

マージオペレータを持つレコード計算

立 木 秀 樹†

最近, Common Lisp にオブジェクト指向の機能を付け加えた言語 CLOS が広まりつつある。CLOS では、総称関数を用いて、オブジェクト指向のメッセージ送信を一種の関数呼び出しとして実現している。しかし、CLOS の総称関数は、Lisp の関数とは別物であり、Lisp の基礎に存在する関数型言語の計算機構を総称関数に拡張しているわけではない。そのためには、 λ 計算が Lisp の理論的基礎であるのと同様、総称関数を関数とする形式計算を考える必要がある。本論文では、そのような形式計算として、レコードとマージ・オペレータを型つき λ 計算に付け加えた計算 λ_m を提案する。 λ_m は、総称関数の動作を関数型言語の枠組でモデル化している。 λ_m のレコードは CLOS のオブジェクトに、関数は総称関数に対応する。レコードのマージはレコードの結合に、関数のマージは CLOS のメソッド融合に対応している。 λ_m は、簡約による型の保存等形式計算としてよい性質を持ち、表示の意味がドメイン上で構成可能である。その際、マージは上限操作として意味が付けられる。

A Record Calculus with Merge Operator

HIDEKI TSUIKI†

Within the past few years, many dialects of Lisp with object oriented features have been designed. CLOS, among them, is the standard for merging object oriented programming with Common Lisp. In CLOS, they implement message passing as a kind of function call using generic functions. In this paper, we propose, as a theoretical foundation of generic functions, a new calculus λ_m which is an extension of record calculus with a merge operator. A record in this calculus corresponds to an object in CLOS, and a function in this calculus corresponds to a generic function in CLOS. A merge of two records corresponds to record concatenation, and a merge of two functions corresponds to method combination. The semantics of λ_m is given domain theoretically, in which the merge operator is interpreted as the least upper bound.

1. はじめに

最近、オブジェクト指向の機能を Lisp に加えた言語が幾つか考えられている。その中でも、CLOS¹⁾ は、Common Lisp のオブジェクト指向の機能として、広く使われている。CLOS では、各クラスがそのクラスのオブジェクトに適用可能なメソッドを持つのではなく、システム全体で同じ名前を持つメソッドを集めて総称関数 (generic function) と呼ばれる 1 つの関数とすることにより、オブジェクト指向の特徴であるメッセージ送信を一種の関数呼び出しとして実現している。総称関数は、引数のクラスに応じて適用可能なメソッドを配送 (method dispatch) する。しかも、適用可能なメソッドが複数存在する場合には、それらのメソッドを融合 (method combination) したものを配

送する。総称関数のこのような動作は、オブジェクト指向計算を関数的に表現する必要から考えられて来たが、その基礎となるような形式計算は、今まで存在しなかった。特に、CLOS の総称関数は Lisp の関数とは別物であり、Lisp の基礎に存在する関数型言語の計算機構が総称関数に拡張されているわけではない。

本論文の目的は、 λ 計算が Lisp の理論的基礎であるのと同様に CLOS の理論的基礎として、総称関数を関数とする形式計算 λ_m を構成することにある。CLOS は Lisp の拡張であり、弱く型付けされた言語であるが、形式体系を考える時には、クラスと型を同一視して、クラスの継承と型の継承を同一視するのが自然である。よって、 λ_m は型つき λ 計算にレコード型と、型継承、および、総称関数を構成するためのマージ・オペレータを加えたものとして構成する。

一般に、総称関数のような機構は、複数の関数が同じ名前を共有しているために生じる一種の多重定義 (overloading) として扱われてきた。 λ_m では、総称関

† 慶應義塾大学環境情報学部
Department of Environmental Information, Keio University

数を、複数の型の引数に対して適用可能で、動作が引数の型に応じて変化する1つの関数として考える。すなわち、総称関数を、名前と独立に存在する計算の扱う対象 (“first class object”) とする。これにより、総称関数を他の総称関数の引数や返値にすること、すなわち、高階な総称関数を考えることが可能となる。

λ_m では、総称関数は、CLOS のように同じ名前を付けることではなく、マージ・オペレータを用いて関数を結合することにより構成する。マージはまず、基礎となるデータ型であるレコードに対して定義され、そして、関数に対して定義される。マージにより構成された関数の適用は、それぞれの構成要素の関数を適用した結果をマージすることにより行われる。

λ_m は、式の型が唯一に定まり、それが計算により保存される等の、形式計算として期待される性質を持っている。また、表示的意味をドメイン上で考えた時、マージは上限を取る操作に対応している。

オブジェクト指向言語の理論的な基礎をあたえるため、幾つかのレコード計算が現在までに提案されてきた^{2)-5), 12)}。しかし、これらは、多相型、静的型チェックといった、オブジェクト指向の静的側面を扱っており、 λ_m のように、メッセージ配送やメソッド融合といったオブジェクト指向の動的な側面を扱う形式計算は、今まで考えられていなかった。

この計算の定義および規則は付録にまわし、本論では、付録を参照しながらその基礎にある考え方について述べる。

λ_m の概略を説明する。まず、付録Aで、 λ_m の型および式を定義する。型にはマージという概念があり、マージ可能規則 (“M-” ではじまる名前を持つ) を満たす型をマージしたものも型となる。付録Bで型の継承に関する推論規則 (“I-” ではじまる名前を持つ) を与える。式は、最初疑似式として与え、その中で、付録Cで定義される型づけ規則 (“T-” ではじまる名前を持つ) により型づけされたものだけを式とする。最後に、付録Dで、 λ_m の実行方法にあたる、簡約規則を与える。式とは別に、正規形 (Canonical form) を考え、式 e が正規形 c に簡約 (reduce) されるという関係 $e \triangleright c$ を、簡約規則 (“E-” ではじまる名前を持つ) で与える。簡約規則を与えるのに、 λ_m の式ではないが、式を簡約する途中でのみ許される中間式を導入する。

