゴールも明らかである。

基底ケースの証明：List-導入規則より，$nil \in Red$，$nil \in White$，$nil \in Blue$ である。また，nil はリスト連結@に関してidentityであり，リストの交換可能性 $\approx$ はreflexiveであるので，$nil \approx nil @ nil @ nil$ が成り立つ。従って，次が得られる。

01) $\downarrow Flag_3(nil) = \{<l_1, l_2, l_3> \in Red \times White \times Blue \mid nil \approx l_1 @ l_2 @ l_3\}$（サブゴール3）
02) $\downarrow Red$（*List*—導入規則）より
03) $\uparrow \ni nil$
04) $\downarrow White$（*List*—導入規則）より
05) $\uparrow \ni nil$
06) $\downarrow Blue$（*List*—導入規則）より
07) $\uparrow \ni nil$
08) $nil \approx nil @ nil @ nil$
09) $\uparrow \ni <nil, nil, nil>$：サブゴール3の結果

帰納ケースの証明：仮定 $x \in A, y \in List(A), z \in Flag_3(y)$ の下で，次を示す必要がある。$Flag_3(cons(x, y)) = \{<l_1, l_2, l_3> \in Red \times White \times Blue \mid cons(x, y) \approx l_1 @ l_2 @ l_3\}$ ここで，Σ-除去規則より，帰納ケースは次のよう求まる。

01) $\downarrow Flag_3(cons(x, y))$（サブゴール4）（$\Sigma$—除去規則）より
02) $\downarrow Flag_3(y) = \{<l_1, l_2, l_3> \in Red \times White \times Blue \mid y \approx l_1 @ l_2 @ l_3\}$（仮定）より
03) $\uparrow \ni z$
04) $[z_1 \in Red, z_2 \in White, z_3 \in Blue, y \approx z_1 @ z_2 @ z_3]$：仮定
05) $\downarrow Flag_3(cons(x, y))$（サブゴール6）

06)　↑∋ $d(z_1, z_2, z_3)$：サブゴール6の結果

07) ↑∋ $E(z, d(z_1, z_2, z_3))$：サブゴール4の結果

更に，サブゴール6は最初の仮定より $color(x) \in Color\ [x \in A]$ であるので，N_3-除去規則より，次のサブゴールが生成および証明される。

01) ↓ $Flag_3(cons(x, y))$（サブゴール6）（N_3—除去規則）より

02)　↓ $Color(= N_3)$（最初の仮定）より

03)　↑∋ $color(x)$

04)　$[color(x) = red]$

05)　↓ $Flag_3(cons(x, y)) = \{< l_1, l_2, l_3 > \in Red \times White \times Blue \mid cons(x, y) \approx l_1 @ l_2 @ l_3\}$（サブゴール7）

06)　　↓ $Red = List(\{x \in A \mid color(x) = red\})$（$List$—導入規則）より

07)　　　↓ $\{x \in A \mid color(x) = red\}$

08)　　　　↓ A（仮定）より

09)　　　　↑∋ x

10)　　　$color(x) = red$（仮定）より

11)　　↑∋ x

12)　　↓ $\{x \in A \mid color(x) = red\}$

13)　　↑∋ z_1

14)　↑∋ $cons(x, z_1)$

15)　↓ $White = List(\{x \in A \mid color(x) = white\})$（仮定）より

16)　↑∋ z_2

17)　↓ $Blue = List(\{x \in A \mid color(x) = blue\})$（仮定）より

18)　↑∋ z_3

19)　$cons(x, z_1) @ z_2 @ z_3 \approx cons(x, y)$

20)　↑∋ $< cons(x, z_1), z_2, z_3 >$：サブゴール7の結果

21)　$[color(x) = white]$

22)　↓ $Flag_3(cons(x, y))$（サブゴール8）

23)　↑∋ $< z_1, cons(x, z_2), z_3 >$：サブゴール8の結果（同様）

24)　$[color(x) = blue]$

25)　↓ $Flag_3(cons(x, y))$（サブゴール9）

26)　↑∋ $< z_1, z_2, cons(x, z_3) >$：サブゴール9の結果（同様）

27) ↑ ∋ $R_3(color(x), < cons(x, z_1), z_2, z_3 >, < z_1, cons(x, z_2), z_3 >, < z_1, z_2, cons(x, z_3) >)$：サブゴール6の結果

