

A Domain-theoretic Semantics of Lax Generic Functions

Hideki Tsuiki

Department of Fundamental Sciences, Kyoto University, Kyoto 606, Japan
tsuiki@i.h.kyoto-u.ac.jp

Abstract

The semantic structure of a calculus λ_m is studied. λ_m is a polymorphic calculus defined over a hierarchical type structure, and a function in this calculus, called a generic function, can be composed from more than one lambda expression and the ways it behaves on each type are weakly related in that it lax commutes with coercion functions.

Since laxness is intermediate between ad-hocness and coherentness, λ_m has syntactic properties lying between those of calculi with ad-hoc generic functions and coherent generic functions studied in [Tsu95]. That is, though λ_m allows self application and thus is not normalizing, it does not have an unsolvable term. For this reason, all the semantic domains are connected by infinitely many mutually recursive equations and, at the same time, they do not have the least elements. We solve them by considering opfibrations and expressing the equations as one recursive equation about opfibrations. We also show the adequacy theorem for λ_m following the construction of A. Pitts and use it to derive some syntactic properties.

1 Introduction

As defined by Strachey, polymorphism is classified into parametric polymorphism and ad-hoc polymorphism ([Rey83]). A parametric polymorphic function is a function which is defined uniformly over types. That is, though it can be viewed as a collection of monomorphic functions, they have the same algorithm written as a single lambda expression. On the other hand, an ad-hoc polymorphic function is a function whose effects on different argument types are unrelated. These two notions are closely related to the syntactic properties and semantic constructions of calculi. For example, Girard-Reynolds second order lambda calculus ([Gir71],[Rey74]) is strongly normalizing and a parametric polymorphic function is characterized in a model as a function which preserves all the relations between all the types ([Rey83],[Has94], [BMM88]). On the other hand, if a second order calculus has a function which is written by type case, it becomes non-normalizing([Has94],[Gir71]).

The author has studied, motivated by the study of object orientedness, polymorphic functions over a hierarchical type structure ([Tsu94],[Tsu98]). Such a function is called a generic function borrowing the terminology of a programming language CLOS. Generic functions are also classified into coherent ones and ad-hoc ones. A coherent generic function is a function which behaves uniformly with respect to the coercible relations between supertypes and subtypes. It ensures, like parametric polymorphic functions, good syntactic and semantic properties. For example, λ_{\otimes}^p , which is a calculus of coherent generic functions, has a normalizing property, and a coherent generic function is characterized so that it preserves the coercible relations. In other words, when we view the hierarchical type structure as a functor from a poset of types to a suitable category of domains, a coherent generic function can be considered as a natural transformation. On the other hand, λ_{\otimes} , which is a calculus of ad-hoc generic functions, was shown to be non-normalizing and a paradoxical operator like Y was shown to be encodable.

In this paper, we consider yet another class of generic functions. It is called a lax generic function. Though a lax generic function is not defined uniformly, the branches of a function are related in that it lax commutes with coercion functions. In other words, it can be considered as a lax transformation when we view the hierarchical type structure as a functor. This calculus is designed considering a model of object oriented programs causing runtime errors.

Since laxness is intermediate between ad-hocness and coherentness, λ_m , which is a calculus of lax generic functions, has syntactic properties lying between those of the two. That is, though a generic function may be applicable to itself and λ_m is not normalizing, it is shown that every basic type expression is reduced to a constant K or to a form $K \otimes M$ for some term M . That is, every term is representing a value K or K plus something. It means that there is no unsolvable term and that an operator like Y is not expressible in it.

