ソフトウェア仕様の差分について

About Differences of Software Specifications

石井 忠夫*

概要

現代社会の中でソフトウェア開発技術はコンピュータを有効に活用する手段として重要であるが、その迅速な開発を進めるにはソフトウェアの保守作業を如何にして効率良く進めるかが課題となる。この保守作業に含まれる諸々の変更要求をソフトウェアは満す必要があり、ソフトウェアが発展する過程と捉えることが可能である。本稿では、ソフトウェアが発展する原理の一般的枠組みの中に現れるソフトウェア仕様の差分と合併について構成的な型理論の中で議論する。

1 はじめに

現代の情報化社会において、ソフトウェア開発技術はコンピュータを社会の中で有効に活用する上で最も重要なものの1つである。従来からソフトウェアの開発には多くの人的資源を必要とし、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、その生産性の向上や妥当性の検証技術が強く望まれている。オブジェクト指向プログラミングや開発方法論、ソフトウェアコンポーネント技術、形式的仕様記述とその検証などはその要求に答えるものである。

ソフトウェアを開発する（またはプログラムを作る）ことは、「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ことである」と見なせる。ここで、この対象とする現実世界の振る舞いを何らかの形式的言語を用いて書き下したものが仕様であり、一貫して曖昧さのない明瞭な仕様を記述するためには論理の言葉が必要となる。この時、この仕様を満すプログラムを求めることは、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、プログラム自身は証明自身に対応すると云える。このような考え方は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[1]。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**[10, 11, 12, 15] であり、この型理論を実装した定理自動証明システムとしてNuPRL[3]、Alfaなどがある。

迅速なソフトウェアの開発を目指す時に、ソフトウェアの保守作業を如何にして効率良く進めるかが問題となる。この保守作業にはプログラム不良の修正や除去の他に、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が含まれるが、この変更を満すプログラムの手直しに多くのコストが充てられている。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[7, 8]。本稿では、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する[6]。具体的には、構成的証明において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[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、Martín-Lof 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）間での差分$\ominus$、合併$\oplus$および共通部分取り出し$\otimes$演算を定義し、発展問題の形式化を試みた。また、[9] の中で与えられた発展ドメインとその中での差分$\ominus$および合併$\oplus$演算との関連について議論した。

第2節では片山が提案したソフトウェア発展問題の一般的枠組みを、その前提条件と共に概説する。第3節では**MTT**の体系、およびその構成的プログラミングについて説明する。第4節では縮退関係を逆から見て発

*ISHII, Tadao［新潟国際情報大学 情報文化学部 情報システム学科］

展関係を定義することで、差分仕様等が扱えるようにMTT体系の拡張を試みる。第5節は纏めとして、発展ドメインとの関連について議論する。

2 ソフトウェア仕様と発展問題

ソフトウェアは現代社会において情報システムを実現する手段として広く活用されている。このようなソフトウェアの開発では、情報システムを構築し適用する対象領域の記述、対象領域内でシステムがどのように振る舞うかの記述、更にはそのシステムに望まれる振る舞いを計算機内で如何に実現するかの記述を明確にする必要がる。ここで、対象領域の記述はシステムに対する問題解決のための要求を明らかにし、システムの振る舞い記述はシステムの仕様を規定するものである。また、計算機内の記述はシステムの仕様からそれを満すプログラムの実現を与える。以上より、ソフトウェア開発の工程は概略、(1) 要求定義、(2) 仕様記述、(3) プログラムの実現に分かれる。実際にソフトウェア開発を進める時には、更に (3) はプログラムの構造を記述する設計、プログラムを作成するプログラミング、プログラムの動作を確認するテスト等の工程に細分化される。

情報システムがその機能の変更や追加または適用環境を変えると、上のソフトウェア開発の各工程が変更となり、結果としてシステムを再構築することになる。これはソフトウェアの保守と呼ばれる工程であり、ソフトウェアのライフサイクルの中で非常に多くのコストがこの工程に充てられている。このコスト削減のために従来から色々なソフトウェア工学的なアプローチが試みられてきた。その中には、オブジェクト指向概念を用いた差分プログラミングやコンポーネントなどのソフトウェア部品の利用がある。ここでは、片山の提案 [7, 8] に基づきソフトウェアの保守をソフトウェアの発展問題として捉える。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。以下で片山の提案によるソフトウェア発展問題の形式的な一般的枠組みを概説する。

