構成的プログラミング言語 Maya の型理論的意味

〒305 つくば市梅園 1-1-4
電子技術総合研究所
木下 佳樹*

1992 年 1 月 10 日

プログラム, プログラムの仕様, プログラムが仕様を満足することの証明をすべて記述することができ, しかも大規模プログラミングをサポートする機能 (パラメータ付モジュール, 継承, 多相的プログラムなど) を備えた計算機言語 Maya を設計している。本発表では Martin-Löf による構成的集合論を用いて, Maya の意味記述を行なう。特にモジュール機能に焦点をあてて議論することにする。

1 はじめに

仕様とその実現の関係を正確に考慮しながら, プログラムの部品の交換をキッチリ行なうことを考えよう。部品をもらう側では, それが何をするものであるか (仕様) について, 必要最小限のことを知るだけで, 実現の詳細についてはできるだけ知らずに済ませたい。プログラム自身は, それが行なうことがらについての最も詳しい仕様の記述ではあるが, そもそも他人の作ったプログラムを使おうという場合はそこまで詳しい仕様記述を必要とすることはない。実はもっと粗い記述だけで十分だからこそ, 自分でプログラムを書かずに他人の書いたものを拝借しようとするのであろう。

実際には, 必要な部品に対して指定する条件は場合によって粗いものから細かいものまでさまざまな形で与えられる。例えば, アルゴリズムは何でもいいから整列化プログラムが欲しいということがあるだろう。別の場合にはもう少し詳しい条件が必要で, 例えば与えられるデータが殆ど整列済の場合が多いのでクイックソートでなくヒープソートによる整列化プログラムが欲しいということもあるかも知れない。

このような状況に対応するには, プログラムの記述言語の他に, 仕様の記述言語を設定し, 仕様記述によって必要な部品を区別するという枠組が適当だと考える。つまり,

- プログラム
- 仕様記述

の二つを別のものとしてたくわえておき, さらにどのプログラムがどの仕様記述を満足するかという関係がわかるようになっておればよいというわけである。でも

(*) プログラムが仕様を満足する

ということを示すのは一般の定理証明と同等であって, 機械に任せっきりにするわけにはいかないばかりか人間にとっても大変な仕事である。そこで, プログラム, 仕様記述の他に

- プログラムが仕様を満足することの証明

も (誰が作るのかは問題だが) 一度作ってたくわえておけば, 必要な時にはその証明をなぞるだけで (*) を確かめることができる。計算機には証明を作りだすことは難しくとも与えられた証明の正しさを検査することは簡単だから, プログラム, 仕様記述, 証明をすべて計算機にたくわえて (*) の保証を計算機にさせることも可能である。計算機による管理を行なうには, これらのものをすべて形式的言語で記述することが必要で, 英語, 日本語などの自然言語による仕様記述では (少なくとも現在は) 計算機による処理に適しない。

さて, Modula-2[9], Standard ML[5][6], OBJ3[3] など, 既存のいわゆるモジュラーなプログラミング言語にはいわゆるモジュール・インターフェイス (今あげた三つの言語ではそれぞれ definition module, signature, theory などと呼ばれる) の記述機能がある。これを必要な部品の仕様記述とすることができないだろうか。実際, これらの言語によるプログラミングでは, モジュール・インターフェイスを仕様記述として使うことがある程度まで意識されているように思われる。けれども少なくとも以上に挙げた三つの言語では, 以上に記したような我々のプログラム部品管理を完全に行なうことはできない。これらの言語のモジュール・インターフェイスの記述力が限られているためである。

例えば, Modula-2 の definition module では implementation module で定義されるべきデータ型および変数や定数などの値とその型を指定するだけで, 定義されたものどおしの間にどのような関係があるのかは formal に記述できず, 仕様記述の肝心な部分は, もっぱら

*Type theoretical semantics of the constructive programming language Maya
Electrotechnical Laboratory
KINOSHITA Yoshiki
e-mail: yoshiki@etl.go.jp

注釈の中に英語や日本語で記述する他なく, 計算機で処理するわけにはいかない。Standard ML には structure 共有という形式的仕様記述機能はあるものの, これは記述力が非常に弱いので事情は本質的に Modula-2 の場合と同じである。OBJ3 では, theory の中に等式による形式的仕様記述を書き込むことができるので, 前二つよりは余程いいのだが, 証明の形式的記述がないので, (*) を示すには処理系が常に定理証明を行なわなければならない。

そこで, プログラム, プログラムの仕様, プログラムが仕様を満足することの証明をすべて記述することができ, しかも大規模プログラミングをサポートする機能 (パラメータ付モジュール, 継承, 多相的プログラムなど) を備えた言語 Maya を設計している。仕様や証明を取り扱う言語なので, Maya 自身にまず数学的な意味を与える必要がある。このための土台として, Martin-Löf による構成的集合論 (以下 Martin-Löf 集合論と呼ぶ) を用いる。この体系は types as formulae の原理に沿って, 集合 (型) とその要素, 命題とその証明などを形式的に取り扱うものである。我々は types as formulae というよりも, types as specification の原理を用いる。つまり, 仕様を集合, 仕様の実現を集合の要素に対応させて考える。ともあれ, Maya は構成的集合論に基づく言語なので, 構成的プログラミング言語であり, また, 大規模プログラミングのための機能を持つので, モジュラーな言語でもある。

2 節では Maya の意味記述に用いる Martin-Löf 集合論を紹介する。次に 3 節で INF 記法による Maya の抽象構文とその直感的意味を紹介する。3.3 節で Martin-Löf 集合論による Maya の意味記述を行なう。Maya のトップ・レベルの抽象構文木を評価することが Martin-Löf 集合論の集合や要素に対するどのような操作に対応するかを示す。

2 Martin-Löf 集合論

Maya の意味は, Martin-Löf 集合論の中で記述される。この体系は [7] で概略が紹介されているが, ここでも簡単に紹介する。

Martin-Löf 集合論は二段階の体系から構成される。はじめに**式の体系** (theory of expression) が与えられ, これによって集合論が対象とする式とその間の同等性が定められる。実は式の体系は唯一つの型定数 0 を持つような型付き $\lambda_{\beta\eta}$ 計算の体系であり, 式の他の式への作用 (関数適用) に関することがすべて式の体系で定められる。また, 簡単なマクロ定義も式の体系で定められる。

式の体系の上に **Martin-Löf 集合論**が構築される。Martin-Löf 集合論では集合もその要素もすべて, 式の体系で定められている式であって, 抽象的なものではない。集合 (あるいは型) とその要素についての基本的な関係が導入され, また, いわゆる Curry-Howard の対応によって命題とその証明が定義される。命題は集合として, 証明は要素としてそれぞれ定義される。自然数や直積など, 基本的な集合は定数とそれに関する導出規則として導入されるが, Martin-Löf 集合論は開かれた体系であり, 新しく集合の定義を加えていくこともできる。

1. 0 はアリティである。
2. $\alpha_1, \ldots, \alpha_n$ $(n \geq 2)$ がアリティであれば, $(\alpha_1 \otimes \cdots \otimes \alpha_n)$ もアリティである。
3. α, β がアリティであれば $\alpha \twoheadrightarrow \beta$ もアリティである。
4. 以上の規則を有限回適用してアリティとわかるものだけがアリティである。

表 1: アリティの定義

2.1 式の体系

唯一つの型定数 0 を持ち, 積を持つ型付き $\lambda_{\beta\eta}$ 計算の体系における項が式である。またこの体系における型はアリティと呼ばれる。後に集合論で出てくる関数集合や直積集合と区別するため, アリティの記述には通常のような $\rightarrow$ や $\times$ が用いられず, かわりに $\twoheadrightarrow$ や $\otimes$ が用いられる。そこで, アリティは表 1のように帰納的に定義される。

アリティは式が他の式にどのように作用 (関数適用の意味で) するかを示すものとして用いる。アリティ 0 を持つ式はもう他の式に作用することができないので**飽和**している。アリティ $0 \twoheadrightarrow (0 \twoheadrightarrow 0)$ を持つ式は, 飽和している式に作用して, アリティ $0 \twoheadrightarrow 0$ を持つ式を生むなどという具合である。

アリティ α を持つ式が, 通常の積を持つ型付き $\lambda_{\beta\eta}$ 計算の体系における「型 α を持つ項」の定義と同様に定義される。繁雑になるのでここでは厳格な帰納的定義は行わない (定義は [7] 参照) が, 以下に記法をざっと説明する。我々は, 定数として何が定められているのかを, その時々に指定していくことにする。また, 変数としては, ゴチック (このような**字体** like this) の文字をつないだもののうち定数以外のものを用いる。関数適用, 関数抽象としてそれぞれ, $d(a)$, $((x)b)$ の形を用いる。また, pairing, projection として, それぞれ $a_1, \ldots, a_n$, $(a).i$ の形を用いる。ここで a はアリティ $\alpha_1 \otimes \cdots \otimes \alpha_n$ $(n \geq 2)$ の式で i は $1 \leq i \leq n$ をみたすとする。