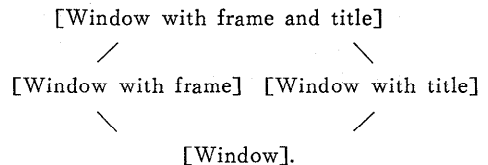
本論では、まず、例により、総称関数をマージで与えるというアイデアを説明する。それから、このシス

テムの型の継承と実行との関係、マージ操作について述べる。最後に、 λ_m の形式計算としての性質を述べ、ドメイン上で、意味論的考察を行う。

2. 例

まず、総称関数が関数のマージと考えられることを、例を用いて説明する。この例は、付録Eで具体的に λ_m の式として与えられる。

今、次のような、多重継承関係にある型を考える。この図では、親の型を下に位置している。



ここで、[Window] はウィンドウの型、[Window with frame] および [Window with title] は、それぞれ枠付きおよびタイトル付きのウィンドウの型、[Window with frame and title] は、枠とタイトル付きウィンドウの型とする。各々の型は、レコード型として定義されているとする。

この上で、ウィンドウを移動する総称関数 move を考える。継承の考え方では、型 [Window] に対して定義された move は、すべての子供の型に対して適用可能である。しかし、型 [Window] に対して定義された move を [Window with frame] の要素に対して適用したのでは、枠は移動されずに元の場所に残されてしまう。そこで、[Window with frame] に対しては、枠を移動するという操作を、[Window] の move に追加しなくてはならない。同様に、[Window with title] に対しては、タイトルを移動するという操作を加えなくてはならない。このように、親のクラスに対する差分だけ定義するプログラミング・スタイルを、オブジェクト指向では Mixin-based programming と呼んでいる⁶⁾。このように定義すると、多重継承の考え方から、[Window with frame and title] に対しては、特に操作を定義しなくても、move の呼び出しで、3つの型に対する move が起動され、タイトルとフレームと共にウィンドウが移動することが期待される。

CLOS では、子のクラスに定義されたメソッドと、親のクラスに定義されたメソッドを組み合わせた実行は、子のクラスのメソッド中で、call-next-method を用いて親の型に対するメソッドを明示的に呼び出すこ

とにより行われる。あるいは、メソッドは *primary*, *after*, *before*, *around* に分類されているので、それを用いると、よりきめ細かな制御が可能である。このように、CLOS では、1つの総称関数の呼び出しで複数のメソッドが組み合わさって呼び出される。このことを、メソッド融合と呼んでいる。

CLOS は手続き的な言語のため複雑なメソッド融合の方法が提供されているが、 λ_m は関数型言語の枠組で考えるので総称関数を構成する各関数の呼び出しの順番を考える必要がない。 λ_m では、関数の融合方法として、適用可能な関数をすべて適用し、結果をマージするという方法を考える。結果をマージするには、関数の結果となるレコードに対してマージが考えられている必要がある。幸い、レコードには、レコード結合 (concatenation) というマージ操作が定義される。上の例でいえば、[Window] を引数として移動先のウィンドウを返す関数、[Window with frame] を引数として移動先の枠を返す関数、[Window with title] を引数として移動先のタイトルを返す関数をマージしたものとして、*move* を定義する。すると、[Window with frame] の要素に *move* を適用した時には、移動先のウィンドウと枠をマージしたものが、[Window with frame and title] の要素に *move* を適用すれば、移動先のウィンドウとタイトルと枠をマージしたものが返ることになる。

これでは、親に対する操作は必ず呼ばれるため、CLOS が許しているように、親に対する操作と完全に異なる操作を子供の型に対して定義できなくなる。しかし、そのような関数は、型変換との整合性を保たないために、プログラムのエラーの原因となると考えられる⁷⁾。 λ_m では、そのような関数は排除している。しかし、次章で述べるように、子供の型と親の型で結果が矛盾している時に、子供の型の結果を優先するように計算を変更するのは容易である。

3. 型継承と関数実行

λ_m の型継承の意味について考える。型の間の継承関係の推論規則は、付録 B で定義されている。この規則は、マージを *intersection type* とみなした時、文献 8, 9) 等の *intersection type* を含む型システムの推論規則と基本的に同じである。

規則は同じでも、本論文の型継承の扱いは、文献 8, 9, 2) 等の部分型 (Subtype) とは使われ方が異なる。部分型は、

$$(\text{SUBS}) \frac{A \text{ は } B \text{ の部分型 } a : A}{a : B}$$

という規則 (Subsumption rule と呼ばれる) により、部分型の要素は親の型にも属するという概念であった。(SUBS) が存在すると、式の型が一意でなくなる。それに対し、ここでモデル化しようとしているのは、各値が型を持っており、その型に応じて関数適用の結果が変化するという計算であり、各値の型は唯一である必要がある。

よって、 λ_m では、継承を型の包含関係と考えるのではなく、各型の間に共通部分はなく 2つの型の間に型変換関数が存在する時に継承関係が存在するとする。

型 A が型 B を継承している時、 $A > B$ と表記することにする。この順序は、Subtype と逆であるが、型変換は、 A の方が B よりも多くの情報を持っており、余分な情報を取り除く操作と考えることができ、情報量の順序として、このほうが自然である。また、正規形 c の S への型変換は $c|_S$ という中間式により表す。