今、考えている発展問題に現れうる全ての仕様の集合を$\boldsymbol{S}$、全てのプログラムの集合を$\boldsymbol{P}$とし、ある仕様$S \in \boldsymbol{S}$からプログラム$P \in \boldsymbol{P}$が導かれることを$S \vdash P$と表す。また、仕様とプログラムの集合にある順序関係$\sqsubseteq$を導入し、仕様SがS'に発展することを$S \sqsubseteq S'$と表す。この時、ソフトウェア発展問題は、仮定$S \vdash P$及び$S \sqsubseteq S'$の下で、条件$P \sqsubseteq P'$及び$S' \vdash P'$を満すプログラムP'を見出すことと定式化される。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。

次に具体的なプログラムの導出法を考えるために、構造$<\boldsymbol{S},\sqsubseteq>$及び$<\boldsymbol{P},\sqsubseteq>$は束と仮定し、更に (1) 任意の仕様$S_1, S_2 \in \boldsymbol{S}$に対して、最小上界$S_1 \sqcup S_2$及び最大下界$S_1 \sqcap S_2$がまた仕様に含まれるとする。これらはそれぞれ、2つの仕様の合併および共通仕様を表している。また、(2) 内容の無い仕様を$\bot$、矛盾した仕様を$\top$で表す。この時、仮定$S \vdash P$及び$S \sqsubseteq S'$の下で発展問題は次のように解かれる。

(1) 仮定$S \sqsubseteq S'$より、$S' = S \sqcup \Delta S$を満す差分仕様ΔSを構成する。この為には、構造$<\boldsymbol{S},\sqsubseteq>$に対して、$S \sqsubseteq S'$ならば$S' = S \sqcup \Delta S$かつ$S \sqcap \Delta S = \bot$を満す$\Delta S$の分離可能性を要請する。

(2) 差分仕様ΔSからその差分プログラム$\Delta S \vdash \Delta P$を実現する。この為には、任意の仕様$S \in \boldsymbol{S}$に対して、プログラム$S \vdash P$の実現可能性を要請する。

(3) 最後に仕様S'を満すプログラム$P' = P \sqcup \Delta P$を実現する。この為には、プログラム実現の単調性：$S_1 \vdash P_1$かつ$S_2 \vdash P_2$ならば$S_1 \sqcup S_2 \vdash P_1 \sqcup P_2$を要請する。

更に、別の発展形態として仕様SとS'が共通の仕様Tから発展した場合を考えることができる。この時、仮定$S \vdash P$及び$T \sqsubseteq S$かつ$T \sqsubseteq S'$の下で発展問題は次のように解かれる。

(4) 仮定$T \sqsubseteq S$より、$S = T \sqcup \Delta S$を満す差分仕様ΔSを構成する。

(5) 差分仕様ΔSからその差分プログラム$\Delta S \vdash \Delta P$を実現する。

(6) 仕様Tを満すプログラム$T \vdash Q$を実現する。この為には、$P = Q \sqcup \Delta P$からプログラムQの縮退可能性を要請する。

(7) 後は仮定$T \vdash Q$及び$T \sqsubseteq S'$に対して、上の手順(1)－(3)を実行すると、仕様S'のプログラムが求まる。

以上は発展問題の一般的枠組みであり、その中で仮定している事柄：(1) $<\mathcal{S}, \sqsubseteq>$ が束構造、(2) 差分仕様の分離可能性、(3) 任意仕様の実現可能性、(4) プログラム実現$\vdash$の単調性、(5) プログラムの縮退可能性等が現実に即しているかどうかについては更に検討を要する。また、仕様SがS'に発展する$S \sqsubseteq S'$とは、(1) S'はSより扱えるデータやケースが多い（機能拡張)、(2) S'はSより機能やデータに関してより具体的かつ詳細な内容を規定している（詳細化/具体化）と解釈する。