式の簡単な**マクロ定義**を許すことにする。n を**名前** (名前は変数などと同様, 適当な文字の列とする), b を変数の自由な出現をふくまない式とするとき, $n \equiv b$ によって, 「n を b の略記として用いる」ということをあらわすものとする。

式 t がアリティ α を持つということを $t{:}\alpha$ と記す。

以下では通常の λ 式と同様に, 変数が**束縛**されている, **自由**にあらわれている等の概念を用いる。また, 式 b の

以下では x, y などで変数を, a, b などで式を, c などで定数を, n などで名前をあらわす。

マクロ: n がアリティ α を持つ式 b によってマクロ定義されていれば $n \equiv b : \alpha_0$。

関数適用: $a \equiv a' : \alpha \to \beta$, $b \equiv b' : \alpha$ であれば $a(b) \equiv a'(b') : \beta_0$。

β-則: $x : \alpha, a : \alpha, b : \beta$ で, しかも $b[x := a]$ の代入が可能であれば $((x)b)(a) \equiv b[x := a] : \beta_0$。

ξ-則: $x : \alpha, b \equiv b' : \beta$ であれば $(x)b \equiv (x)b' : \alpha \to \beta_0$。

α-則: $x : \alpha, y : \alpha, b : \beta$ で, しかも y が b の中に自由にあらわれなければ $(x)b \equiv (y)(b[x := y]) : \alpha \to \beta_0$。

η-則: $x : \alpha, b : \alpha \to \beta$ で, しかも x が b の中に自由にあらわれなければ $(x)(b(x)) \equiv b : \alpha \to \beta_0$。

組: $a_1 \equiv a_1' : \alpha_1, \ldots, a_n \equiv a_n' : \alpha_n$ であれば
$a_1, \ldots, a_n \equiv a_1', \ldots, a_n' : \alpha_1 \otimes \cdots \otimes \alpha_{n0}$

surjective pairing: $e : \alpha_1 \otimes \cdots \otimes \alpha_n$ であれば
$(e).1, \ldots, (e).n \equiv e : \alpha_1 \otimes \cdots \otimes \alpha_{n0}$

射影 1: $a \equiv a' : \alpha_1 \otimes \cdots \otimes \alpha_n$ で $1 \leq i \leq n$ であれば $(a).i \equiv (a').i : \alpha_{i0}$

射影 2: $a_1 \equiv a_1' : \alpha_1, \ldots, a_n \equiv a_n' : \alpha_n$ で $1 \leq i \leq n$ であれば $(a_1, \ldots, a_n).i \equiv a_i : \alpha_{i0}$

反射律: $a : \alpha$ ならば $a \equiv a : \alpha_0$

対称律: $a \equiv b : \alpha$ ならば $b \equiv a : \alpha_0$

推移律: $a \equiv a' : \alpha, a' \equiv a'' : \alpha$, ならば $a \equiv a'' : \alpha_0$

表 2: 式の同等性

中に自由にあらわれている変数 $x : \alpha$ に式 $a : \alpha$ を代入して得られる式も通常と同様に定義し, これを $b[x := a]$ であらわす。

同じアリティ α を持つ式 a, b の間に同等性を表わす二項関係 $a \equiv b : \alpha$ を $\beta\eta$-conversion として定義する。この定義を表 2にまとめておく。

型付 λ 計算でよく知られているように, $\equiv$ は決定可能である。

2.2 Martin-Löf 集合論

Martin-Löf 集合論はいわゆる Martin-Löf 型理論あるいは直観主義的型理論 (Intuitionistic Type Theory) のことである。ここでは全体集合 (universe) U を一つだけ用いるので, $\mathbf{ITT_1}$ などと記されているものと同等であるが, ここでは [7] に従った形式化を用いる。

Martin-Löf 集合論では式に関する判定の導出規則が定められる。また, 基本的な集合やその要素をあらわす定数とそれに伴う式の間の同等性および簡約化の規則も与えられる。

2.2.1 判定

判定は式と同じように形式的な対象であり, 四つの形がある。つまり a, b, A, B が式であるとき

$$A\ set \qquad A = B \qquad a \in A \qquad a = b \in A$$

などが判定であり, Martin-Löf 集合論ではこれ以外の形の判定はない。これらの判定はそれぞれ, A が集合である, A と B が等しい集合である, a が集合 A の要素である, a と b が集合 A の等しい要素である, という主張をあらわす。実はもっと正確な意味が [7] の中で英語で説明されているが, 我々はプログラムへの応用上, 判定の意味よりも判定の導出に注目するので, 意味の説明はここではしない。

2.2.3 以下で判定の導出規則が説明される。それらの導出規則を用いて判定 $A\ set$ が導出できるとき, 式 A を**集合** という。また, 判定 $a \in A$ が導出できるとき, 式 a を集合 A の**要素**という ($a \in A$ が導出できておれば常に $A\ set$ が導出できるように導出規則ができている)。

Martin-Löf 集合論では, (形式的な) 命題と集合は同一視され, また命題の証明は集合の要素と同一視される。つまり, $A\ set$ が導出できるとき A を**命題**といい, $a \in A$ が導出できるとき, a を A の**証明**という。また, 命題が正しいというのはその命題の証明を実際に構成できること, と定められる。以下, Martin-Löf 集合論の説明では集合と命題を自由にとりかえて議論する。

命題とその証明が形式的に定義されているので, Martin-Löf 集合論はその中に形式論理を含んでいる。

仮定付き判定もある。仮定付き判定とその導出は n について相互に帰納的に, 次にように定義される。x_1, ..., x_n が変数, a, b, A, B, A_1, ..., A_n ($n \geq 1$) が式のとき

$A(x_1, \ldots, x_n)\ set$ [Γ]
$A(x_1, \ldots, x_n) = B(x_1, \ldots, x_n)$ [Γ]
$a(x_1, \ldots, x_n) \in A(x_1, \ldots, x_n)$ [Γ]
$a(x_1, \ldots, x_n) = b(x_1, \ldots, x_n) \in A(x_1, \ldots, x_n)$ [Γ]

の四つは仮定付き判定である。ただし Γ は次のような並びとする:

$$x_1 \in A_1, x_2 \in A_2(x_1) \ldots x_n \in A_n(x_1, \ldots, x_{n-1})$$

ここで [Γ] を**仮定**という。

2.2.2 定数の導入

三種類の定数を区別する。つまり集合をあらわす**集合定数**, 正規要素 (canonical element) をあらわす**正規定数**, それに非正規要素をあらわす**非正規定数**の三種類である。この区別は定数の導入のときに指定される。

正規定数と非正規定数の区別は, 要素をあらわす式を正規式と非正規式に分け, 集合の要素を論じるときは, その集合に属する正規式だけを論じればいいようにするためのものである。正規式を定義しよう。飽和していて (つまりアリティが 0 で), 自由変数を含まない式は一般に

$$c(e_1, \ldots, e_n)$$

の形をしているが, ここで c が正規定数の時, この式を正規式という。それ以外の式をすべて非正規式という。非正規式が集合の要素である場合, その集合の非正規要素に関する簡約規則を何回か適用することによって, 正規式に変形できるようになっている (そのように簡約規則を定めることになっている)。

$$\frac{A\ set}{x \in A \quad x \in A}$$

$$\frac{a \in A}{a = a \in A} \qquad \frac{A\ set}{A = A}$$

$$\frac{a = b \in A}{b = a \in A} \qquad \frac{A = B}{B = A}$$

$$\frac{a = b \in A \quad b = c \in A}{a = c \in A} \qquad \frac{A = B \quad B = C}{A = C}$$

$$\frac{a \in A \quad A = B}{a \in B} \qquad \frac{a = b \in A \quad A = B}{a = b \in B}$$

$$\frac{C(x)\ set\ [x \in A] \quad a \in A}{C(a)\ set}$$

$$\frac{C(x)\ set\ [x \in A] \quad a = b \in A}{C(a) = C(b)}$$

$$\frac{c(x) \in C(x)\ [x \in A] \quad a \in A}{c(a) \in C(a)}$$

$$\frac{c(x) \in C(x)\ [x \in A] \quad a = b \in A}{c(a) = c(b) \in C(a)}$$

$$\frac{B(x) = C(x)\ [x \in A] \quad a \in A}{B(a) = C(a)}$$

$$\frac{b(x) = c(x) \in B(x)\ [x \in A] \quad a \in A}{b(a) = c(a) \in B(a)}$$

表 3: 判定導出の一般的規則

2.2.3 判定の導出規則 — 一般的規則

ここでは判定 P の導出規則のうち, 集合定数に依らないものを説明する。

判定 $P_1, \ldots, P_n$ から判定 P を導出することを,

$$\frac{P_1 \quad \cdots \quad P_n}{P}$$

と記す。このとき, P, P_i $(1 \leq i \leq n)$ は一般に仮定付き判定であるが, その仮定には, この規則に関係するものだけを記すことにする。例えば

$$\frac{A\ set \quad B(x)\ set\ [x \in A]}{\Pi(A, B)\ set}$$

と書いた場合, 実際には任意の仮定リスト Γ, Δ について,

- Γ には x に関する仮定が含まれていない。
- $y \in A$ が Γ に含まれ, $y \in B$ が Δ に含まれておれば, $A \equiv B$ ($\equiv$ は式の体系で定義されたもの)。