(SUBS) を用いずに、親の型に定義された関数を子供の型にも適用可能にするため、型の推論規則に (T-APP) を、簡約規則に (E-APP) を採用する。(E-APP) は、関数適用の簡約の際に、引数に型変換を行って適当な型の値に変換してから適用するというものである。文献 10) 等では、型推論時に (SUBS) の適用ごとに A から B への型変換を静的挿入した式を構成し、構成した結果を、(SUBS) の存在しない計算で実行している。しかし、マージ・オペレータの存在する時には、 F, G が関数の型の時、例えば、 $F \vee G$ から F への型変換は、 λ_m の式として表現できない。よって、型変換を λ 式の適用まで遅られて、(E-APP) により解決することにする。(E-APP) の規則は、「 $(\lambda x^S. e) e'$ を簡約したもの c' は、 e' の型 S' を計算し、 $S' > S$ を確認し、 e' を簡約した結果 c を計算し、 e に現れる x を $c|_S$ に置き換えたものを簡約して得られるもの」と読む。ここで、実行時に必要に応じて式の型を計算しているが、Dynamic Typing¹¹⁾ のように、各式に型も同時に持たせておけば、この手間を省いて効率的に実現可能である。

4. マージ操作

マージ操作 $\vee$ の意味するものを、詳しく考察する。マージの意味は、マージ式を簡約した結果 (各 e, e' に対し、 $e \vee e' \triangleright c$ となる c) によって与えられる。

まず、*int*, *bool* といった基礎型に対するマージは、

$1 \vee 1 \triangleright 1, \text{true} \vee \text{true} \triangleright \text{true}$ という具合に両者が等しい時のみ定義することができる (E-LUB), (E-CONST), (N-LUBN 1). これ以外の基礎型に対するマージは, 実行時にエラーを発生するとする (N-LUBN 2).

レコードに対しては, 文献 12) などで研究されているように, 結合操作としてマージが自然に定義される. 例えば, $\{l_1=1, l_2=\text{true}\} \vee \{l_3=2, l_4=1\} \triangleright \{l_1=1, l_2=\text{true}, l_3=2, l_4=1\}$ である. 両方のレコードが同じスロットを持つ場合には, 再帰的にマージを行う. 例えば, $\{l_1=\{l=1\}, l_2=\text{true}\} \vee \{l_1=\{l=1, l'=2\}, l_3=1\} \triangleright \{l_1=\{l=1, l'=2\}, l_2=\text{true}, l_3=1\}$ となる (N-LUBR).

もちろん, 常にマージが可能なのではない. 例えば, $\{l_1=1, l_2=\text{true}\}$ と $\{l_1=\text{true}, l_3=1\}$ はマージ不能である. また, $\{l_1=1, l_2=\text{true}\}$ と $\{l_1=2, l_3=1\}$ もマージ不能である. 前者のマージは, l_1 スロットの型が異なるため, 次章で述べるように, 型が与えられず, 式から除かれる.

後者のマージ・エラーは, 静的に型により排除することはできない. 基礎型のマージ・エラーに起因しており, 実行時にエラーを発生する. このエラーが静的に判断できないのはこの計算の欠点とも考えられる. しかし, 基礎型に対するマージは常に左側の値を用いると計算規則 (N-LUBN 2) を変更することにより, マージ・エラーを起こさないように容易に λ_m を変更可能である. この時, 子供の型に対する操作を左側に書くように記法を定めると, マージが矛盾する時には子供の型に対する操作を優先することになる. 実行時のエラーを考えるのは, 2 章で述べたように, 継承関係を保存しない関数を許さないためである.

マージ・オペレータが存在する時, 複数のスロットを持つレコードは, 単一のスロットを持つレコードのマージとして表記可能である. よって, λ_m では, 複数のスロットを持つレコードは, 単一スロットを持つレコードのマージとして表し, 複数スロットによる記法は, 略記法と考える. ただし, 正規形としては, 複数スロットのレコードを考える.

関数のマージは, 値に対するマージから導かれる. すなわち, $f \vee g$ は, x に適用した時に, 片方だけ適用可能ならその結果を返し, 両方適用可能なら両方適用しその結果をマージしたものを返す関数とする. 上の章で述べたように, $f: A \rightarrow B, x: C$ とした時, f が x に適用可能なのは, $C \triangleright A$ の時である. これは, 型エラーを表す中間式 nil を考え, 関数が適用

可能でない時には, nil を返すとして関数を全域的なものにし (E-APPN), nil は, 何とマージしてもその値となる値 (N-NIL 1, N-NIL 2) とすれば, 片方しか適用できない場合を特別視する必要がなくなる. すなわち, $f \vee g$ は, x に適用した時に, f と g と両方適用し, その結果をマージしたものを返すとする (E-APPL).

関数のマージは, それ以上簡約不能であり, それ自身, 正規形である (N-NUBF). また, 関数がある型に型変換したものも, 正規形である (N-COEF). このために, 関数のマージを適用するための規則 (E-APPL), および, 関数を型変換したものを適用するための規則 (E-APPC) が存在する.

5. 型のマージ

マージ式は, それぞれの型をマージした型に属する (T-LUB). マージは, 型のなす順序集合を考えた時, 上限を表している. λ_m では, 値のマージを, マージできる型を制限することにより制限している. 型 A と B がマージ可能なことを, $A \uparrow B$ と書く.

まず, マージ可能なのは, 同一の基礎型, レコード型同士, あるいは関数型同士に限られる. レコードに関して, (M-LABEL 1) により, ラベルが異なればマージ可能であり, (M-LABEL 2) により, ラベルが同じでもラベルの型がマージ可能ならば, マージ可能である. レコード式と同様, 複数スロットを持つレコード型は, 一つのスロットを持つレコード型の上限として表す.

関数型に関しては, まず, 定義域の型 S_1 と S_2 がマージ不可能なら, 両方の型に型変換可能な引数は存在せず, 常に, どちらか片方の関数しか適用されず, マージ可能である (M-FUN 1). S_1 と S_2 がマージ可能なら, $S_1 \vee S_2$ の引数を適用した結果の型 $S_1' \vee S_2'$ が存在しなくてはならない (M-FUN 2).

6. λ_m の性質

λ_m は, 形式計算として期待される, 以下の性質を持っている. 証明は, 概略を述べるにとどめる.