3 構成的型理論

計算機内で情報システムの仕様を満すプログラムを実現するのがプログラム言語の役割りであるが、その中でデータ型の概念 [5, 16] は高級言語が初めて提案されたFORTRANの時代から既に存在している。データは問題とする対象領域内に存在する現実の物や概念を抽象化した計算機内での表現であり、データ型の導入により各種型に属する定数や変数の集合を分離決定し、また型宣言によりそれらの値や計算の仕方が決定できる。これにより、データ構造の設計やプログラムの整合性の判定等に利用できる。型に関する性質を論じる形式的体系として現在まで色々な体系が提案されてきたが [13, 14]、この型の解釈により数学の形式的証明やプログラムの合成/検証などを計算機上で行える [1]。次に述べる構成的型理論は、型を用いて論理記号の直観主義的な解釈を与えることで、構成的数学の展開および計算機上で仕様からプログラムを合成および検証する仕方を与える。

構成的型理論にはMartin-Löfの型理論 **MTT**[10, 11] やCoquandの **CC**[4] 等が知られているが、ここでは **MTT** について概略を説明する。**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。例えば、自然数の型の中で、0, 1, 2,... が正規形表現、$2 + 2$, 2×2, 2^2 などが非正規形表現となる。**MTT** は実際に表3.1に示す型と表現から構成される。N_n ($n = 0, 1, ...$) は有限集合の型（列挙型）を表す。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。Nは自然数の型を表し、正規形表現として0, 1, 2,... を持つ。$succ(n)$はnの次の数を返すコンストラクタである。非正規形表現$R(c, d, e(x, y))$は最初に$c \in N$を評価し、その値が0の時にはdを評価し、また、値が$succ(n)$の時には$e[n, R(n, d, e(x, y))/x, y]$を評価する。これにより原始帰納関数が定義できる。また、自然数の和$a + b$と積$a \times b$はそれぞれ$R(b, a, succ(y))$と$R(b, 0, y + a)$で表せる。$(\Pi x \in A)B(x)$は依存積の型であり一般的関数$A \rightarrow B(x)[x \in A]$を表し、正規形表現$(\lambda x)b$は関数抽象、また、非正規形表現$Ap(c, a)$は関数適用を表す。最初に$c \in (\Pi x \in A)B(x)$を評価し、その値が$(\lambda x)b$の時には$b[a/x]$を$Ap(c, a)$の値とする。$(\Sigma x \in A)B(x)$は依存和の型であり一般的レコード$A \times B(x)[x \in A]$を表し、正規形表現$< a, b >$は順序対、また、非正規形表現$E(c, d(x, y))$は分離関数を表す。最初に$c \in (\Sigma x \in A)B(x)$を評価し、その値が$< a, b >$の時には$d[a, b/x, y]$を評価する。順序対の第1および第2要素を取り出す射影関数はそれぞれ$Fst(c) = E(c, x)$, $Snd(c) = E(c, y)$で表せる。$A + B$は直和の型であり、正規表現として$inl(a)$, $inr(b)$を持つ。inlとinrは左射影および右射影を表すコンストラクタであり、例えば、要素aが直和の左成分Aに属すことを示している。非正

規表現 $D(c, d(x), e(e))$ は最初に $c \in A + B$ を評価し、その値が $inl(a)$ または $inr(b)$ に応じてそれぞれ $d[a/x]$ または $e[b/y]$ を評価する。$I(A, a, b)$ は判定 $a = b \in A$ を表す型である。また、$(Wx \in A)B(x)$ は Wellordering の型である。

表 3.1: **MTT** の型と表現

型形式	正規形表現	非正規形表現
N_n $(n = 0, 1,...)$	$0_n, 1_n,..., (n\text{-}1)_n$	$R_n(c, c_0, c_1, ..., c_{n-1})$
N	$0, succ(n)$	$R(c, d, e(x, y))$
$(\Pi x \in A)B(x)$	$(\lambda x)b$	$Ap(c, a)$
$(\Sigma x \in A)B(x)$	$< a, b >$	$E(c, d(x, y))$
$A + B$	$inl(a), inr(b)$	$D(c, d(x), e(y))$
$I(A, a, b)$	r	$J(c, d)$
$(Wx \in A)B(x)$	$sup(a, b)$	$T(c, d(x, y, z))$

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。**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、依存積型 Π の推論規則は次で与えられる。

(1) Π-formation :

$$\frac{A \text{ type} \quad \begin{matrix}(x \in A)\\ B(x) \text{ type}\end{matrix}}{(\Pi x \in A)B(x) \text{ type}} \qquad \frac{A = C \quad \begin{matrix}(x \in A)\\ B(x) = D(x)\end{matrix}}{(\Pi x \in A)B(x) = (\Pi x \in C)D(x)}$$