という条件のもとで,

$$\frac{A\ set\ [\Gamma] \quad B(x)\ set\ [\Delta, x \in A]}{\Pi(A, B)\ [\Gamma, \Delta]\ set}$$

という導出を行なってよいということを意味していることとする。

さて, **表 3**に, 集合定数に依らない判定導出規則の主なものを掲げる。この他にも, 判定に与えられている意味から導出規則を導きだすことができる。完全な導出規則は望むことができないが, どの程度の規則を仮定すれば当座の用に間に合うのかをはっきりさせる必要がある。

2.2.4 集合の定義

Martin-Löf 集合論は開かれた体系である。新しい集合を導入していくことができるが。新しい集合を導入するには次のようなことを行なわなければならない。

1. 新しい集合定数 S を導入し, そのアリティ α を定める。
2. 新しい正規定数 $c_1, \ldots, c_m$ を導入し, それらのアリティ $\beta_1, \ldots, \beta_m$ を定める。これらを S の構成子と呼ぶ。
3. 新しい非正規定数 $s_1, \ldots, s_n$ を導入し, それらのアリティ $\gamma_1, \ldots, \gamma_n$ を定める。これらを S の選択子と呼ぶ
4. 各選択子 s_i について, それを頭に持つ式 $s_i(e)$ を書き換える簡約規則を与える。これらの簡約規則を繰り返し適用することによって, $s_i(e)$ は必ずどれかの正規式 $c_j(e')$ に書き換えられるようでなければならない。
5. 各集合定数に対して次の四種類の導出規則を与える。実はこれらの導出規則はそれぞれ一定の形をしていなければならないが, ここではそれにはふれない。

   形成規則 判定 $A\ set$ や $A = B$ を導出するための規則である。

   導入規則 各構成子 c c', に対する判定
   $$c(e_1, \ldots, e_n) \in A \quad \text{や}$$
   $$c(e_1, \ldots, e_n) = c'(e'_1, \ldots, e'_n) \in A$$
   を導出するための規則である。

   除去規則 除去規則は S によってできる集合の要素についての命題 $C(x)$ を示す方法を与える。この規則は, $C(x)$ が, A のすべての要素について成り立つことを示すには, A のすべての正規要素 p について $C(p)$ を示せば十分であるという, 一種の構造的帰納法である。選択子は, この規則において x のパターンマッチをさせる役割を持つ。

   等号規則 A の等号規則は 4. において選択子に対して定められた簡約規則によって生成される等号を形式化した規則である。

以下では, Maya の意味記述に用いる Martin-Löf 集合論で, はじめに基本的なものとして導入される集合を説明する。これらの集合のみを考えている時には, Martin-Löf 集合論の中に含まれている形式論理は矛盾しないが, 上の方法によって, これ以外の集合 (命題) を導入した場合には, その無矛盾性が保証されない。無矛盾性を保証しつつ Martin-Löf 集合論に新しい集合を導入する一般的方法についての考察もある [1][2] が, まだ決定的な方法が出ていない。

2.2.4.1 空集合

集合定数 $\emptyset : 0$
構成子 なし
選択子 $\mathsf{case}_\emptyset : 0 \rightarrow 0$
選択子書き換え なし
形成規則

$$\frac{}{\emptyset\ set}$$

導入規則 なし
除去規則

$$\frac{a \in \emptyset \quad C(x)\ set\ [x \in \emptyset]}{\mathsf{case}_\emptyset(a) \in C(a)}$$

等号規則 なし

2.2.4.2 一点集合

集合定数 $\mathsf{T} : 0$
構成子 $\mathsf{tt} : 0$
選択子 $\mathsf{case}_{\mathsf{T}} : 0 \otimes 0 \rightarrowtail 0$
選択子書き換え

$$\frac{a \Rightarrow \mathsf{tt} \quad b \Rightarrow q}{\mathsf{case}_{\mathsf{T}}(a, b) \Rightarrow q}$$

形成規則

$$\frac{}{\mathsf{T}\ \mathit{sct}}$$

導入規則

$$\frac{}{\mathsf{tt} \in \mathsf{T}}$$

除去規則

$$\frac{a \in \mathsf{T} \quad C(x)\ \mathit{sct}\ [x \in \mathsf{T}] \quad b \in C(\mathsf{tt})}{\mathsf{case}_{\mathsf{T}}(a, b) \in C(a)}$$

等号規則

$$\frac{C(x)\ \mathit{sct}\ [x \in \mathsf{T}] \quad b \in C(\mathsf{tt})}{\mathsf{case}_{\mathsf{T}}(\mathsf{tt}, b) = b \in C(\mathsf{tt})}$$

2.2.4.3 集合族の直積

集合定数 $\Pi : 0 \otimes (0 \rightarrowtail 0) \rightarrowtail 0$
構成子 $\lambda : (0 \rightarrowtail 0) \rightarrowtail 0$
選択子 $\mathsf{apply} : 0 \otimes 0 \rightarrowtail 0$
選択子書き換え

$$\frac{a \Rightarrow \lambda(c) \quad c(b) \Rightarrow q}{\mathsf{apply}(a, b) \Rightarrow q}$$

形成規則

$$\frac{A\ \mathit{sct} \quad B(x)\ \mathit{sct}\ [x \in A]}{\Pi(A, B)\ \mathit{sct}}$$

導入規則

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

除去規則

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

等号規則

$$\frac{b(x) \in B(x)\ [x \in A] \quad a \in A}{\mathsf{apply}(\lambda(b), a) = b(a) \in B(a)}$$

2.2.4.4 同等性の集合

集合定数 $\mathsf{Id} : 0 \otimes 0 \otimes 0 \rightarrowtail 0$
構成子 $\mathsf{id} : 0 \rightarrowtail 0$
選択子 $\mathsf{idpeel} : 0 \otimes 0 \rightarrowtail 0$
選択子書き換え

$$\frac{c \Rightarrow \mathsf{id}(a)}{\mathsf{idpeel}(c, d) \Rightarrow d(a)}$$

形成規則

$$\frac{A\ \mathit{sct} \quad a \in A \quad b \in A}{\mathsf{Id}(A, a, b)\ \mathit{sct}}$$

導入規則

$$\frac{a \in A}{\mathsf{id}(a) \in \mathsf{Id}(A, a, a)}$$

除去規則

$$\frac{\begin{array}{l} a \in A \quad b \in A \quad c \in \mathsf{Id}A, a, b \\ C(x, y, z)\ \mathit{sct}\ [x \in A, y \in A, z \in \mathsf{Id}(A, x, y)] \\ d(x) \in C(x, x, \mathsf{id}(x))\ [x \in A] \end{array}}{\mathsf{idpeel}(c, d) \in C(a, b, d)}$$

等号規則

$$\frac{\begin{array}{l} a \in A \\ C(x, y, z)\ \mathit{sct}\ [x \in A, y \in A, z \in \mathsf{Id}(A, x, y)] \\ d(x) \in C(x, x, \mathsf{id}(x))\ [x \in A] \end{array}}{\mathsf{idpeel}(\mathit{id}(a), d) = d(a) \in C(a, a, \mathsf{id}(a))}$$

2.2.4.5 自然数の集合

集合定数 $\mathsf{N} : 0$
構成子 $0 : 0,\ \mathsf{succ} : 0 \rightarrowtail 0$
選択子 $\mathsf{natrec} : 0 \otimes 0 \otimes (0 \otimes 0 \rightarrowtail 0) \rightarrowtail 0$
選択子書き換え

$$\frac{\begin{array}{c} a \Rightarrow 0 \quad b \Rightarrow q \\ \hline \mathsf{natrec}(a, b, c) \Rightarrow q \\ a \Rightarrow \mathsf{succ}(d) \quad c(d, \mathsf{natrec}(d, b, c)) \Rightarrow q \end{array}}{\mathsf{natrec}(a, b, c) \Rightarrow q}$$

形成規則

$$\frac{}{\mathsf{N}\ \mathit{sct}}$$

導入規則

$$\frac{}{0 \in \mathsf{N}} \qquad \frac{a \in \mathsf{N}}{\mathsf{succ}(a) \in \mathsf{N}}$$