Theorem 1 (型継承の決定性) $S \triangleright S'$ は決定可能である.

proof) λ_m の型推論規則は, 文献 8) の intersection type と同等であり, 文献 8) と同様に示せる. $\square$

型の継承関係は, 型推論に用いられており, その決定性は, この簡約規則が実際に動くために必要であ

る. $A > B$ かつ $B > A$ の時, A と B は同値な型 ($A \approx B$) という. 型の同値関係も, 決定可能である.

Theorem 2 (型の唯一性) $H \vdash e : S, H \vdash e : S'$ の時, S と S' は等しい.

prof) 適用可能なルールが一意に定まるので, 明らかである. $\square$

Theorem 3 (簡約の唯一性, 型の保存) 閉じた式 e に対し, $e \triangleright c$ が成り立つ c は存在すれば唯一であり, c は e と同値な型を持つ.

prof) 型づけされた式に対しては, 必ずどれかの 1 つだけのルールが適用される. また, 各ルールの $\triangleright$ の両辺で, 型が変化していないことも確かめられる. $\square$

残念ながら, λ_m では, 正規化定理 (Normalization Theorem) は成り立たない. すなわち, どのような正規形にも簡約できない式が存在する. 例えば, int を省略して i , $(i \rightarrow i) \rightarrow i \vee i \rightarrow i$ を省略して S とし, $e : ((S \rightarrow i) \vee S)$ を $(\lambda x^S. xx) \vee (\lambda x^{i \rightarrow i}. 1) \vee (\lambda x^{i. 1})$ とした時, ee は, i の型を持つ式であるが, ee の正規形を求める手続きは終了しない.

	$\vdots$
	$e s s \quad e s s s \triangleright c_5$
	$\vdots$
	$e s s \quad e s s \triangleright c_3$
(E-APP)	$(\lambda x^S. xx) e s \triangleright c_3 \quad (\lambda x^{i \rightarrow i}. 1) e s \triangleright 1$
	$(\lambda x^{i. x}) e s \triangleright nil \quad NF(c_3 \vee 1 \vee nil) = c_2$
(E-APPL)	$e e s \triangleright c_2 \quad NF(c_2 i) = c_1$
(E-APPC)	$e s \quad e s \triangleright c_1$
(E-APP)	$(\lambda x^S. xx) e \triangleright c_1 \quad (\lambda x^{i \rightarrow i}. 1) e \triangleright 1$
	$(\lambda x^{i. x}) e \triangleright nil \quad NF(c_1 \vee 1 \vee nil) = c$
(E-APPL)	$ee \triangleright c$

これは, ee の簡約が, $\dots 1 \vee 1 \vee 1$ という無限のマージを生成することを意味する. 正規化定理が成り立たないことは, この計算の欠点ではあるが, この例はかなり特殊な例であり, もっと意味のある例が存在するかどうか不明である. 実際, これは総称関数を引数として渡さなければ生じず, オブジェクト指向の基礎となる計算としては, これは生じない. 詳しくは, 文献 13) を参照されたい.

7. 表示の意味

文献 14) で構成した意味領域に, 各型のマージエラーを表す値を最大元として追加することにより, λ_m の表示の意味が与えられる. ここでは, λ_m の直観的な理解を与えるための説明を行う. 詳しくは, 文献 13) を参照されたい.

型のなす集合を $\mathcal{G}$ とする. $\mathcal{G}$ は, 型の継承関係により順序集合となっている. 型 t に対応する領域を, $\mathcal{D}(t)$ で表わすことにする. λ_m の特徴は, 型変換とマージ操作である. 型変換の意味を与えるため, $t > s$ の時, 以下の性質を満たす $\mathcal{D}(t)$ から $\mathcal{D}(s)$ への型変換関数 $\delta_{t,s}$ が存在すると考える.

1. $\delta_{t,t} = id_t$.
2. $u > t > s$ の時, $\delta_{t,s} \circ \delta_{u,t} = \delta_{u,s}$.

マージ操作は, 意味領域の上で上限として意味を与える. マージは, マージ・エラーを起こすことがある. マージ・エラーを表現するために, 各型 t に対して, $\mathcal{D}(t)$ に, 最大元 $\top_t$ が存在するとする. これにより, $\mathcal{D}(t)$ は, 順序集合となる. この順序は, PCF¹⁵⁾ の意味論等で現れる停止しない計算による順序とは異なる. 特に, 後者では, 終了しない計算を表すために最小元 ($\perp$) を与えていたのに対称的であることに注意されたい. マージは, 異なる型の要素に対しても定義される. そのためには, 各 t に対し, $\mathcal{D}(t)$ を合わせた全体 $\mathcal{D}$ もまた順序が与えられる必要がある. この順序は, $\delta_{t,s}$ と, 各型の中の順序から, 推移閉包をとることにより構成する. (T-LUB) の規則により, $t \uparrow s$ の時, $\mathcal{D}(t)$ の元と $\mathcal{D}(s)$ の元の上限が存在する必要がある.

このような性質を満たす $M = (\mathcal{D}, \mathcal{G}, \mathcal{A})$ の 3 つ組みを, 意味領域として考える. ここで, $\mathcal{D}$ は値のなす順序集合, $\mathcal{G}$ は型のなす順序集合, $\mathcal{A}$ は値の型を与える $\mathcal{D}$ から $\mathcal{G}$ への関数である. $\mathcal{G}$ の順序は型継承を表し, $\mathcal{D}$ の順序は値間の型変換可能性を表している. $\mathcal{A}^{-1}(t)$ は, 上の $\mathcal{D}(t)$ と一致する. 上であげた性質に加えて, 関数空間やレコード空間を定義するための幾つかの性質, および, 領域方程式を解くための幾つかの性質を加えた数学的構造を, M-domain と定義する.