For these reasons, the construction of the semantic domains becomes non-trivial. In order to interpret the hierarchical type structure, we give the semantics as a functor D from a poset \mathcal{T} of types to a suitable category \mathcal{C} of domains. Then, the relations each domain should satisfy can be expressed as the following equation between functors:

$$D = D_{\mathbf{B}} \times [D \xrightarrow{\text{Lax}} D].$$

The semantics of ad-hoc generic functions was given in [Tsu98] by solving a similar equation between functors for the case that \mathcal{C} is the category of pointed cpo's by applying the standard theory of solutions of categorical equations in [SP82]. This equation is also solvable if \mathcal{C} is the category of pointed cpo's. However, since there is no unsolvable term in λ_m , we should assign a non-pointed cpo to each type. The decades of study on domain theory and on axiomatic domain theory shows that the existence of the bottom element plays an essential role in solving a domain equation ([SP82], [Fre91], [Fio94]), and therefore the standard theories are not applicable directly to this problem. In

this study, we solve this by considering an opfibration \mathcal{D} obtained by gathering all the domains, and expressing the above equation as an equation between opfibrations. By considering a simultaneous construction of the poset \mathcal{T} and \mathcal{D} , a construction similar to [SP82] becomes applicable and the equation is solved.

Recently, [Pit94,Pit96] has developed a technique for proving the adequacy of a language with respect to the model constructed over the minimal invariant of a recursive domain equation. We also show the adequacy property with respect to this semantics following this construction. From this adequacy property, many syntactic properties are derived including the non-existence of an unsolvable term and laxness of a generic function.

In the next section, we give motivating examples of lax generic functions from object oriented programming. Then, we introduce lax, coherent, and ad-hoc generic functions over a simple mathematical model in Section 3. After that, we define the calculus λ_m in Section 4 and study its syntactic properties in Section 5. We consider the equations we need to solve in Section 6, and reformulate it over opfibrations in Section 7. The solution is given in Section 8 and then the semantics is given in Section 9. Finally, the adequacy property is studied in Section 10.

2 Laxness of Methods and Object Oriented Programming

In object oriented programming, one can re-define the behavior of a subclass to a message by overriding the supertype definitions. This overriding is so powerful that one can re-define the behavior of a subclass to a message completely different from those of superclasses. For example, when “Bus” is a subclass of “Car”, one can write the method “move” for the class “Bus” as moving a bus backward, while “move” for the class “Car” moves a car forward. This kind of program is usually difficult to read or maintain, and more prone to error. Therefore, it is implicit in object oriented programming that methods with the same name should be programmed to have *the same kind of meaning*.

Then, a question arises when a set of methods can be said to have the same kind of meaning. Since we are interested in properties the collection of all the methods with the same name have, we adopt the notion of a generic function, which is a function composed by all the methods with the same name. Then an answer to the above question is given as a property of a generic function. For fundamental studies of object orientedness via generic functions, see [Tsu98] and [CGL95].

As such a property, [Tsu94] defined and studied the notion of a coherent generic function, which is defined such that coercions and applications of the generic function commute (see Figure 1), or in other words, a generic function preserves the coercible order relation. The notion of a coherent generic

function is mathematically sound in that it corresponds to being a natural transformation as we explain in the next section, and was already used in [Rey81] to give a semantics to an overloaded operator.

In this paper, we consider a weaker condition. That is, coercions and applications of the generic function *lax* commutes (see Figure 1). We explain how this condition is related with object oriented programming in three ways.

The first one is about a generic function causing runtime errors or exceptions. Error handling is important in particular in object oriented programming because a class may be reused in various context, and most programming languages have error handlers like the catch and throw mechanism of Java [GJGL96]. It seems plausible that if a message sent to “an object considered as a superclass object” causes an error, then the same message sent to the object also causes an error, but not vice versa. As an example, consider the “Car” and “Bus” example with a limit speed defined for a “Bus”. Then, a method “set-speed” in “Bus” may cause a runtime error if the argument exceeds the limit, where it does not cause an error if it is considered as a car. As a more illustrative example, consider a “computer” class and a “network computer” subclass. The initialization step on a “network computer” class may cause an error like network configuration error, whereas a “computer” does not. In general, a subclass is more likely to produce a runtime error because it has a more sophisticated structure which may cause an unexpected status. Though coercion and application do not commute in these cases, they are expected to lax commute in the sense that if we consider an order structure on each type with the top element a special value denoting runtime error, then “coercion after application” is bigger than “application after coercion” by the order on the result type.