除去規則

$$\frac{\begin{array}{l} a \in \mathsf{N} \\ C(v)\ \mathit{sct}\ [v \in \mathsf{N}] \\ d \in C(0) \\ e(x, y) \in C(\mathsf{succ}(x))\ [x \in \mathsf{N}, y \in C(x)] \end{array}}{\mathsf{natrec}(a, d, e) \in C(a)}$$

等号規則

$$\frac{\begin{array}{l} a \in \mathsf{N} \\ C(v)\ \mathit{sct}\ [v \in \mathsf{N}] \\ d \in C(0) \\ e(x, y) \in C(\mathsf{succ}(x))\ [x \in \mathsf{N}, y \in C(x)] \end{array}}{\mathsf{natrec}(0, d, e) = d \in C(0)}$$

$$\frac{\begin{array}{l} a \in \mathsf{N} \\ C(v)\ \mathit{sct}\ [v \in \mathsf{N}] \\ d \in C(0) \\ e(x, y) \in C(\mathsf{succ}(x))\ [x \in \mathsf{N}, y \in C(x)] \end{array}}{\mathsf{natrec}(\mathsf{succ}(a), d, e) = e(a, \mathsf{natrec}(a, d, e)) \in C(\mathsf{succ}(0))}$$

2.2.4.6 直和

集合定数 $+ : 0 \otimes 0 \rightarrowtail 0$
構成子 $\mathsf{inl} : 0 \rightarrowtail 0,\ \mathsf{inr} : 0 \rightarrowtail 0$
選択子 $\mathsf{when} : 0 \otimes (0 \rightarrowtail 0) \otimes (0 \rightarrowtail 0) \rightarrowtail 0$
選択子書き換え

$$\frac{\begin{array}{c} a \Rightarrow \mathsf{inl}(d) \quad b(d) \Rightarrow q \\ \hline \mathsf{when}(a, b, c) \Rightarrow q \\ a \Rightarrow \mathsf{inr}(d) \quad b(d) \Rightarrow q \end{array}}{\mathsf{when}(a, b, c) \Rightarrow q}$$

形成規則

$$\frac{A\ \mathit{sct} \quad B\ \mathit{sct}}{+(A, B)\ \mathit{sct}}$$

導入規則

$$\frac{a \in A \quad B\ \mathit{sct}}{\mathsf{inl}(a) \in +(A, B)} \qquad \frac{A\ \mathit{sct} \quad b \in B}{\mathsf{inr}(b) \in +(A, B)}$$

除去規則

$$\frac{\begin{array}{l} c \in +(A, B) \\ C(v)\ \mathit{sct}\ [v \in +(A, B)] \\ d(x) \in C(\mathsf{inl}(x))\ [x \in A] \\ e(y) \in C(\mathsf{inr}(y))\ [y \in B] \end{array}}{\mathsf{when}(c, d, e) \in C(c)}$$

等号規則

$$\frac{\begin{array}{l} a \in A \\ C(v)\ \mathit{sct}\ [v \in +(A, B)] \\ d(x) \in C(\mathsf{inl}(x))\ [x \in A] \\ e(y) \in C(\mathsf{inr}(y))\ [y \in B] \end{array}}{\mathsf{when}(\mathsf{inl}(a), d, e) = d(a) \in C(\mathsf{inl}(a))}$$

$$\frac{\begin{array}{l} b \in B \\ C(v)\ \mathit{sct}\ [v \in +(A, B)] \\ d(x) \in C(\mathsf{inr}(x))\ [x \in A] \\ e(y) \in C(\mathsf{inl}(y))\ [y \in B] \end{array}}{\mathsf{when}(\mathsf{inr}(b), d, e) = e(b) \in C(\mathsf{inr}(b))}$$

2.2.4.7 集合族の直和

集合定数 $\Sigma : 0 \otimes (0 \rightarrowtail 0) \rightarrowtail 0$
構成子 $(\_, \_) : 0 \otimes 0 \rightarrowtail 0$ 以下では $(\_, \_)(a, b)$ を (a, b) と略記する。
選択子 $\mathsf{split} : 0 \otimes (0 \otimes 0 \rightarrowtail 0) \rightarrowtail 0$
選択子書き換え

$$\frac{a \Rightarrow (c, d) \quad b(c, d) \Rightarrow q}{\mathsf{split}(a, b) \Rightarrow q}$$

形成規則

$$\frac{A\ \mathit{sct} \quad B(x)\ \mathit{sct}\ [x \in A]}{\Sigma(A, B)\ \mathit{sct}}$$

導入規則

$$\frac{a \in A \quad B(x)\ \mathit{sct}\ [x \in A] \quad b \in B(a)}{(a, b) \in \Sigma(A, B)}$$

除去規則

$$\frac{\begin{array}{l} c \in \Sigma(A, B) \\ C(v)\ \mathit{sct}\ [v \in \Sigma(A, B)] \\ d(x, y) \in C((x, y))\ [x \in A, y \in B(x)] \end{array}}{\mathsf{split}(c, d) \in C(c)}$$

等号規則

$$\frac{\begin{array}{l} a \in A \quad b \in B(a) \\ c \in \Sigma(A,B) \\ C(v)\ set\ [v \in \Sigma(A,B)] \\ d(x,y) \in C(\langle x,y\rangle)\ [x \in A, y \in B(x)] \end{array}}{\mathrm{split}(\langle a,b\rangle, d) = d(a,b) \in C(\langle a,b\rangle)}$$

2.2.4.8 整列順序

集合定数 $\mathrm{W} : 0 \otimes (0 \twoheadrightarrow 0) \twoheadrightarrow 0$

構成子 $\mathrm{sup} : 0 \otimes (0 \twoheadrightarrow 0) \twoheadrightarrow 0$

選択子 $\mathrm{wrec} : 0 \otimes (0 \otimes (0 \twoheadrightarrow 0) \otimes (0 \twoheadrightarrow 0) \twoheadrightarrow 0) \twoheadrightarrow 0$

選択子書き換え

$$\frac{a \Rightarrow \mathrm{sup}(c,d) \quad b(c,d,(x)\mathrm{wrec}(d(x),b)) \Rightarrow q}{\mathrm{wrec}(a,b) \Rightarrow q}$$

形成規則

$$\frac{A\ set \quad B(x)\ set\ [x \in A]}{\mathrm{W}(A,B)\ set}$$

導入規則

$$\frac{a \in A \quad b(x) \in W(A,B)\ [x \in B(a)]}{\mathrm{sup}(a,b) \in \mathrm{W}(A,B)}$$

除去規則

$$\frac{\begin{array}{l} a \in \mathrm{W}(A,B) \\ C(v)\ set\ [v \in \mathrm{W}(A,B)] \\ b(y,z,u) \in C(\mathrm{sup}(y,z)) \\ \quad [y \in A, z(x) \in \mathrm{W}(A,B)\ [x \in B(y)], \\ \quad u(x) \in C(z(x))\ [x \in B(y)]] \end{array}}{\mathrm{wrec}(a,b) \in C(a)}$$

等号規則

$$\frac{\begin{array}{l} d \in A \quad e(x) \in \mathrm{W}(A,B)\ [x \in B(d)] \\ C(v)\ set\ [v \in \mathrm{W}(A,B)] \\ b(y,z,u) \in C(\mathrm{sup}(y,z)) \\ \quad [y \in A, z(x) \in \mathrm{W}(A,B)\ [x \in B(y)], \\ \quad u(x) \in C(z(x))\ [x \in B(y)]] \end{array}}{\mathrm{wrec}(\mathrm{sup}(d,e),b) \in b(d,e,(x)\mathrm{wrec}(e(x),b))C(\mathrm{sup}(d,e))}$$

2.2.4.9 全体集合

集合定数 $\mathrm{U} : 0,\ \mathrm{Set} : 0 \twoheadrightarrow 0$

構成子

$$\begin{array}{l} \widehat{\emptyset} : 0, \quad \widehat{\mathrm{T}} : 0, \quad \widehat{\mathrm{N}} : 0 \\ \widehat{\mathrm{Id}} : 0 \otimes 0 \otimes 0 \\ \widehat{+} : 0 \otimes 0 \twoheadrightarrow 0 \\ \widehat{\Pi} : 0 \otimes (0 \twoheadrightarrow 0) \twoheadrightarrow 0 \\ \widehat{\Sigma} : 0 \otimes (0 \twoheadrightarrow 0) \twoheadrightarrow 0 \\ \widehat{\mathrm{W}} : 0 \otimes (0 \twoheadrightarrow 0) \twoheadrightarrow 0 \end{array}$$