M_B を, 基礎となる型のなす, 以下の M-domain とする.

$$\mathcal{G}_B = \{null, int, bool\}.$$

$$\mathcal{D}(null) = \{nil\}$$

$$\mathcal{D}(i) = \text{最大元を持つ flat upper}$$

$$\text{semi-lattice}(i = int, bool).$$

λ_m では, 型として, レコード型と関数型を考えた. λ_m の意味領域は, これらの型も含んだ M-domain を考えなくてはならない.

レコードは, ラベルのなす可算集合 $\mathcal{L}$ から $\mathcal{D}$ への関数空間で, 有限個のラベルでのみ nil 以外の値を持

つものとする。値が nil であることは、そのラベルが存在しないことに対応する。レコード型は、同様のラベルのなす加算集合から $\mathcal{I}$ への関数とする。 $\mathcal{M}$ が M-domain の時、 $\mathcal{M}$ 上のレコードのなす M-domain を $[\mathcal{L} \rightarrow \mathcal{M}]$ とする。

関数に関しては、総称関数に対応するものも考えなくてはならない。そのために、関数は、各型間の関数ではなく、値空間 $\mathcal{D}$ 間の関数として構成する。M-domain $\mathcal{M} = (\mathcal{D}, \mathcal{I}, \mathcal{A})$, $\mathcal{M}' = (\mathcal{D}', \mathcal{I}', \mathcal{A}')$ が与えられた時に、関数の型として、 $\mathcal{I}$ から $\mathcal{I}'$ への単調関数を考える。そして、関数として、 $\mathcal{D}$ から $\mathcal{D}'$ への単調関数で、型を保存するものを考える。すなわち、 $\mathcal{D}$ の元で同じ型をもつものは、 $\mathcal{D}'$ の同じ型に移されるものを考える。こうして得られた M-domain を、 $[\mathcal{M} \rightarrow \mathcal{M}']$ とおく。

さて、ここで欲しいのは、基本型、レコード型、関数型を全て含む領域である。 $\mathcal{M}$ を、次の領域方程式の解とする。この領域方程式の解法については、文献 14) を参照されたい。

$$\mathcal{M} = \mathcal{M}_B + [\mathcal{L} \rightarrow \mathcal{M}] + [\mathcal{M} \rightarrow \mathcal{M}]$$

$\mathcal{M}$ 上で、 λ_m の表示の意味が与えられる。すなわち、型式 S に対しその意味として $\mathcal{I}$ の要素を与える関数 $\mathcal{E}^S$ 、式 e に対して (自由変数の意味を与える環境 $\mathcal{H}$ をもらって) その意味として $\mathcal{D}$ の要素を与える関数 $\mathcal{E}$ が定義され、

1. $T > S \iff \mathcal{E}^T[T] > \mathcal{E}^S[S] \text{ in } \mathcal{I}$.
2. $H \vdash e : S$ なら、 $\mathcal{A}(\mathcal{E}[e](\mathcal{H})) = \mathcal{E}^S[S]$.
3. $e \triangleright e'$ なら、 $\mathcal{E}[e](\mathcal{H}) = \mathcal{E}[e'](\mathcal{H})$.
4. $H \vdash e : S$ なら、 $\mathcal{E}[e](\mathcal{H}) = \mathcal{E}[e|s](\mathcal{H})$.
5. $H \vdash e : U, U > T > S$ なら、 $\mathcal{E}[e|\tau|s](\mathcal{H}) = \mathcal{E}[e|s](\mathcal{H})$.

といった性質を持つ。ただし、 $\mathcal{H}$ は、型環境 H に対して、各変数 x が $\mathcal{A}(\mathcal{H}(x)) = \mathcal{E}^T[H(x)]$ を満たすような環境である。

8. おわりに

本論文では、マージ・オペレータを型および値上に持つ計算を考えた。レコードに対するマージは、レコードの結合を意味し、関数に対するマージは、関数を組み合わせて総称関数を構成することに対応している。マージは、表示の意味を考えた時、上限に対応している。この計算は、関数型言語とオブジェクト指向言語を融合した言語を考えるための基礎となることが期待される。 λ_m の計算に基づく言語を設計するのは、

これからの課題である。

ここでは、**rec** および **if** に関しては、引数の型を固定し、総称関数のように、引数の型変換による適用を許さないようにしたが、これは、ルールを複雑にするのを避けるためで、本質的な欠点ではない。また、ここでは触れなかったが、文献 7) で扱われているような *int* と *real* の継承および、それにより起こる総称関数に対しても、この枠組は適用可能である。

recursion に関しては、文献 10) のように、Y オペレータを各型に定義するのではなく、primitive recursion を用いた。これは、Y により止まらない計算を含めると、停止性と、マージと、2種類の順序を扱う必要が出て、ドメイン上での意味が困難になるためである。Y オペレータとマージ・オペレータを持つ計算の、証明論的な性質および意味論的な性質を研究するのは、将来の課題である。また、オブジェクト指向の型システムの特徴である多相型は、 λ_m では取り扱わなかった。多相型による拡張も、将来の課題である。

謝辞 本研究を進めるにあたり、東京大学の萩谷昌己助教授、京都大学の長谷川立氏、西崎真也氏、沖電気の大堀淳氏との議論は有用でした。また、慶応義塾大学の斉藤信男教授に感謝します。