(2) Π-introduction :

$$\frac{A \text{ type} \quad \begin{matrix}(x \in A)\\ b(x) \in B(x)\end{matrix}}{(\lambda x)b(x) \in (\Pi x \in A)B(x)} \qquad \frac{A \text{ type} \quad \begin{matrix}(x \in A)\\ b(x) = d(x) \in B(x)\end{matrix}}{(\lambda x)b(x) = (\lambda x)d(x) \in (\Pi x \in A)B(x)}$$

(3) Π-elimination :

$$\frac{c \in (\Pi x \in A)B(x) \quad a \in A}{Ap(c, a) \in B(a)} \qquad \frac{c = d \in (\Pi x \in A)B(x) \quad a = b \in A}{Ap(c, a) = Ap(d, b) \in B(a)}$$

(4) Π-equality :

$$\frac{a \in A \quad \begin{matrix}(x \in A)\\ b(x) \in B(x)\end{matrix}}{Ap((\lambda x)b(x), a) = b(a) \in B(a)} \qquad \frac{\begin{matrix}(x \in A)\\ c \in (\Pi x \in A)B(x)\end{matrix}}{c = (\lambda x)Ap(c, x) \in (\Pi x \in A)B(x)}$$

ここで、（1）の形成規則は依存積型を構成する為の必要十分条件を定める。（2）の導入規則は依存積型の正規形表現が何かを定める。（3）の除去規則は依存積型の対象を定義域とする関数を定める。（4）の等号規則は依存積型の対象間の等号関係を定める。**MTT** の中で論理式 $A \wedge B$, $A \vee B$, $A \supset B$, $(\forall x \in A)B(x)$, $(\exists x \in A)B(x)$ はそれぞれ $(\Sigma x \in A)B$, $A + B$, $(\Pi x \in A)B$, $(\Pi x \in A)B(x)$, $(\Sigma x \in A)B(x)$ と見なすことで型を用いて論理記号の構成的な解釈が得られる。例えば、全称命題 $(\forall x \in A)B(x)$ は、型 A の任意の対象を依存型 $B(x)[x \in A]$ の対象に写す関数として $(\Pi x \in A)B(x)$ で解釈できる。プログラムの仕様は一般的に $(\forall x \in A)(\exists y \in B(x))C(x, y)$ と表せるので型では $(\Pi x \in A)(\Sigma y \in B(x))C(x, y)$ と解釈できる。今、この仕様型の対象 e が **MTT** 体系の中で導出できたとすると、次の推論により仕様を満すプログラム $f = (\lambda x)Fst(Ap(e, x))$ が求まる。

$$\frac{A \text{ type} \quad \dfrac{\dfrac{e \in (\Pi x \in A)(\Sigma y \in B(x))C(x, y) \quad (x \in A)}{Ap(e, x) \in (\Sigma y \in B(x))C(x, y)}\,(\Pi - elim)}{Fst(Ap(e, x)) \in B(x)}\,(left - proj)}{(\lambda x)Fst(Ap(e, x)) \in (\Pi x \in A)B(x)}\,(\Pi - intro)$$

4 差分仕様の形式化

Martin-Löf の型理論 **MTT** ではその構成的な解釈により、型とその対象は論理式とその（具体的な構成法の示された）証明に対応する。従って、ソフトウェアの仕様を述語論理式で表現し、その型の対象を **MTT** 体系の中で証明することにより、第3節の終りで述べた導出に従い仕様を満す実行プログラムが抽出できる。ここでは、ソフトウェア仕様記述と実行プログラムの導出を共に表現および実行可能な **MTT** 体系の中で、第2節で述べたソフトウェア発展問題の形式化について議論する。

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 $\Pi, \Sigma, +, I, W$ 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 $0_n, 1_n, ..., (n\text{-}1)_n$ および $0, succ(n)$ を基底表現として、コンストラクタ $(\lambda\_)\_, <\_, \_>, inl(\_), inr(\_), sup(\_, \_)$ およびセレクタ $R_n(c,...), R(c, \_, \_), Ap(c, \_), E(c, \_), D(c, \_, \_), J(c, \_), T(c, \_)$ 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を $\tilde{B} = B[T, T, ..., T / B_1, B_2, ..., B_n]$ 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。この2つの判定を用いて各型の発展関係を推論規則で定める。例えば依存積型 Π の推論規則は、第3節で示した（1）－（4）に次を追加する。