選択子

$$\begin{array}{l} \mathrm{urec} : 0 \otimes 0 \otimes 0 \otimes 0 \otimes \\ \quad (0 \otimes 0 \otimes 0 \otimes 0 \twoheadrightarrow 0) \otimes \\ \quad (0 \otimes 0 \otimes 0 \otimes 0 \twoheadrightarrow 0) \otimes \\ \quad (0 \otimes (0 \twoheadrightarrow 0) \otimes 0 \otimes (0 \twoheadrightarrow 0) \twoheadrightarrow 0) \otimes \\ \quad (0 \otimes (0 \twoheadrightarrow 0) \otimes 0 \otimes (0 \twoheadrightarrow 0) \twoheadrightarrow 0) \otimes \\ \quad (0 \otimes (0 \twoheadrightarrow 0) \otimes 0 \otimes (0 \twoheadrightarrow 0) \twoheadrightarrow 0) \\ \twoheadrightarrow 0 \end{array}$$

選択子書き換え

$$\frac{a \Rightarrow \widehat{\emptyset} \quad a_1 \Rightarrow b}{\mathrm{urec}(a,a_1,a_2,a_3,a_4,a_5,a_6,a_7,a_8) \Rightarrow b}$$

$$\frac{a \Rightarrow \widehat{\mathrm{T}} \quad a_2 \Rightarrow b}{\mathrm{urec}(a,a_1,a_2,a_3,a_4,a_5,a_6,a_7,a_8) \Rightarrow b}$$

$$\frac{a \Rightarrow \widehat{\mathrm{N}} \quad a_3 \Rightarrow b}{\mathrm{urec}(a,a_1,a_2,a_3,a_4,a_5,a_6,a_7,a_8) \Rightarrow b}$$

$$\frac{a \Rightarrow \widehat{\mathrm{Id}}(A,c,d) \quad a_4(A,c,d,\mathrm{urec}(A,a_1,\ldots,a_8)) \Rightarrow b}{\mathrm{urec}(a,a_1,a_2,a_3,a_4,a_5,a_6,a_7,a_8) \Rightarrow b}$$

$$\frac{\begin{array}{l} a \Rightarrow \widehat{+}(A,B) \\ a_5(A,B,\mathrm{urec}(A,a_1,\ldots,a_8),\mathrm{urec}(B,a_1,\ldots,a_8)) \Rightarrow b \end{array}}{\mathrm{urec}(a,a_1,a_2,a_3,a_4,a_5,a_6,a_7,a_8) \Rightarrow b}$$

$$\frac{\begin{array}{l} a \Rightarrow \widehat{\Pi}(A,B) \\ a_6(A,B,\mathrm{urec}(A,a_1,\ldots,a_8),(w)\mathrm{urec}(B(w),a_1,\ldots,a_8)) \Rightarrow b \end{array}}{\mathrm{urec}(a,a_1,a_2,a_3,a_4,a_5,a_6,a_7,a_8) \Rightarrow b}$$

$$\frac{\begin{array}{l} a \Rightarrow \widehat{\Sigma}(A,B) \\ a_7(A,B,\mathrm{urec}(A,a_1,\ldots,a_8),(w)\mathrm{urec}(B(w),a_1,\ldots,a_8)) \Rightarrow b \end{array}}{\mathrm{urec}(a,a_1,a_2,a_3,a_4,a_5,a_6,a_7,a_8) \Rightarrow b}$$

$$\frac{\begin{array}{l} a \Rightarrow \widehat{\mathrm{W}}(A,B) \\ a_8(A,B,\mathrm{urec}(A,a_1,\ldots,a_8),(w)\mathrm{urec}(B(w),a_1,\ldots,a_8)) \Rightarrow b \end{array}}{\mathrm{urec}(a,a_1,a_2,a_3,a_4,a_5,a_6,a_7,a_8) \Rightarrow b}$$

形成規則

$$\frac{\overline{\mathrm{U}\ set}}{\mathrm{Set}(A)\ set}$$

$$\frac{\mathrm{U}\ set \quad A \in \mathrm{U}}{\mathrm{Set}(A)\ set}$$

導入規則

$$\frac{}{\widehat{\emptyset} \in \mathrm{U}} \quad \frac{}{\widehat{\mathrm{T}} \in \mathrm{U}} \quad \frac{}{\widehat{\mathrm{N}} \in \mathrm{U}}$$

$$\frac{}{\mathrm{Set}(\widehat{\emptyset}) = \emptyset} \quad \frac{}{\mathrm{Set}(\widehat{\mathrm{T}}) = \mathrm{T}} \quad \frac{}{\mathrm{Set}(\widehat{\mathrm{N}}) = \mathrm{N}}$$

$$\frac{A \in \mathrm{U} \quad a \in \mathrm{Set}(A) \quad b \in \mathrm{Set}(A)}{\widehat{\mathrm{Id}}(A,a,b) \in \mathrm{U}}$$

$$\frac{A \in \mathrm{U} \quad a \in \mathrm{Set}(A) \quad b \in \mathrm{Set}(B)}{\mathrm{Set}(\widehat{\mathrm{Id}}(A,a,b)) = \mathrm{Id}(\mathrm{Set}(A),a,b)}$$

$$\frac{A \in \mathrm{U} \quad B \in \mathrm{U}}{\widehat{+}(A,B) \in \mathrm{U}}$$

$$\frac{A \in \mathrm{U} \quad B \in \mathrm{U}}{\mathrm{Set}(\widehat{+}(A,B)) = +(\mathrm{Set}(A),\mathrm{Set}(B))}$$

$$\frac{A \in \mathrm{U} \quad B(x) \in \mathrm{U}\ [x \in \mathrm{Set}(A)]}{\widehat{\Pi}(A,B) \in \mathrm{U}}$$

$$\frac{A \in \mathrm{U} \quad B(x) \in \mathrm{U}\ [x \in \mathrm{Set}(A)]}{\mathrm{Set}(\widehat{\Pi}(A,B)) = \Pi(\mathrm{Set}(A),(x)\mathrm{Set}(B(x)))}$$

$$\frac{A \in \mathrm{U} \quad B(x) \in \mathrm{U}\ [x \in \mathrm{Set}(A)]}{\widehat{\Sigma}(A,B) \in \mathrm{U}}$$

$$\frac{A \in \mathrm{U} \quad B(x) \in \mathrm{U}\ [x \in \mathrm{Set}(A)]}{\mathrm{Set}(\widehat{\Sigma}(A,B)) = \Sigma(\mathrm{Set}(A),(x)\mathrm{Set}(B(x)))}$$

$$\frac{A \in \mathrm{U} \quad B(x) \in \mathrm{U}\ [x \in \mathrm{Set}(A)]}{\widehat{\mathrm{W}}(A,B) \in \mathrm{U}}$$

$$\frac{A \in \mathrm{U} \quad B(x) \in \mathrm{U}\ [x \in \mathrm{Set}(A)]}{\mathrm{Set}(\widehat{\mathrm{W}}(A,B)) = \mathrm{W}(\mathrm{Set}(A),(x)\mathrm{Set}(B(x)))}$$

除去規則

$$\frac{\begin{array}{l} a \in \mathrm{U} \\ C(v)\ set\ [v \in \mathrm{U}]a_1 \in C(\widehat{\emptyset}) \\ a_2 \in C(\widehat{\mathrm{T}}) \\ a_3 \in C(\widehat{\mathrm{N}}) \\ a_4(x,y,z,u) \in C(\widehat{\mathrm{Id}}(x,y,z)) \\ \quad [x \in \mathrm{U}, y \in \mathrm{Set}(x), z \in \mathrm{Set}(x), u \in C(x)] \\ a_5(x,y,z,u) \in C(\widehat{+}(x,y)) \\ \quad [x \in \mathrm{U}, y \in \mathrm{U}, z \in C(x), u \in C(y)] \\ a_6(x,y,z,u) \in C(\widehat{\Pi}(x,y)) \\ \quad [x \in \mathrm{U}, y(v) \in \mathrm{U}\ [v \in \mathrm{Set}(x)], \\ \quad\quad z \in C(x), u(v) \in C(y(v))\ [v \in \mathrm{Set}(x)]] \\ a_7(x,y,z,u) \in C(\widehat{\Sigma}(x,y)) \\ \quad [x \in \mathrm{U}, y(v) \in \mathrm{U}\ [v \in \mathrm{Set}(x)], \\ \quad\quad z \in C(x), u(v) \in C(y(v))\ [v \in \mathrm{Set}(x)]] \\ a_8(x,y,z,u) \in C(\widehat{\mathrm{W}}(x,y)) \\ \quad [x \in \mathrm{U}, y(v) \in \mathrm{U}\ [v \in \mathrm{Set}(x)], \\ \quad\quad z \in C(x), u(v) \in C(y(v))\ [v \in \mathrm{Set}(x)]] \end{array}}{\mathrm{urec}(a,a_1,\ldots,a_8) \in C(a)}$$