参 考 文 献

- 1) Bobrow, D., DeMichiel, L., Gabriel, R., Keene, S., Kiczaler, G. and Moon, D.: Common Lisp Object System Specification: X3j13 Document 88-002r, *SIGPLAN Notices*, Vol. 23, Special Issue (1988).
- 2) Cardelli, L.: A Semantics of Multiple Inheritance, *LNCS*, 173, pp. 51-67, Springer-Verlag (1984).
- 3) Cardelli, L. and Mitchell, J.: Operations on Records, *MSCS*, Vol. 1, No. 1, pp. 3-48 (1991).
- 4) Ohori, A. and Buneman, P.: Static Type Inference for Parametric Classes, *Proc., OOPSLA 89: ACM Conf.*, pp. 445-455 (1989).
- 5) Wand, M.: Complete Type Inference for Simple Objects, *Proc., IEEE Symposium on Logic in Computer Science*, pp. 37-44 (1987).
- 6) Bracha, G. and Cook, W.: Mixin-Based Inheritance, *Proc., OOPSLA 90: ACM Conf.*, pp. 303-311 (1990).
- 7) Reynolds, J. C.: Using Category Theory to Design Implicit Conversions and Generic Operators, *LNCS*, 94, Springer-Verlag (1981).
- 8) Coppo, M., Dezani-Ciancaglini, M. and Venneri, B.: Functional Characters of Solvable Terms, *Z. Math. Logik Grundlagen Math.*,

Vol. 27, pp. 45-58 (1981).

- 9) Reynolds, J. C.: The coherence of languages with intersection types, *Theoretical Aspects of Computer Software*, Volume 526 of LNCS, pp. 657-700 (1991).
- 10) Breazu-Tannen, V., Gunter, C. A. and Scedrov, A.: Computing with Coercions, *Proc. ACM Conf. Lisp and Functional Programming*, pp. 44-59 (1990).
- 11) Abadi, M., Cardelli, L., Pierce, B. and Plotkin, G.: Dynamic Typing in a Statically Typed Language, *ACM Trans. Prog. Lang. Syst.*, Vol. 13, No. 2, pp. 237-268 (1991).
- 12) Ohori, A. and Buneman, P.: Type Inference in a Database Programming Language. *Proc. ACM Conf. LISP and Functional Programming*, pp. 174-183 (1988).
- 13) Tsuiki, H.: A Record Calculus with a Merge Operator, PhD thesis, Keio University (1992).
- 14) 立木秀樹: 型継承と高階総称関数の表示的意味論, コンピュータ・ソフトウェア, Vol. 7, No. 4, pp. 60-78 (1990).
- 15) Plotkin, G.: Lcf Considered as a Programming Language, *Theor. Comput. Sci.*, Vol. 5, pp. 223-255 (1977).

付 録

A λ_m の型式と式

型式 (メタ変数 S) は, 次のように定義される.

$$S ::= \text{int} | \text{bool} | \{l : S\} | S_1 \rightarrow S_2 \\ | S_1 \vee S_2 (S_1 \uparrow S_2 \text{ の時}).$$

ここで, $S_1 \uparrow S_2$ は, 次のように定義される.

$$(M\text{-REFL}) \quad S \uparrow S$$

$$(M\text{-LABEL } 1) \quad \{l : S\} \uparrow \{l' : S'\}$$

$$(M\text{-LABEL } 2) \quad \{l : S\} \uparrow \{l' : S'\}$$

$S \uparrow S'$ の時

$$(M\text{-FUN } 1) \quad S_1 \rightarrow S_2 \uparrow S_1' \rightarrow S_2'$$

$S_1 \uparrow S_1'$ の時

$$(M\text{-FUN } 2) \quad S_1 \rightarrow S_2 \uparrow S_1' \rightarrow S_2'$$

$S_1 \uparrow S_1'$ かつ $S_2 \uparrow S_2'$ の時

$$(M\text{-LUB}) \quad (S_1 \vee S_2) \uparrow S_3$$

$S_1 \uparrow S_3$ かつ $S_2 \uparrow S_3$ の時.

型のうち, 基礎型 (メタ変数 N), 関数の型 (メタ変数 F), レコードの型 (メタ変数 A) を, 以下のよう

$$N ::= \text{int} | \text{bool} | N_1 \vee N_2,$$

$$F ::= S_1 \rightarrow S_2 | F_1 \vee F_2,$$

$$A ::= \{l : S\} | A_1 \vee A_2.$$

疑似式 (メタ変数 e) は, 次のように定義される.

$$e ::= x | 0 | \text{succ}(e) | \text{rec}(e_1, e_2, e_3)$$

$$| \text{true} | \text{false} | \text{if}(e_1, e_2, e_3)$$

$$| \lambda x^S. e | e_1 \ e_2 | \{l = e\} | e.l | e \vee e'$$

$l_1 \dots l_n$ がすべて異なる時, $\{l_1 : S_1\} \vee \dots \vee \{l_n : S_n\}$ および $\{l_1 = e_1\} \vee \dots \vee \{l_n = e_n\}$ のことを, それぞれ $\{l_1 : S_1, \dots, l_n : S_n\}$, $\{l_1 = e_1, \dots, l_n = e_n\}$ と書く. $\text{succ}(e)$ は, $e+1$ を表し, $\text{rec}(e_1, e_2, e_3)$ は, primitive recursion を表している. 疑似式の中で, 付録 C の型づけ規則によってある型環境 H に対して型が与えられるものを式とする.