(5) Π-order :

$$\frac{A \sqsubseteq C \quad \begin{matrix}(x_1 \in A, x_2 \in C) \\ B(x_1) \sqsubseteq D(x_2)\end{matrix}}{(\Pi x_1 \in A)B(x_1) \sqsubseteq (\Pi x_2 \in C)D(x_2)}$$

$$\frac{A \text{ type} \quad \begin{matrix}(x \in A) \\ b(x) \sqsubseteq d(x) \in B(x)\end{matrix}}{(\lambda x)b(x) \sqsubseteq (\lambda x)d(x) \in (\Pi x \in A)B(x)} \qquad \frac{c \sqsubseteq d \in (\Pi x \in A)B(x) \quad a \sqsubseteq b \in A}{Ap(c, a) \sqsubseteq Ap(d, b) \in B(a)}$$

ここで、上の順序規則は依存積型の型と正規形および非正規形表現の発展関係 $\sqsubseteq$ を定める。また、基底型について $N_k \sqsubseteq N_l (k \leq l) \Longleftrightarrow N_k \subseteq N_l, N \sqsubseteq N$ および任意の型 A に対して $T \sqsubseteq A$ とする。基底表現については $m_k \sqsubseteq m_l \in N_k (k \leq l), 0 \sqsubseteq 0, succ(a) \sqsubseteq succ(b) \Longleftrightarrow a \leq b$ および任意の表現 b に対して $t \sqsubseteq b$ とする。この時、**MTT** の型の集まり（および表現の集まり）の中で発展関係 $\sqsubseteq$ は半順序となる。

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。基底型に対して、（1）$N_k \oplus N_l = N_r$（但し、$r = |N_k \cup N_l|$）および $N_k \otimes N_l = N_s$（但し、$s = |N_k \cap N_l|$）、（2）$N \oplus N = N$ および $N \otimes N = N$ とする。各型構成子に対しては、（3）$(\Pi x_1 \in A)B(x_1) \oplus (\Pi x_2 \in C)D(x_2) = (\Pi x \in A \oplus C)B(x) \oplus D(x)$ および $(\Pi x_1 \in A)B(x_1) \otimes (\Pi x_2 \in C)D(x_2) = (\Pi x \in A \otimes C)B(x) \otimes D(x)$ などとし、また、（4）任意の型 A, B に対し $A \oplus B = A + B$ および $A \otimes B = T$ と定める。基底表現については、（1）$m_k = m_l (k \leq l)$ の時、$m_k \oplus m_l = m_k \otimes m_l = m_k$、また $m_k \neq m_l (k \leq l)$ の時、$m_k \oplus m_l = inl(m_k)$ または $inr(m_l)$ および $m_k \otimes m_l = t$、（2）$0 \oplus 0 = 0 \otimes 0 = 0, succ(a) \oplus succ(b) = succ(a + b)$ および $succ(a) \otimes succ(b) = succ(c)$（但し、$c$ は a と b のどちらか大きくない方）とする。各コンストラクタおよびセレクタに対しては、（3）$(\lambda x_1)b(x_1) \oplus (\lambda x_2)d(x_2) = (\lambda x)(b(x) \oplus d(x))$ および $(\lambda x_1)b(x_1) \otimes (\lambda x_2)d(x_2) = (\lambda x)(b(x) \otimes d(x))$、また $Ap(c, a) \oplus Ap(d, b) = Ap(c \oplus b, a \oplus b)$ および $Ap(c, a) \otimes Ap(d, b) = Ap(c \otimes d, a \otimes b)$ などと

し、また、(4) 任意の表現 a, b に対しては $a \oplus b = inl(a)$ または $inr(b)$ および $a \otimes b = t$ と定める。この時、**MTT** の型の集まり（および表現の集まり）の中で2つの演算 $\oplus$ と $\otimes$ はそれぞれ最小上界と最大下界となり、型（および表現）の順序構造は束となる。

定理 4.2 任意の表現 $a \in A, b \in B$ に対して、$c \sqsubseteq a \in C$ かつ $c \sqsubseteq b \in C$ を満す $c \in C$ が存在する時には $a \oplus b \in A \oplus B$、また、$a \sqsubseteq c \in A$ かつ $b \sqsubseteq c \in B$ を満す $c \in C$ が存在する時には $a \otimes b \in A \otimes B$ となる。