等号規則 省略 — 書き換え規則から復元できる。

2.2.5 略記法の導入

2.2.1 で述べたように, Martin-Löf 集合論では命題と集合を同一視する。例えば, $\emptyset$ は証明を持たない命題だからこれは恒偽命題とみなされる。このような読みかえをしやすくするため, また, 他の目的のために, 次のよ

```
INF- 超記号 ::= 構文定義記号 | 構文定義終止記号
              | 構文作用記号類 | 構文括弧類。
構文定義記号 ::= 「::=」。
構文定義終止記号 ::= 「。」。
構文作用記号類 ::= 反復指示記号 {。。。}
                 | 区切り指示記号 {△}
                 | 可換積記号 {•} | 選択記号 {|}
                 | 選択積記号 {／}。
反復指示記号 ::= 「。。。」。
区切り指示記号 ::= 「△」。
可換積記号 ::= 「•」。
選択記号 ::= 「|」。
選択積記号 ::= 「／」。
構文括弧類 ::= ( | ) | [ | ] | 「 | 」。
構文定義式 ::= 構文定義見出し 構文単位名 {左辺}
              「::=」構文式 {右辺} 「。」。
構文素子 ::= 実素字列 | 構文単位名 | ( )
           | ( 構文式 ) | [ 構文式 ]。
構文因子 ::= 構文素子 [[△ 構文素子] 。。。]。
構文項 ::= 構文因子 。。。
         | 構文因子 △ •。。。[△ 構文素子]。
構文式 ::= 構文項 △ |。。。
         | 構文項 (／構文項)。。。。
```

表 4: INF の構文の自身による定義

うな式の体系におけるマクロ定義を導入する。

$\bot \equiv \emptyset$　　(恒偽命題)

$\forall \equiv \Pi$

$\rightarrow \equiv (A)(B)\Pi(A,(x)B)$

(ただし x は A にも B にもあらわれないとする)

$\supset \equiv (A)(B)\Pi(A,(x)B)$

(ただし x は A にも B にもあらわれないとする)

$\vee \equiv +$

$\times \equiv (A)(B)\Sigma(A,(x)B)$

$\wedge \equiv \times$

$\exists \equiv \Sigma$

3 Maya の構文と直感的意味

3.1 INF

Maya の抽象構文記述のために [10] で定義された INF を用いる。この節では INF を簡単に説明する。表 4 に INF 自身で定義された INF の構文定義を掲げる。INF は BNF の変形拡張なので, BNF に慣れている読者は構文からその意味を類推できると思われる。ここでは, いくつかの BNF にない部分についてのみ説明する。

INF 記述中の注釈は中括弧 ({ と }) によって囲われた部分である。また, 終端記号はかぎ括弧で囲ったり (例えば「記号」) 下線を引いたり (例えば 記号) して非終端記号と区別する。

構文定義見出しについては [10] でも定義されていない。これは構文定義式を引用する際のラベルとして働くものであり, 本稿では常に空のまま残しておくことにする。

反復指示記号 。。。 を構文素子 P に続けた形

$$P\ 。。。$$

は P が表わす形を一つ以上連接した形, つまり $p = p_1 p_2 \cdots p_k$, ただし $k \geq 1$ で $p_1, p_2, \ldots, p_k$ はすべて P が表わす形, となるような p の形すべてを表わす。また,

$$P\ \triangle\ Q\ 。。。$$

は, P が表わす形に加えて, P を二回以上繰り返す場合はその間に Q が表わす形を挟んで得られる形, つまり $p = p_1 q_1 p_2 q_2 \cdots q_{k-1} p_k$, ただし $k \geq 2$ で $p_1, p_2, \ldots, p_k$ はすべて P が表わす形, $q_1, q_2, \ldots, q_k$ はすべて Q が表わす形, となるような p の形すべてを表わす。

INF にはこの他に, 定まった個数の構文因子が表わす形をちょうど一回ずつ任意の順序で連接する可換積 (•), ある構文項の集まりから高々一個ずつを選んでそれらが表わす形を任意の順序で連接する選択積 (／) などの機能も用意されているが, 本稿では用いないのでここでは詳しく説明しない。

3.2 Maya の抽象構文

INF による Maya の抽象構文の定義を表 5 に掲げる。Maya はまだ設計途中で, 計算機による構文解析の対象とすべき具象構文の検討を行なう段階にはない。たとえば, 型としては Martin-Löf 集合論での小さな型をすべて書けるようにしたいが, 具体的な構文をまだ考えていない。

3.3 Maya の直感的意味

まず直感的な意味を記す。Maya のトップ・レベルは仕様宣言, 実現宣言, 基本型定義の三種類ある。このうち, 通常用いられるのは前二者だけである。

3.3.1 基本型定義

基本型定義は, 文字通り Maya の基本型の定義を与えるものである。プログラミング言語の基本型は, 言語定義の時に固定され, プログラマがそれを拡張して新しい基本型を定義することは許されないのがふつうである。けれども, 基本型というのは, プログラミング言語が稼働する環境に応じて, そこで最も効率よく動くようなものを選択できるようにするべきである。そこで, Maya には基本型の定義の機能も持たせる。

Maya の基本型は Martin-Löf 集合論の集合を対応させて意味づけられる。そこで, 新しい基本型の定義は, 2.2.4 で説明したような意味で, Martin-Löf 集合論に新しい集合を導入することに相当する。2.2.4 で述べたように, 新しい集合の導入のし方によっては Martin-Löf 集合論に含まれている形式論理が矛盾する可能性がある。新しい集合を含んだ Martin-Löf 集合論の無矛盾性の保証は基本型定義の作者によって, Martin-Löf 集合論とは別の枠組でなされなければならない。したがっ

```
仕様宣言 ::= spec 名前 [ 名前「∈」仕様 ] is
             [import 名前 △ 「;」 ooo]
             [type 名前 △ 「;」 ooo]
             [val (名前「∈」型) △ 「;」 ooo]
             [axiom 命題 △ 「;」 ooo]
             end。
仕様 ::= 名前。
型 ::= 修飾付名前 | 型「×」型 | 型「→」型
   | W「(」型「,」型「)」
   {その他, 集合を表わす式に相当するもの}。
修飾付名前 ::= 名前 △ 「.」 oooo
命題 ::= 「⊥」 | 項「=」項 | 命題「∧」命題
     | 命題「∨」命題 | 命題「⊃」命題
     | 「∀」名前「∈」型 命題
     | 「∃」名前「∈」型 命題。
項 ::= 修飾付名前
   | 項[「(」項 △ 「,」 ooo 「)」]
   | 「(」名前「)」項。
実現宣言 ::= impl 名前 [ 名前 「∈」 仕様 ] is
             [import 名前 △ 「;」 ooo]
             [type (名前 「≡」 型) △ 「;」 ooo]
             [val (名前 「≡」 項) △ 「;」 ooo]
             [proof 項 △ 「;」 ooo]
             end
             「∈」仕様。
実現 ::= 実現 [ 実現 ]
     | 名前。
基本型定義 ::= PRIM 定数導入 IS
               CONSTR 定数導入 △ 「;」 ooo
               SELECT 定数導入 △ 「;」 ooo
               REWRITE (項「⇒」項) △ 「;」 ooo
               END。
定数導入 ::= 名前 「:」 アリティ [ 導出規則 ]
アリティ ::= ○ | アリティ「→≫」アリティ。
導出規則 ::= 判定 [ 判定 △ 「,」 ooo]。
判定 ::= 項 set | 項「=」項
     | 項「∈」項 | 項「=」項「∈」項
     | 判定 [(名前「∈」項)△ 「,」 ooo]。
```

表 5: Maya の抽象構文

て、基本型定義は、「システム・モジュール」とでもいうべきもので、一般ユーザに開放されるべきものではない。

基本型定義が書かれる場合として, 例えば Maya を新しい環境で実現する時に, 新しい環境で効率的に実現でき, しかも Maya の実現に役立つような集合を導入する場合を想定している。なお, 基本型定義の処理を行なうと, 判定の導出規則が新しく増えるので, 言語処理系自身の変更が必要である。これは好ましいことではないので, Martin-Löf 集合論の枠組自体についての考察を行なって, このようなことが起こらないようにしたいと考えている。

3.3.2 実現

Maya ではプログラム・モジュールのことを実現と呼ぶ。実現はデータ型, 値とそれが持つべきデータ型, およびそれらの間に成り立つ関係の証明を一束にしたものである。データ型, 値, 証明は, それぞれ type, val, proof のキーワードで始まる並びで記される。これらの記述の中で他の実現を参照することもでき, その場合は参照される実現の名前を import に始まる並びに列挙する (継承)。さらに, 実現を実現によってパラメータ化することもでき, その場合には, 仮引数となるべき実現とそれが満たすべき仕様を, はじめに大括弧でくくって示す。

実現において type, val で定義されるデータ型, 値には名前がつけられる。データ型や値を無関係に集めてきた組を記すだけなら, 実現の中で名前をつける必要はない。けれども実現の要素の間には