B 型 継 承

型の上の継承関係 $>$ は, 以下のように定義される.

$$(I\text{-ID}) \quad \overline{S > S}$$

$$(I\text{-TRANS}) \quad \frac{S > S' \quad S' > S''}{S > S''}$$

$$(I\text{-RLUB}) \quad \overline{S_1 \vee S_2 > S_1}$$

$$(I\text{-LLUB}) \quad \frac{S > S_1' \quad S > S_2'}{S > S_1' \vee S_2'}$$

$$(I\text{-ASLUB}) \quad \overline{(S_1 \vee S_2) \vee S_3 \approx S_1 \vee (S_2 \vee S_3)}$$

$$(I\text{-COLUB}) \quad \overline{S_1 \vee S_2 \approx S_2 \vee S_1}$$

$$(I\text{-LABEL}) \quad \frac{S > S'}{\{l : S\} > \{l : S'\}}$$

$$(I\text{-LUBL}) \quad \overline{\{l : S\} \vee \{l : S'\} > \{l : (S \vee S')\}}$$

$$(I\text{-FUN}) \quad \frac{S_1 < S_3 \quad S_2 > S_4}{S_1 \rightarrow S_2 > S_3 \rightarrow S_4}$$

$$(I\text{-LUBF}) \quad \frac{S_1 \rightarrow S_1' \vee S_2 \rightarrow S_2' >}{(S_1 \vee S_2) \rightarrow (S_1' \vee S_2')}$$

C 型 づ け 規 則

型環境 (メタ変数 H) とは, $x_1 : S_1, \dots, x_n : S_n$ という形の変数への型づけの列である. 疑似式の型づけ規則を以下のように与える.

$$(T\text{-VAR}) \quad \frac{H \text{ includes } x : S}{H \vdash x : S}$$

$$(T\text{-FUN}) \quad \frac{H, x : S \vdash e : S'}{H \vdash \lambda x^S. e : S \rightarrow S'}$$

$$(T\text{-APP}) \quad \frac{H \vdash e_1 : F \quad H \vdash e_2 : S \quad AP(F, S) \neq \text{null}}{H \vdash e_1 e_2 : AP(F, S)}$$

$$(T\text{-LABEL } 1) \quad \frac{H \vdash e : S}{H \vdash \{l = e\} : \{l : S\}}$$

$$(T\text{-LABEL } 2) \quad \frac{H \vdash e : A \quad A.l \neq \text{null}}{H \vdash e.l : A.l}$$

$$(T\text{-LUB}) \quad \frac{H \vdash e : S \quad H \vdash e' : S'}{H \vdash e \vee e' : S \vee S'}$$

$$\begin{aligned}
(\text{T-ZERO}) \quad & \overline{H \vdash 0 : \text{int}} \\
(\text{T-SUC}) \quad & \frac{H \vdash e : \text{int}}{H \vdash \text{succ}(e) : \text{int}} \\
& H \vdash e_1 : S \quad H \vdash e_2 : S' \rightarrow (\text{int} \rightarrow S'') \\
(\text{T-REC}) \quad & \frac{H \vdash e_3 : \text{int} \quad S \simeq S' \simeq S''}{H \vdash \text{rec}(e_1, e_2, e_3) : S} \\
(\text{T-BOOL}) \quad & \frac{b = \text{true or false}}{H \vdash b : \text{bool}} \\
& H \vdash e : \text{bool} \quad H \vdash e' : S' \quad H \vdash e'' : S'' \\
& S' \simeq S'' \\
(\text{T-IF}) \quad & \frac{}{H \vdash \text{if}(e, e', e'') : S''}
\end{aligned}$$

$AP(F, S)$ と $A.l$ は、次のように定義される。

$$AP(S_1 \vee S_2, S') = AP(S_1, S') \vee AP(S_2, S')$$

$$AP(S \rightarrow S', S'') = S' \quad (S'' > S)$$

$$AP(S \rightarrow S', S'') = \text{null} \quad (S'' \not> S)$$

$$\{l' : S\}.l = \text{null}$$

$$\{l : S\}.l = S$$

$$(S \vee S').l = S.l \vee S'.l$$

ここで、 null は、 $\text{null} \vee S = S$ を満たす特殊な型定数である。

$AP(S, V_1 \vee \dots \vee V_n) \simeq AP(S, V_1) \vee \dots \vee AP(S, V_n)$ とは限らない。すなわ、関数は型の継承関係に関して加法的ではない。

D 簡約規則

正規形 c を、以下のように定義する。

$$n ::= 0 \mid \text{succ}(n) \mid \text{true} \mid \text{false}$$

$$d ::= \lambda x^S. e \mid d \mid_F \mid d_1 \vee \dots \vee d_n$$

$$r ::= \{l_1 = c_1, \dots, l_n = c_n\}$$

$$c ::= n \mid d \mid r \mid \text{nil}$$

また、中間式を、式の定義を nil および $c|s$ で拡張したものとする。正規形は、中間式の部分集合となっている。簡約規則 $e \triangleright c$ を、閉じた中間式 e および、正規形 c に対して定義する。ここで、ほとんど正規形になっている式（正規形のマージおよび、正規形の型変換）を正規形にする規則は、 NF として別に記述する。 e が式の時には、 $e \triangleright c$ となる c は nil ではないことが証明できる。すなわち、 nil は、計算の中間状態として使われるだけであり、式および正規形には現れない。

$$(\text{E-CONST}) \quad \frac{}{c \triangleright c}$$

$$(\text{E-LABEL}) \quad \frac{e \triangleright \{l_1 = c_1, \dots, l_n = c_n\}}{e.l_i \triangleright c_i}$$

$$f \triangleright d_1 \vee \dots \vee d_n \quad d_i e \triangleright c_i (i=1, \dots, n)$$

$$(\text{E-APPL}) \quad \frac{NF(c_1 \vee \dots \vee c_n) = c'}{fe \triangleright c'}$$

$$(\text{E-APP}) \quad \frac{e' : S' \quad S' > S \quad e' \triangleright c \quad e[x := NF(c|s)] \triangleright c'}{(\lambda x^S. e)e' \triangleright c'}$$

$$(\text{E-APPN}) \quad \frac{e' : S' \quad S' \not> S}{(\lambda x^S. e)e' \triangleright \text{nil}}$$

$$(\text{E-APPC}) \quad \frac{e : S \quad fe \triangleright c \quad NF(c|_{AP(F, S)}) = c'}{(f|_F)e \triangleright c'}$$

$$(\text{E-RCD}) \quad \frac{e \triangleright c}{\{l = e\} \triangleright \{l = c\}}$$