The second one is about the view that the collection of all the values of all the types constitute one large domain. Since a generic function is applicable to more than one types, it is natural to consider one large domain \mathcal{D} which consists of all the domains of all the types, and consider a generic function as one (partial) function from \mathcal{D} to \mathcal{D} . If we do not have an order structure on each domain, then the natural order structure on \mathcal{D} is the coercible order. However, if each domain has an order structure, then the order structure on \mathcal{D} should be formed as the mixture of the coercible order and the order structures on each domain, which is formed as a simple example of an opfibration. Then, it is natural to consider the property that a generic function preserves this order structure, or in other words, returns more informative value to more informative argument, if the order structure on \mathcal{D} is considered as an information order. This condition actually coincides with the laxness, as we shall see in Section 7.

The third one is about the semantics of non-coherent generic functions in object oriented languages. We will discuss this problem in the Conclusion.

3 Coherent, Lax, and Ad-hoc Generic Functions

We formalize mathematically the intuitive idea in the previous section. We consider a poset $(\mathcal{T}, \sqsupseteq)$ of types with subtype relations. We write $s \sqsupseteq t$ when s is a subtype of t . Note that this order is opposite to the one usually used. Though the usual order is natural when we view it as set inclusion, the opposite order fits very well with the domain theory we develop in this paper; we view the subtype relation not as set inclusion but as the coercible relation and, through this view, a value of a subtype becomes more informative than its coercion to a supertype. Though the poset $(\mathcal{T}, \sqsupseteq)$ is the particular one corresponding to the type structure of λ_m given in 4.1, one may consider it as any poset in this section.

For each $t \in \mathcal{T}$, we consider a domain $D(t)$ of values of type t . In addition, when $s \sqsupseteq t$, we consider a coercion function $\mathbf{coerce}_{s,t}$ from $D(s)$ to $D(t)$ such that $\mathbf{coerce}_{s,s}$ is the identity, and $\mathbf{coerce}_{t,u} \circ \mathbf{coerce}_{s,t} = \mathbf{coerce}_{s,u}$. Thus, for a suitable category \mathcal{C} of domains, D is a functor from the poset \mathcal{T} considered as a category to \mathcal{C} . In order to make the presentation simpler, we also suppose that \mathcal{T} has a least element $\perp_{\mathcal{T}}$ corresponding to type error, and that $D(\perp_{\mathcal{T}})$ is the terminal object $\mathbf{1}$ of \mathcal{C} which is the one-point domain $\{\perp_{\mathcal{D}}\}$.

Over this type structure, we define a generic function to be a collection of monomorphic functions $m = \{m_t : D(t) \rightarrow D(F(t)) \mid t \in \mathcal{T}\}$. Here, F is a monotonic function from \mathcal{T} to \mathcal{T} , mapping the argument type to the result type. We call F the type of the generic function m . Though a generic function is not, in general, applicable to all the types, by defining $D(F(t))$ to be $\mathbf{1}$ and m_t to be the terminal arrow when m is not applicable to t , we can consider that t ranges over \mathcal{T} as in this definition.

Now, we consider three classes of generic functions as in Figure 1.

The first one is that coercions and application of a generic function commute. In other words, m is a natural transformation from the functor D to $D \circ F$. We call such a generic function a coherent generic function. Another is that we do not impose any condition on m . We call such a generic function an ad-hoc generic function to emphasize this fact. The author has studied and compared calculi with ad-hoc and coherent generic functions in [Tsu94] and [Tsu98].