1. 後の方で定義する値の型として前に定義したデータ型を参照する。
2. 前に定義した値を用いて値を定義する。
3. 前に定義したデータ型や値を用いて証明を記す。

などの依存関係があるので, データ型と値には参照のために名前をつけておく必要がある。証明自身は他から参照されることがないので, これには名前付けの必要がない。

3.3.3 仕様

仕様は, 実現で定義されるべきデータ型, 値とそれが持つべきデータ型を列挙し, それらの間に成り立つべき関係を公理として記すものである。データ型, 値, 関係は, それぞれ type, val, axiom のキーワードで始まる並びで記される。これらの記述の中で他の実現を参照することもでき, その場合は参照される実現の名前を import に始まる並びに列挙する。さらに, 記述される実現が実現によってパラメータ化されている場合もあり, その場合には, 仮引数となるべき実現とそれが満たすべき仕様をはじめに大括弧でくくって示す。

3.3.3.1 仕様と Σ 集合, Π 集合

仕様は Martin-Löf 集合論の集合をあらわす。仕様があらわす集合はつねに

$$\Pi(A, (x)\Sigma(A_1, (x_1)\Sigma(A_2 \ldots (x_{n-1})\Sigma(A_n, \mathsf{T}))))$$

という形をしている。Π はパラメータ付の仕様の場合にだけつく。はじめの A は仮引数となる実現が満たすべき仕様である。

さて, 実現はデータ型, 値, 公理などを一組に束ねた組をあらわすのだから, それを要素とする仕様が直積のようなものであると考えるのが自然である。仕様では, 通常の直積のようにいくつかのものを無関係に束ねるだけでなく, 組の後の方に表われる要素が属するべき集合が, はじめの方に表われた要素に依存して決められる場合もある。例えば,

```
...
type t
val a ∈ t
...
```

という仕様の断片を考えると, 値 a の属するべき集合は, より前に宣言されたデータ型 t に依存して決まる。Σ 集合を使うとこのような情況をうまくあらわすことができる。

仕様のパラメータ部の説明に Π 集合を用いるのも同様で, 本体に記されている要素の型が, パラメータに依存する場合がある。例えば,

```
spec S [X∈S′ ] is
...
val b∈X. t
...
```

では, 値 b の属するべき集合がパラメータ X に依存しており, Π 集合がこのような情況をやはりうまくあらわしている。

さらに, Σ, Π 集合には式の体系からくる可視性規則があり, これを仕様でも自然に利用することができる。Maya では仕様や実現自身の名前は大局的な定義しか許さないが, それらの中で定義されるデータ型や値の名前の可視性規則には, Σ, Π に備わっている可視性規則がそのまま適用できる。

3.3.3.2 仕様と実現の関係

すでに述べたように, 仕様は Martin-Löf 集合論の集合をあらわし, 実現は要素をあらわす。実現 I のあらわす要素が仕様 S のあらわす集合に属するとき, I は S の実現であるという。S で指定されているデータ型, 値はすべて I で定義されており, しかもそれらの間に, S の公理に記された関係が成り立っている, ということが, 集合に属するという判定で表現されるわけである。

3.3.3.3 データ型と全体集合

データ型は集合によってあらわされるが, 我々はデータ型を仕様の要素として考えたい。Martin-Löf 集合論では一般には集合を要素として考えることができないが, 2.2.4.9 で説明したように, 「小さな集合」は Set と $\hat{}$ によって全体集合 U の値にに符号化することができる。データ型を, 小さな集合だけに限っても通常の値を対象とするプログラミングには何もさしつかえることがないので, Maya ではデータ型を U の要素によって意味づける。

なお, Maya はパラメータ付仕様の形をもってはいるが, これを実パラメータに適用する操作は定義されない。仕様自体は小さな集合ではないので, 仕様を返す関数というものも Martin-Löf 集合論で定義できないからである。パラメータ付仕様の適用の操作は便利なので, これを記述できるようにしたいが, そのためには

1. Martin-Löf 集合論のかわりに, 「非述語的」型理論を意味の記述に用いる。
2. U の上にさらに大きな全体集合 U_1 を定める

などの方法が考えられる。

4 Maya の Martin-Löf 集合論による意味

Maya の三種類のトップ・レベルのうち, 仕様宣言と実現宣言の意味は Martin-Löf 集合論におけるマクロ定義として, 基本型定義の意味は, Martin-Löf 集合論自体の拡張として, それぞれ意味づけられる。以下ではその意味づけを, トップ・レベル評価器の動作としてスケッチする。

トップ・レベル評価器は, 現在の環境 $\mathcal{E}_C$ の下で Maya のトップ・レベルを評価して, その結果, $\mathcal{E}_C$ を変更したり, トップ・レベル評価器自身の動作を変えたり (これは基本型定義を評価した場合) する。

4.1 環境

まず, 環境を定義しよう。環境 $\mathcal{E}$ は, つぎのものから構成される。

1. Martin-Löf 集合論の集合定義の集まり。ひとつの集合定義は 2.2.4 節で示したように, ひとつの集合定数, 構成子の集まり, 選択子の集まり, 選択子簡約規則の集まり, それに例の四種類の導出規則の集まりからなる。
2. 式の体系におけるマクロ定義の集まり。ただし, ここでは, 要素をあらわすマクロ定義には, その要素が属する集合も定義の中に含めておくこととする。そこで, ここでのマクロ定義は次の二種類である。
   (a) $n \equiv e$ ただし e は集合
   (b) $n \equiv e \in e'$ ただし e' は集合, e は e' の要素。
   このマクロ定義の集まりは仕様や実現の名前を, その実体で定義するものである。

環境の初期値 $\mathcal{E}_0$ は, 集合定義の集まりとして 2.2.4 で示した集合定義を持ち, 空のマクロ定義の集まりを持つ環境である。

4.2 基本型定義の評価

基本型定義 P は, 構文から想像されるように, 一つの集合定義をあらわしている。この集合定義を $[\![P]\!]$ と書こう。現在の環境 $\mathcal{E}_C$ が $\mathcal{E}$ の時に基本型定義を評価すると, 評価器は $\mathcal{E}$ の集合定義に $[\![P]\!]$ を加えた環境 $\mathcal{E}'$ をつくり, 現在の環境 $\mathcal{E}_C$ を $\mathcal{E}'$ で置き換える。

4.3 仕様宣言の評価

まず, $\mathcal{E}_C = \mathcal{E}$ のときに仕様宣言 S が宣言する集合 $[\![S]\!]$ を示そう。ここで, 便利のために次のような略記法を用いる。つまり, $(\Sigma x \in A)B$, $(\Pi x \in A)B$ をそれぞれ $\Sigma(A,(x)B)$, $\Pi(A,(x)B)$ の略記として用いる。

さて, 仕様 S がパラメータを持たないとき, つまり

```
spec S is
import i_1 ; ... ; i_l
type t_1 ; ... ; t_m
val v_1∈u_1 ; ... ; v_n∈u_n
axiom a_1 ; ... ; a_p
end
```

の形のときには, $[\![S]\!]$ は次のような集合である。

$$
\begin{aligned}
&(\Sigma x_{11} \in A_{11}) \ldots (\Sigma x_{1l_1} \in A_{1l_1}) \\
&\quad \cdots \\
&(\Sigma x_{l1} \in A_{l1}) \ldots (\Sigma x_{ll_1} \in A_{ll_1}) \\
&(\Sigma t_1 \in \mathsf{U}) \ldots (\Sigma t_m \in \mathsf{U}) \\
&(\Sigma v_1 \in \mathsf{Set}([\![\mathsf{u}_1]\!])) \ldots (\Sigma v_n \in \mathsf{Set}([\![\mathsf{u}_n]\!])) \\
&(\Sigma \alpha_1 \in [\![\mathsf{a}_1]\!]) \ldots (\Sigma \alpha_p \in [\![\mathsf{a}_p]\!]) \\
&\mathsf{T}
\end{aligned}
$$

以下, この集合について説明する。ここで, はじめの

$$(\Sigma x_{11} \in A_{11}) \ldots (\Sigma x_{1l_1} \in A_{1l_1})$$

については,

$$\mathsf{i}_1 \equiv t_1 \in (\Sigma x_{11} \in A_{11}) \ldots (\Sigma x_{1l_1} \in A_{1l_1})\mathsf{T}$$

というマクロ定義が E にあるものとする。i_1 は既に評価され終えた実現の名前である。i_1 は仕様の名前でないことに注意されたい。また, 仕様があらわす集合は一般には頭に $(\Pi X \in A)$ がついても良いが, import による継承が許されるのはこの接頭辞がない場合だけである。[1]