$$(\text{E-LUB}) \quad \frac{e_1 \triangleright c_1 \quad e_2 \triangleright c_2 \quad NF(c_1 \vee c_2) = c'}{e_1 \vee e_2 \triangleright c'}$$

$$(\text{E-SUC}) \quad \frac{e \triangleright c}{\text{succ}(e) \triangleright \text{succ}(c)}$$

$$(\text{E-REC 1}) \quad \frac{e \triangleright \text{succ}(c) \quad e_2 \text{rec}(e_1, e_2, c) \triangleright c'}{\text{rec}(e_1, e_2, e) \triangleright c'}$$

$$(\text{E-REC 2}) \quad \frac{e \triangleright 0 \quad e_1 \triangleright c'}{\text{rec}(e_1, e_2, e) \triangleright c'}$$

$$(\text{E-IF 1}) \quad \frac{e \triangleright \text{true} \quad e' \triangleright c}{\text{if}(e, e', e'') \triangleright c}$$

$$(\text{E-IF 2}) \quad \frac{e \triangleright \text{false} \quad e'' \triangleright c}{\text{if}(e, e', e'') \triangleright c}$$

(N-LUBN 2) は、マージ・エラーを起こすが、これは、式の値となるのではなく、例外として、式の簡約全体を終了させる。

$$(\text{N-NIL 1}) \quad \overline{NF(c \vee \text{nil}) = c}$$

$$(\text{N-NIL 2}) \quad \overline{NF(\text{nil} \vee c) = c}$$

$$(\text{N-LUBN 1}) \quad \frac{n_1 = n_2}{NF(n_1 \vee n_2) = n_1}$$

$$(\text{N-LUBN 2}) \quad \frac{n_1 \neq n_2}{NF(n_1 \vee n_2)} \text{ はマージ・エラー.}$$

$$(\text{N-LUBR}) \quad \frac{NF(c_i \vee c'_i) = c''_i (i=1, \dots, m)}{NF(\{l_1 = c_1, \dots, l_m = c_m\} \vee \{l_1 = c'_1, \dots, l_n = c'_n\}) = \{l_1 = c''_1, \dots, l_m = c''_m, \dots, l_n = c'_n\}}$$

$$(\text{N-LUBF}) \quad \overline{NF(d_1 \vee d_2) = d_1 \vee d_2}$$

$$(\text{N-COEN}) \quad \overline{NF(n|_N) = n}$$

$$(\text{N-COER 1}) \quad \frac{NF(c_i|s_i) = c'}{NF(\{l_1 = c_1, \dots, l_n = c_n\} | \{t_i : s_i\}) = \{l_i = c'\}}$$

$$(\text{N-COER 2}) \quad \frac{NF(c|_{A_i}) = c_i \quad NF(c_1 \vee \dots \vee c_n) = c'}{NF(c|_{A_1 \vee \dots \vee A_n}) = c'}$$

$$(\text{N-COEF}) \quad \overline{NF(d|_F) = d|_F}$$

E 例の λ_m での表現

2章であげた例を、 λ_m で記述する。[rectangle]を、

$$[\text{rectangle}] = \{x : \text{integer} \quad ; x \text{ position}$$

```

y: integer      ; y position
w: integer      ; width
h: integer      ; height
}

```

と定義し、各型を、

[Window]=[rectangle].

[Window with frame]=

[Window] $\vee$ {frame: [rectangle]}.

[Window with title]=

[Window] $\vee$

{title: ([rectangle] $\vee$ {content: string})}.

[Window with frame and title]=

[Window with frame] $\vee$ [Window with title].

と定義する。例1の継承関係が成り立つ。[Window with frame and title] は、2つの型がマージ可能なので、定義される。この定義で、各型名は、単に、レコード型の略記である。ウィンドウの移動関数は、[rectangle] に対して（すなわち、[window] に対して）次のように定義される。

$\text{move } 1 = \lambda r^{[\text{rectangle}]} . \lambda dx^{int} . \lambda dy^{int} .$

$\{x = r.x + dx, y = r.y + dy,$

$w = r.w, h = r.h\}$

そして、[Window with frame] と [Window with title] に対して

$\text{move } 2 = \lambda r^{[\text{Window with frame}]} . \lambda dx^{int} . \lambda dy^{int} .$

$\{ \text{frame} = (\text{move } 1 \ r . \text{frame } dx \ dy) \},$

$\text{move } 3 = \lambda r^{[\text{Window with title}]} . \lambda dx^{int} . \lambda dy^{int} .$

$\{ \text{title} = ((\text{move } 1 \ r . \text{title } dx \ dy) \vee$

$\{ \text{content} = r . \text{title} . \text{content} \}) \},$

と定義される。ここで、“+” は、**rec**, **suc** を用いて $int \rightarrow int \rightarrow int$ の関数として、 $\lambda x^{int} . \lambda y^{int} . \text{rec}(x, \lambda z^{int} . \lambda w^{int} . \text{suc}(z, w))$ と定義されている。move を $\text{move } 1 \vee \text{move } 2 \vee \text{move } 3$ とおく。move が [Window with frame] の要素 w に適用されれば、 $(\text{move } 1 \ w) \vee (\text{move } 2 \ w)$ となる。また、[Window with frame and title] の要素 w に適用されれば、 $(\text{move } 1 \ w) \vee (\text{move } 2 \ w) \vee (\text{move } 3 \ w)$ となる。

(平成4年4月22日受付)

(平成5年2月12日採録)



立木 秀樹 (正会員)

1963年生。1986年京都大学理学部卒業。1988年京都大学理学部大学院修士課程(数理解析専攻)修了。同年10月、京都大学数理解析研究所助手。1990年より慶應義塾大学

環境情報学部助手。理学博士。型理論、数理論理学、カテゴリ理論などのプログラム言語の基礎理論に興味を持つ。日本ソフトウェア科学会、ACM各会員。