従って，問題の仕様 S を満たすプログラムは次で得られる。

01) ↓ $S = (\Pi\, l \in List(A))\, Flag_3(l)$（ゴール）（Π—導入規則）より

02)　$[l \in List(A)]$：仮定

03)　↓ $Flag_3(l)$（サブゴール1）（$List$—除去規則）より

04)　　↓ $List(A)$（サブゴール2）（仮定）より

05)　　↑∋ l

06)　　↓ $Flag_3(nil)$（サブゴール3）

07)　　↑∋ $< nil, nil, nil >$

08)　$[x \in A, y \in List(A), z \in Flag_3(y)]$：仮定

09)　$\downarrow Flag_3(cons(x, y))$（サブゴール 4）

10)　$\uparrow \ni E(z, R_3(color(x), <cons(x, z_1), z_2, z_3>, <z_1, cons(x, z_2), z_3>, <z_1, z_2, cons(x, z_3)>))$

11)　$\uparrow \ni LR(l, <nil, nil, nil>,$
$E(z, R_3(color(x), <cons(x, z_1), z_2, z_3>, <z_1, cons(x, z_2), z_3>, <z_1, z_2, cons(x, z_3)>)))$

12) $\uparrow \ni (\lambda l)LR(l, <nil, nil, nil>,$
$E(z, R_3(color(x), <cons(x, z_1), z_2, z_3>, <z_1, cons(x, z_2), z_3>, <z_1, z_2, cons(x, z_3)>)))$

三色の任意の並びのリストデータ l を入力すると，本プログラムは帰納的に実行し，$l = nil$ の時には 3 分割$<nil, nil, nil>$を出力する。また，$l = cons(x, y)$の時には x の色を判定 $color(x)$して，y の 3 分割結果に更に x のデータを追加する。リストの全データに対してこの操作を繰り返すことでプログラムは終了する。

4　ソフトウェア仕様の計算

ソフトウェア発展問題とは，ソフトウェア開発工程の中で要求定義やシステムに対する仕様が変更された時に，その変更を許容するようにプログラムの実現が図れる原理を確立することである。即ち，ソフトウェアの仕様が漸増的に変化すると，それに呼応してプログラムの実現が漸増的に得られる原理及び仕組みを構築するのが目標となる。ここで，仕様の発展 $S \sqsubseteq S'$が任意であれば，両者に共通部分が無くプログラム P'は新規に実現する必要がある。他方，両者に共通部分が多く含まれ，両仕様の差分を活用して P から P'が効果的に実現される時には発展問題を考えることができる。構成的プログラミング，例えば，Martin-Löf 型理論 **MTT** の中でこの問題考えるにあたり，最初に **MTT** の型および表現の間に発展関係を表す順序$\sqsubseteq$を導入する。**MTT** の型は N_nと N を基底型として，それらに型構成子 $\Pi, \Sigma, +, I, List, W$ を適用して帰納的に複雑な型が構成される。また，**MTT** の表現についても $0_n, 1_n, ..., (n-1)_n, 0, nil$ を基底表現として，コンストラクタ$(\lambda_)_$, $<\_, \_>$, $succ(\_)$, $cons(\_, \_)$, $inl(\_)$, $inr(\_)$, $sup(_, _)$およびセレクタ $R_n(c, ...)$, $R(c, \_, \_)$, $Ap(c, \_)$, $E(c, \_)$, $D(c, \_, \_)$, $J(c, \_)$, $LR(c, \_, \_)$, $T(c, \_)$ を用いて帰納的に複雑な表現が構成される。ここで，1 要素集合の型 N_1（また T と表す）$= \{t\}$ は，その非正規形表現（プログラム）$R_1(c, c_t)$ が nop 文またはプログラムの連接に対応していることより，型および表現の縮退を次で定義する。

定義 4.1 任意の表現 b においてその中に含まれる基底表現のいくつか $b_1, b_2, ..., b_n$を t で置換して得られる表現を $\tilde{b} = b[t, t, ..., t/b_1, b_2, ..., b_n]$ とする時，b は $\tilde{b}$ に縮退すると呼ぶことにする。また，任意の型 B に対しても同様に，B の中に含まれる基底型のいくつかを T で置換して得られる型 $\tilde{B} = B[T, T, ..., T/B_1, B_2, ..., B_n]$ を元の型の縮退とする。