この行は i_1 の継承を表現するものである。つまり, この行より後では, 名前 x_{1i} は実現 i_1 で x_{1i} として宣言されたものを指すことになる。以下

$$(\Sigma x_{l1} \in A_{l1}) \ldots (\Sigma x_{1l_1} \in A_{ll_1})$$

までは同様にそれぞれの継承を表現する。次の

$$(\Sigma\ \mathsf{t}_1 \in \mathsf{U}) \ldots (\Sigma\ \mathsf{t}_m \in \mathsf{U})$$

は $\mathsf{t}_1 \ldots \mathsf{t}_m$ に対応するもので, これで, データ型を一つ与えることは, 全体集合 U の要素を一つ与えることである, ということを表現している。

$$(\Sigma\ \mathsf{v}_1 \in \mathsf{Set}([\![\mathsf{u}_1]\!])) \ldots (\Sigma\ \mathsf{v}_n \in \mathsf{Set}([\![\mathsf{u}_n]\!]))$$

は val $\mathsf{v}_1 \in \mathsf{u}_1$; ... ; $\mathsf{v}_n \in \mathsf{u}_n$

に対応しているものである。仕様宣言において u_i は, ほぼ Martin-Löf 集合論の小さな集合となる式に対応しているが, その小さな集合を $[\![\mathsf{u}_i]\!]$ によってあらわす。同様に a_i も, ほぼ Martin-Löf 集合論の命題となる式に対応しているが, その命題を $[\![\mathsf{a}_i]\!]$ によってあらわす。そして, 各 a_i に対して, 新しい変数 α_i をとりだして

$$(\Sigma \alpha_1 \in [\![\mathsf{a}_1]\!]) \ldots (\Sigma \alpha_p \in [\![\mathsf{a}_p]\!])\mathsf{T}$$

をつくる。実はこうして作った命題は

$$[\![\mathsf{a}_1]\!] \wedge \cdots \wedge [\![\mathsf{a}_p]\!]$$

に他ならないが, 取り扱いやすくするため Σ の形を保っておく。

以上で, パラメータのない場合を説明したが, パラメータがある場合の $[\![S]\!]$ は頭に $(\Pi X \in S')$ がつくだけである。つまり S が

```
spec S [X∈S'] is
... end
```

の形の場合, $[\![S]\!]$ は

$$(\Pi\ \mathsf{X} \in [\![\ \mathsf{S}'\ ]\!]) \cdots$$

という形の集合である。

さて, $\mathcal{E}_C = \mathcal{E}$ のときに上のような形の仕様宣言 S を評価すると, 評価器は $\mathcal{E}$ に マクロ定義 $\mathsf{S} \equiv [\![S]\!]$ を付け加えて新しい環境 $\mathcal{E}'$ を作り, $\mathcal{E}_C$ の値を $\mathcal{E}'$ で置き換える。

4.4 実現宣言の評価

前節と同様に, $\mathcal{E}_C = \mathcal{E}$ のときに実現宣言 I によって宣言される要素 $[\![I]\!]$ をまず示そう。

I がパラメータを持たない実現宣言 であるとき, つまり

```
spec I is
import i_1 ; ... ; i_l
type t_1 ≡ τ_1 ; ... ; t_m ≡ τ_m
val v_1≡υ_1 ; ... ; v_n≡υ_n
proof π_1 ; ... ; π_p
end ∈ S
```

[1] 実はこれらの制限はすべて, パラメータ付仕様の実引数への適用を定義しないために必要になっていることである。

の形のときには, $[\![I]\!]$ は次のような要素である。

$$\begin{array}{l}\langle\ [\![\mathsf{i}_1]\!],\ldots,[\![\mathsf{i}_l]\!]\\ \tau_1,\ldots,\tau_m,\\ \upsilon_1,\ldots,\upsilon_n\\ \pi_1,\ldots,\pi_p\\ \mathsf{tt}\\ \rangle\end{array}$$

以下, この要素について説明する。

$\mathcal{E}$ の中に $\mathsf{i}_i \equiv \langle e_1,\ldots,e_\nu,\mathsf{tt}\rangle \in e'$ というマクロ定義があるとき,

$$[\![\mathsf{i}_i]\!] = \langle e_1,\ldots,e_\nu\rangle$$

と定める。 i_i のマクロ値の最後の tt だけを除くのである。

τ_i, υ_i, π_i については, それぞれがもともと Martin-Löf 集合論の要素を忠実に反映した構文をしているから, それらが何をあらわすかは明らかであろう。

パラメータがある場合の $[\![I]\!]$ は $\lambda((X)\ldots)$ で囲われる。つまり I が

```
spec I [X∈S'] is
··· end
```

の形の場合, $[\![I]\!]$ は

$$\lambda((\mathsf{X})\cdots)$$

という形の関数になる。

さて, $\mathcal{E}_C = \mathcal{E}$ のときに上のような形の実現宣言 I を評価すると, 評価器は $\mathcal{E}$ に マクロ定義 $\mathsf{I} \equiv [\![I]\!] \in \mathsf{S}$ を付け加えて新しい環境 $\mathcal{E}'$ を作り, $\mathcal{E}_C$ の値を $\mathcal{E}'$ で置き換える。

5 むすび

Martin-Löf 集合論を紹介し, 次に言語 Maya の抽象構文を紹介し, Maya の意味を, トップ・レベルの抽象構文木を評価することが Martin-Löf 集合論の集合や要素をどのように操作することになるのかを示すことによって与えた。

Maya を使ってどのようなことができるのかを例示することができなかったが, いくつか可能性をあげよう。まず, これははじめの目的であるが, 証明つきのプログラム・モジュールを Maya によって書くことができる。しかもそのモジュールをパラメータ化して, 部品としてとっておくことができる。

基本型定義によって machine instruction の定義を与えたり, プロセス代数に基づく基本型を与えたりして, reactive な系を Martin-Löf 集合論のような型理論で記述できないかと考えている。

Maya では, プログラム (実現) には常にそれが満足する仕様がくっついているが, ある目的のために書かれたプログラムを, 予想もしなかったような別の仕様を実現するものとして用いることが, 部品化の魅力である。Maya ではパラメータ付実現を用いて, 実現を別の仕様を満たすものとして見なおすことができる。これは OBJ3 での view の機能に相当する。また, Standard ML の functor はちょうど Maya のパラメータ付実現に相当する。

なお, Maya のデータ型定義はマクロ定義を基本にしていて, 表面上, 再帰的なデータ型定義はあらわれていないが, 整列順序 W を用いてリストなど, 通常の再帰的データ型はたいてい定義することができる。Standard ML などの関数型言語での再帰的データ型定義の構文を syntax sugar として導入し, それを W による定義に翻訳する方法を与えることもできる。

さて, ここで与えた意味づけはまだきわめて informal なものであり, これから formal な意味論に仕上げていかねばならない。その前に克服すべき問題点がいくつかあるので列挙する。

まず, 現在の形式化では, 基本型定義を評価すると処理系の変更が必要であるが,

これは好ましくない。そこで, 式の体系はもっと強力にして処理系の基本的能力を高め, そのようなことを避ける方法がないかと考えている。

また, パラメータ付仕様のパラメータへの適用ができないが, これはいかにも不自然な制限であり, 不便も感じるので, やはり意味の世界となる Martin-Löf 集合論を整備して完全なパラメータ付仕様を取り扱えるようにしたい。

Maya では証明は実現と一体になっており, はじめにのべたように仕様, 実現, 証明の三つを別々に保存することができない。証明を構文的に分離することは簡単であるが, 証明自身も部品化できる可能性があるのではないかと考え, 現在のようにプログラムと一体にしている。パラメータ付実現によって, プログラムだけでなく, 証明をも写すこともできるのではないかと考えている。

参考文献

[1] Dybjer, Peter: Inductive Sets and Families in Martin-Löf's Type Theory and Their Set-Theoretic Semantics, in [4], 1990.

[2] Dybjer, Peter: An inversion principle for Martin-Löf's type theory, in *Proceedings of the Workshop on Programming Logic*, Båstad, 1989.

[3] Goguen, J.A.; Winkler, T.: Introducing OBJ3, SRI-CSL-88-9, SRI International Technical Report, 1988.

[4] Huet, G.; Plotkin, G. (eds.): Proceedings of the first workshop in logical frameworks, Antibes, 1990.

[5] Milner, R; Tofte, Mads; Harper, R.: The Definition of Standard ML, MIT Press, 1990.
[6] Milner, R; Tofte, Mads: Commentary on Standard ML, MIT Press, 1991.
[7] Nordström, Bengt; Petersson, Kent; Smith, Jan M.: Programming in Martin-Löf's Type Theory, Oxford University Press, 1990.
[8] Salvesen, A. and Smith, J.M.: The Strength of the Subset Type in Martin-Löf's type theory, in *Proceedings of LICS '88*, IEEE, 1988.
[9] Wirth, N.: Programming in Modula-2, Springer-Verlag, 1982.
[10] 米田 信夫 (編): プログラム言語, 岩波講座 情報科学 9, 岩波書店, 1983.