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$\ominus$を帰納的に定義する。基底型に対して、(1) $N_k \sqsubseteq N_l (k \leq l)$ の時、$N_l \ominus N_k = N_v$（但し、$v = |N_l \setminus N_k|$）とし、また (2) $N \sqsubseteq N$ の時、$N \ominus N = N_0$ とする。各型構成子に対しては、(3) $(\Pi x_1 \in A)B(x_1) \sqsubseteq (\Pi x_2 \in C)D(x_2)$ の時、$(\Pi x_2 \in C)D(x_2) \ominus (\Pi x_1 \in A)B(x_1) = (\Pi x \in C \ominus A)B(x) \ominus D(x)$ などとし、また、(4) 順序が付かない任意の型 A, B に対しては $A \ominus B = N_0$ とする。基底表現については、(1) $m_k = m_l (k \leq l)$ の時、$m_l \ominus m_k = t$、また $m_k \neq m_l$ $(k \leq l)$ の時、$m_l \oplus m_k = \varepsilon$（空要素を表す）、(2) $0 \ominus 0 = \varepsilon$, $succ(a) \sqsubseteq succ(b)$ の時、$succ(b) \ominus succ(a) = succ(b - a)$ とする。各コンストラクタおよびセレクタに対しては、(3) $(\lambda x_2)b(x_1) \sqsubseteq (\lambda x_2)d(x_2) \in (\Pi x_1 \in A)B(x_1)$ の時、$(\lambda x_1)d(x_2) \ominus (\lambda x_1)b(x_1) = (\lambda x)d(x) \ominus b(x)$（但し、$x \in C \ominus A$）および $Ap(c, a) \sqsubseteq Ap(d, b) \in B(a)$ の時、$Ap(d, b) \ominus Ap(c, a) = Ap(d \ominus c, b \ominus a)$ などとし、また、(4) 順序が付かない任意の表現 a, b に対しては $a \ominus b = \varepsilon$ と定める。

定理 4.3 MTT の型および表現の集合をそれぞれ $\mathcal{T}, \mathcal{E}$ とすると次が得られる。
(1) 任意の型 $A, B \in \mathcal{T}$ に対して、$A \sqsubseteq B$ ならば $A \oplus (B \ominus A) = B$ かつ $A \otimes (B \ominus A) = N_0$ となる。(2) 同様に、任意の表現 $a, b \in \mathcal{E}$ に対して、$a \sqsubseteq b \in A$ かつ $b \in B$ ならば $a \oplus (b \ominus a) = b \in B$ かつ $a \otimes (b \ominus a) = \varepsilon \in N_0$ となる。(3) 任意の表現 $a, b \in \mathcal{E}$ に対して、$a \sqsubseteq b \in A$ かつ $b \in B$ ならば $b \ominus a \in B \ominus A$ となる。

5 議論

片山は論文 [9] の中で発展ドメインを定義し、その上でタグ集合を使用した仕様の差分 $\ominus$ と合併 $\oplus$ 演算を提案している。以下では、この論文との関連について述べる。発展関係を導入した仕様（およびプログラム）の構造 $<\mathcal{S}, \sqsubseteq>$ $(<\mathcal{P}, \sqsubseteq>)$ が次の条件を満す時、発展ドメインと定める。(1) $\sqsubseteq$ が半順序となる。(2) 任意の仕様 $S, S' \in \mathcal{S}$ に対して、これらの下界集合が存在する。(3) $S, S' \in \mathcal{S}$ に対して、$S \sqsubseteq S'$ ならば $S' = S \oplus (S' \ominus S)$ を満す2つの演算 : $\ominus: \mathcal{S} \times \mathcal{S} \to (\mathcal{A}_S \to \mathcal{S})$ および $\oplus: \mathcal{S} \times (\mathcal{A}_S \to \mathcal{S}) \to \mathcal{S}$ が定義されており、$S \sqsubseteq S'$ ならば $S' \ominus S = \{(a_i, S_i) | 1 \leq i \leq n, a_i \in \mathcal{A}_S, S_i \in \mathcal{S}\}$ および $S \oplus (S' \ominus S) = S \oplus \{(a_i, S_i) | 1 \leq i \leq n, a_i \in \mathcal{A}_S, S_i \in \mathcal{S}\} = S'$ となる。即ち、$<\mathcal{S}, \sqsubseteq>$ の $\sqsubseteq$ が半順序かつ下に有界の時、タグ集合 $\mathcal{A}_S$ を用いて $S \sqsubseteq S'$ ならば $S' \ominus S$ が求まり、かつ $S \oplus (S' \ominus S) = S'$ を満す2つの演算 $\ominus, \oplus$ を定義した構造 $<\mathcal{S}, \ominus, \oplus>$ が発展ドメインである。