この縮退の定義を用いて **MTT** の判定(1)－(4)に，更に(5) $A \sqsubseteq B$ および(6) $a \sqsubseteq b \in A$ の 2 つの判定を追加する。ここで，(5) A より B はより複雑な型または問題である，(6)型 A の中で a より b はより複雑な対象またはプログラムである等に解釈し，この複雑さの増大で発展関係$\sqsubseteq$を定める。例えば，$A \sqsubseteq B$ は(1) $A \equiv B$（A と B が構文的に同じ），(2) $A \equiv T$，または(3) $A = \tilde{B}$ のいずれかを表すとする。

次に，型（および表現）の順序構造の中で 2 つの基本的な演算$\oplus$と$\otimes$を帰納的に定義する。これら 2 つの演算はそれぞれ型（および表現）の合併と共通部分を取り出す操作と解釈する。更に，**MTT** 型

（および表現）の束構造の中でその2つの型（または表現）が発展関係にある時，それらの間の差分演算⊖を帰納的に定義する。この時，これらのソフトウェア仕様の計算（合併，共通部分，差分）を如何に定めるかが課題となる。

5 仕様に関する基本演算への要請

本節では，第3節で構成的プログラミングの具体例として取り挙げたDijkstraのDutch national flag問題を使い，第4節で導入したソフトウェア仕様の基本演算(合併⊕，共通部分⊗および差分⊖)の振る舞いについて検討する。元の問題（赤，白，青の並び替え）の仕様Sに対して，次の派生問題を考える。

(1) 2つのケースの二色の問題(赤，白および白，青の並び替え)を考え，それぞれの仕様をS_1およびS_2とすると，$S_1 = (\Pi\, l \in List(A_1))\{<l_1, l_2> \in Red \times White \mid l \approx l_1 @ l_2\} = (\Pi\, l \in List(A_1))Flag_2(l)$および$S_2 = (\Pi\, l \in List(A_2))\{<l_2, l_3> \in White \times Blue \mid l \approx l_2 @ l_3\} = (\Pi\, l \in List(A_2))Flag_2'(l)$となる。

(2) また，2つのケースの一色の問題(青の並び替えおよび白の並び替え)の仕様をS_3およびS_4とすると，$S_3 = (\Pi\, l \in List(A_3))\{l_3 \in Blue \mid l \approx l_3\} = (\Pi\, l \in List(A_3))Flag_1(l)$および$S_4 = (\Pi\, l \in List(A_4))\{l_2 \in White \mid l \approx l_2\} = (\Pi\, l \in List(A_4))Flag_1'(l)$となる。

この時，これらの仕様型S_1, S_2, S_3, S_4のプログラムを第3節と同様に導出すると，それぞれ以下の通りである。

01) $\downarrow S_1 = (\Pi\, l \in List(A_1))\{<l_1, l_2> \in Red \times White \mid l \approx l_1 @ l_2\}$

02) $\uparrow \ni (\lambda l)LR(l, <nil, nil>, E(z, R_2(color(x), <cons(x, z_1), z_2>, <z_1, cons(x, z_2)>)))$

03) $\downarrow S_2 = (\Pi\, l \in List(A_2))\{<l_2, l_3> \in White \times Blue \mid l \approx l_2 @ l_3\}$

04) $\uparrow \ni (\lambda l)LR(l, <nil, nil>, E(z, R_2(color(x), <cons(x, z_2), z_3>, <z_2, cons(x, z_3)>)))$

05) $\downarrow S_3 = (\Pi\, l \in List(A_3))\{l_3 \in Blue \mid l \approx l_3\}$(および$S_4$)