発展ドメインの例として λ −発展ドメインについて考える。λ −発展ドメイン $\mathcal{D}_\lambda = <\Lambda, \sqsubseteq, \ominus, \oplus>$ とは、関数適用および関数抽象からなる λ 項の集合上で $\sqsubseteq, \ominus, \oplus$ を定義したものである。今、任意の λ 項 $t_1, t_2 \in \Lambda$ に対して、$t_1 \sqsubseteq t_2$ を (1) $t_1 = t_2$ または $\exists s_1, s_2, ..., s_k \in \Lambda$, $t_2 = t_1 s_1 s_2 \cdots s_k = (\cdots (t_1 s_1) s_2 \cdots) s_k$ とする。この時、$<\Lambda, \sqsubseteq>$ は半順序かつ下に有界 $(I = (\lambda x)x$ が存在$)$ となり、また、$\ominus$と$\oplus$は次で定まる。$t_1 \sqsubseteq t_2$ に対して、(1) $t_1 = t_2 = t$ の時は、$t_2 \ominus t_1 = t \ominus t = \{\}$ および $t_1 \oplus (t_2 \ominus t_1) = t_1 \oplus \{\} = t \oplus \{\} = t$、(2) $\exists s_1, s_2, ..., s_k \in \Lambda$, $t_2 = t_1 s_1 s_2 \cdots s_k$ の時は、$t_2 \ominus t_1 = \{(1, s_1), (2, s_2), ..., (k, s_k)\}$ および $t_1 \oplus (t_2 \ominus t_1) = t_1 \oplus \{(1, s_1), (2, s_2), ..., (k, s_k)\} = t_1 s_1 s_2 \cdots s_k = t_2$ となる。ここで、タグ集合は $\mathcal{A}_S = N$ であり、λ 項 $s_1, s_2, ..., s_k$ を適用する場所情報を自然数として保持し、差分と合併演算の中で活用している。

次に、この λ −発展ドメインを **MTT** の中で考える。今、$S_2 = (\Pi x_1 \in A_1)(\Pi x_2 \in A_2) \cdots (\Pi x_k \in A_k)B(x_1, x_2, ..., x_k)$ および $S_1 = (\Pi x_1 \in T)(\Pi x_2 \in T) \cdots (\Pi x_k \in T)B(x_1, x_2, ..., x_k)$ とする。この時、$T \sqsubseteq A_i (1 \leq i \leq k)$ より、$S_1 \sqsubseteq S_2$ となる。今、$(\lambda x_1)(\lambda x_2) \cdots (\lambda x_k)b(x_1, x_2, ..., x_k) (= e$ と表す$) \in S_2$ とすると、$Ap(... Ap(Ap(e, a_1), a_2), ..., a_k) (= b_2$ と表す$) \in B(a_1, a_2, ..., a_k)$ となり、また $Ap(... Ap(Ap(e, t), t), ..., t) (= b_1$ と表す$) \in B(t, t, ..., t)$ となる。この時、$t \sqsubseteq a_i (1$