06) $\uparrow \ni (\lambda l)LR(l, nil, R_1(color(x), cons(x, z)))$

ここで，(1)仕様S_1とS_2，および(2)仕様SとS_3の演算について考える。

(1) 仕様S_1とS_2の演算：

$$
\begin{aligned}
S_1 \oplus S_2 &= (\Pi\, l \in List(A_1))\{<l_1, l_2> \in Red \times White \mid l \approx l_1 @ l_2\} \oplus \\
&\quad (\Pi\, l \in List(A_2))\{<l_2, l_3> \in White \times Blue \mid l \approx l_2 @ l_3\} \\
&= (\Pi\, l \in List(A_1) \oplus List(A_2))\{<l_1, l_2> \in Red \times White \mid l \approx l_1 @ l_2\} \oplus \\
&\quad \{<l_2, l_3> \in White \times Blue \mid l \approx l_2 @ l_3\} \\
&= (\Pi\, l \in List(A_1 \cup A_2))\{<l_1, l_2, t> \in Red \times White \times T \mid l \approx l_1 @ l_2 @ t\} \oplus \\
&\quad \{<t, l_2, l_3> \in T \times White \times Blue \mid l \approx t @ l_2 @ l_3\} \\
&= (\Pi\, l \in List(A))\{<l_1, l_2, t> \oplus <t, l_2, l_3> \in (Red \times White \times T) \oplus (T \times White \times Blue) \mid \\
&\quad l \approx (l_1 @ l_2 @ t) \oplus (t @ l_2 @ l_3)\} \\
&= (\Pi\, l \in List(A))\{<l_1 \oplus t, l_2 \oplus l_2, t \oplus l_3> \in (Red \oplus T) \times (White \oplus White) \\
&\quad \times (T \oplus Blue) \mid l \approx (l_1 \oplus t) @ (l_2 \oplus l_2) @ (t \oplus l_3)\} \\
&= (\Pi\, l \in List(A))\{<l_1, l_2, l_3> \in Red \times White \times Blue \mid l \approx l_1 @ l_2 @ l_3\} \\
&= S
\end{aligned}
$$

ここで，Tは第4節で述べた1要素集合の型$N_1 = \{t\}$であり，そのプログラム$R_1(c, c_t)$はnop文またはプログラムの連接に対応している。$A_1 \cup A_2$は（赤，白）と（白，青）の和集合を表す。仕様S_1

に対して T を使い，直積型 $Red\times White$ を $Red\times White\times T$ と置き換える。S_2 も同様に $T\times White\times Blue$ に置き換える。これより，$Red\times White \sqsubseteq Red\times White\times Blue$ および $White\times Blue \sqsubseteq Red\times White\times Blue$ となる。また，型について $Red\oplus T = Red$ および $White\oplus White = White$ とし，表現については $l_1\oplus t = l_1$ および $l_2\oplus l_2 = l_2$ とする。

$$
\begin{aligned}
S_1\otimes S_2 &= (\Pi\, l\in List(A_1))\{<l_1, l_2>\in Red\times White \mid l \approx l_1@l_2\}\otimes \\
&\quad (\Pi\, l\in List(A_2))\{<l_2, l_3>\in White\times Blue \mid l \approx l_2@l_3\} \\
&= (\Pi\, l\in List(A_1)\otimes List(A_2))\{<l_1, l_2>\in Red\times White \mid l \approx l_1@l_2\}\otimes \\
&\quad \{<l_2, l_3>\in White\times Blue \mid l \approx l_2@l_3\} \\
&= (\Pi\, l\in List(A_1\cap A_2))\{<l_1, l_2, t>\in Red\times White\times T \mid l \approx l_1@l_2@t\}\otimes \\
&\quad \{<t, l_2, l_3>\in T\times White\times Blue \mid l \approx t@l_2@l_3\} \\
&= (\Pi\, l\in List(A_4))\{<l_1, l_2, t>\otimes<t, l_2, l_3>\in (Red\times White\times T)\otimes(T\times White\times Blue) \mid \\
&\quad l \approx (l_1@l_2@t)\otimes(t@l_2@l_3)\} \\
&= (\Pi\, l\in List(A_4))\{<l_1\otimes t, l_2\otimes l_2, t\otimes l_3>\in (Red\otimes T)\times(White\otimes White) \\
&\quad \times(T\otimes Blue) \mid l \approx (l_1\otimes t)@(l_2\otimes l_2)@(t\otimes l_3)\} \\
&= (\Pi\, l\in List(A_4))\{<t, l_2, t>\in T\times White\times T \mid l \approx t@l_2@t\} \\
&= (\Pi\, l\in List(A_4))\{l_2\in White \mid l \approx l_2\} \\
&= S_4
\end{aligned}
$$

ここで，A_4 は A_1 と A_2 の共通集合であり，白のみの物の集合を表す。また，型について $Red\otimes T = T$ および $White\otimes White = White$ とし，表現については $l_1\otimes t = t$ および $l_2\otimes l_2 = l_2$ とする。