$\leq i \leq k$) より、$b_1 \sqsubseteq b_2 \in B(t, t, ..., t)$ となる。従って、プログラムの差分は $b_2 \ominus b_1 = Ap(... Ap(Ap(e, a_1), a_2), ..., a_k) \ominus Ap(... Ap(Ap(e, t), t), ..., t) = Ap(... Ap(Ap(e \ominus e, a_1 \ominus t), a_2 \ominus t), ..., a_k \ominus t) = Ap(... Ap(Ap(\varepsilon, a_1), a_2), ..., a_k)$ となる。また、プログラムの合併は $b_1 \oplus (b_2 \ominus b_1) = Ap(... Ap(Ap(e, t), t), ..., t) \oplus Ap(... Ap(Ap(\varepsilon, a_1), a_2), ..., a_k) = Ap(... Ap(Ap(e \oplus \varepsilon, t \oplus a_1), t \oplus a_2), ..., t \oplus a_k) = Ap(... Ap(Ap(e, a_1), a_2), ..., a_k) = b_2$ が得られる。ここで $t \in T$ の意味を見る為に、$b(x) = (\lambda x)(x + 2) \in (\Pi x \in N)N$ を考える。$b(3) = 3 + 2 \in N$ および $b(t) = t + 2 \in N$ となるが、t は標準的な等号を持つ自然数の型 N においてその中の全ての数が一致する時の対象を表しており、$t + 2$ より $3 + 2$ では3が明示的に区別して指定されているので $t \sqsubseteq 3 \in N$ から $t + 2 \sqsubseteq 3 + 2 \in N$ が云える。先の演算におけるタグ集合の情報は、ここでは表現（または型）の形成規則の中に埋め込まれていると云える。また、先の λ 項の順序付けは項の抽象度（関数適用の深度）で定義しているが、ここでは同じ抽象度の中で順序付けを与えている、即ち、$t_1 \sqsubseteq t_2$ を t_1 と $t_2 = t_1 s_1 s_2 \cdots s_k$ の間の関係と見るのではなく、$t_1 = t'_1 tt \cdots t$ と $t_2 = t'_1 s_1 s_2 \cdots s_k$ の間の関係と見る点で異なっている。

更に、ある2つの自然数 $a, b \in N$ に対してその和と積の存在を考える。2つの数の和が存在することは $(\exists z \in N)(z = a + B)$ と表せるので、型で表現すると $S_1 = (\Sigma z \in N)I(N, z, R(b, a, succ(y)))$ となる。また、同様にして2つの数の積が存在することは $S_2 = (\Sigma z \in N)I(N, z, R(b, 0, succ^a(y)))$ となる。但し、$succ^a(y) = \overbrace{succ(succ \cdots succ}^{a\text{回}}(y) \cdots))$ とする。この時、共通仕様は $S_1 \otimes S_2 = (\Sigma z \in N \otimes N)I(N \otimes N, z \otimes z, R(b \otimes b, succ^a(0) \otimes 0, succ(y) \otimes succ^a(y))) = (\Sigma z \in N)I(N, z, R(b, 0, succ(y)))$、また、差分仕様は $S_1 \ominus (S_1 \otimes S_2) = (\Sigma z \in N \ominus N)I(N \ominus N, z \ominus z, R(b \ominus b, succ^a(0) \ominus 0, succ(y) \ominus succ(y))) = (\Sigma z \in N_0)I(N_0, \varepsilon, R(\varepsilon, R(\varepsilon, succ^a(0), \varepsilon))$ および $S_2 \ominus (S_1 \otimes S_2) = (\Sigma z \in N_0)I(N_0, \varepsilon, R(\varepsilon, \varepsilon, succ^a(0)))$ となる。ここで、共通仕様の中の $R(b, 0, succ(y))$ は和と積の共通なプログラムを与え、また、各差分仕様の中の $R(\varepsilon, succ^a(0), \varepsilon)$ および $R(\varepsilon, \varepsilon, succ^a(0))$ は和と積の共通プログラムからの差分を特徴付けている。

参考文献

[1] M. Beeson, *Foundations of Constructive Mathematics,* Springer-Verlag, 1985.

[2] 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.

[3] R.L. Constable et al., *Implementing Mathematics with the NuPRL Proof Development System,* Prentice-Hall, Englewood CliRs, NJ, 1986.

[4] T. Coquand and G. Huet, *The Calculus of Constructions,* Information and Computation,vol.76(1988), pp.95-120.

[5] 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.

[6] 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.

[7] 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.

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

[9] 片山卓也, 発展ドメイン：ソフトウェア発展のための理論的枠組み, **コンピュータソフトウェア**, vol.21, Nr.3(2004), pp.11-21.

[10] 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.

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

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

[13] 桜井貴文, 直観主義論理と型理論, **情報処理**, vol.30, Nr.6(1989), pp.626-634.
[14] 龍田真, **型理論**, 近代科学社, 1992.
[15] S. Thompson, *Type Theory and Functional Programming,* Addison-Wesley Publishers, 1991.
[16] N. Wirth, 浦昭二, 國府方久史共訳, **アルゴリズムとデータ構造**, 共立出版, 1986.