(2) 仕様 S と S_3 の演算：

$$
\begin{aligned}
S\ominus S_3 &= (\Pi\, l\in List(A))\{<l_1, l_2, l_3>\in Red\times White\times Blue \mid l \approx l_1@l_2@l_3\}\ominus \\
&\quad (\Pi\, l\in List(A_3))\{l_3\in Blue \mid l \approx l_3\} \\
&= (\Pi\, l\in List(A)\ominus List(A_3))\{<l_1, l_2, l_3>\in Red\times White\times Blue \mid l \approx l_1@l_2@l_3\}\ominus \\
&\quad \{l_3\in Blue \mid l \approx l_3\} \\
&= (\Pi\, l\in List(A\setminus A_3))\{<l_1, l_2, l_3>\in Red\times White\times Blue \mid l \approx l_1@l_2@l_3\}\ominus \\
&\quad \{<t, t, l_3>\in T\times T\times Blue \mid l \approx t@t@l_3\} \\
&= (\Pi\, l\in List(A_1))\{<l_1, l_2, l_3>\ominus<t, t, l_3>\in Red\times White\times Blue)\ominus(T\times T\times Blue) \mid \\
&\quad l \approx (l_1@l_2@l_3)\ominus(t@t@l_3)\} \\
&= (\Pi\, l\in List(A_1))\{<l_1\ominus t, l_2\ominus t, l_3\ominus l_3>\in (Red\ominus T)\times(White\ominus T)\times(Blue\ominus Blue) \mid \\
&\quad l \approx (l_1\ominus t)@(l_2\ominus t)@(l_3\ominus l_3)\} \\
&= (\Pi\, l\in List(A_1))\{<l_1, l_2, t>\in Red\times White\times T \mid l \approx l_1@l_2@t\} \\
&= (\Pi\, l\in List(A_1))\{<l_1, l_2>\in Red\times White \mid l \approx l_1@l_2\} \\
&= S_1
\end{aligned}
$$

ここで，$A\setminus A_3$ は A と A_3 の差集合を表し，A_1 に等しい。仕様 S_3 に対して T を使い，$Blue$ を直積型 $T\times T\times Blue$ と置き換える。これより，$Blue \sqsubseteq Red\times White\times Blue$ となる。また，型について，$Blue\ominus T = Blue$ および $Blue\ominus Blue = T$ とし，表現について，$l_1\ominus t = l_1$ および $l_3\ominus l_3 = t$ とする。更に，(1)のケースにおいて，$S_1\ominus S_2$ は $Blue \sqsubseteq T$ が成立しないので，$T\ominus Blue$ が未定義のため計算できない。

同様に，プログラムについても基本演算（合併⊕，共通部分⊗および差分⊖）の振る舞いを定める

ことができる。

参考文献

[1] L. Boerio, Extending Pruning Techniques to Polymorphic Second Order λ-Caculus, in *Proceedings of ESOP '94, Edinburgh, LNCS*, no. 788, eds. by D.Sannella, Springer-Verlag, 1994, pp. 120-134.

[2] S. Berardi and L. Boerio, *Using Subtyping in Program Optimization*, TLCA, 1995, pp. 63-77.

[3] C.A.R. Hoare, Notes on data structuring, in *Structured Programming*, eds. by E.W. Dijkstra et al., New York, Academic Press, 1972, pp. 83-174.

[4] T. Ishii, An Extension of Martin-Löf's Type Theory with an Evolution Relation, in *Proceedings of the 34th MLG meeting at Echigo-Yuzawa*, 2001, pp. 33-37.

[5] 石井忠夫，ソフトウェア仕様の差分について，新潟国際情報大学情報文化学部紀要，vol. 10 (2007), pp. 147-154.

[6] 片山卓也，ソフトウェア発展原理と研究課題，日本ソフトウェア科学会第 14 回大会論文集，(1997), pp. 297-300.

[7] T. Katayama, *A Theoretical Framework of Software Evolution*, International Workshop on Principles of Software Evolution, In Conjunction with International Conferenceon Software Engineering 1998 (ICSE98), pp. 1-5.

[8] P. Martin-Löf, Constructive Mathematics and Computer Programming, in *Logic, Methodolgy and Philosophy of science VI*, eds. by L.J. Cohen et al., North-Holland, Amsterdam, 1982, pp. 153-179.

[9] P. Martin-Löf, *Intuitionistic Type Theory*, Notes by Giovanni Sambin of a Series of Lectures Given in Padua, June 1980, Bibliopolis, Napoli, 1984.

[10] T. Bengt, P. Kent and J.P. Smith, *Programming in Martin-Löf's Type Theory, An Introduction,* Oxford Science Publications, 1990.