In this paper, we consider yet another condition. Consider that each object C of \mathcal{C} has an order structure (\preceq_C) . For example, we consider a co-flat poset like $\begin{array}{c} \top^{\mathbf{Int}} \\ \swarrow \downarrow \searrow \\ 1 \quad 2 \quad 3 \dots \end{array}$ for a basic type with pointwise extension to functional types. The order $e \preceq_C f$ intuitively means that f is more informative than e . The element \top^C of $D(C)$ is the overdefined element representing conflicting information. Accordingly, we consider that each component m_t of a generic function m is monotonic.

Then, we consider the condition that coercions and generic function applications lax commute with respect to the order on a domain. That is, when

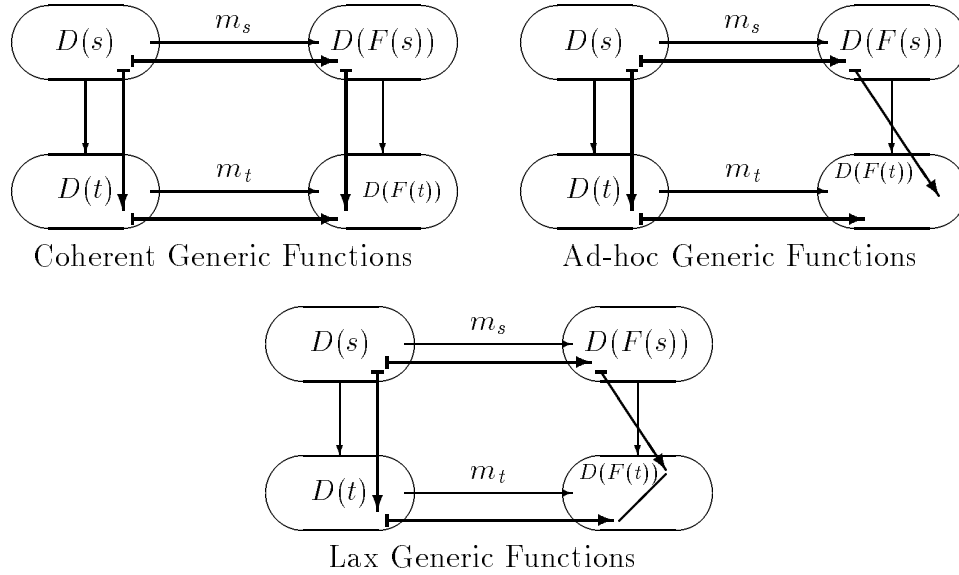


Fig. 1. Coherent, lax, and ad-hoc generic functions. A supertype is written below a subtype in these figures.

$x : s$ and $s \sqsupseteq t$, $m_t(\mathbf{coerce}_{s,t}(x)) \preceq_{D(F(t))} \mathbf{coerce}_{F(s),F(t)}(m_s(x))$. This condition can be stated using a 2-categorical term that f is a lax transformation from D to $D \circ F$ by considering \mathcal{C} as a two category with a two cell between $f, g : C \rightarrow C'$ being the pointwise order relation. We call such a generic function a lax generic function.

4 The Calculus λ_m

4.1 The Poset \mathcal{T} of Types

Since the type of a term needs to be calculated statically, we only consider generic function types which are finite functions from \mathcal{T} to \mathcal{T} . That is, a generic function type is expressed as the least upper bound of step functions, where a step function $\text{Step}(s, t)$ for finite elements s and t maps arguments bigger than s to t and other arguments to $\perp_{\mathcal{T}}$. We write $[\mathcal{I} \xrightarrow{\text{Fin}} \mathcal{I}']$ for the poset of finite functions from a poset \mathcal{I} to \mathcal{I}' .

Since a lax generic function is also treated as a first class value in λ_m , we need to consider a circular structure on \mathcal{T} . Suppose that $\mathcal{T}_{\mathbf{B}}$: the finite flat poset of basic types is given. For example, $\mathcal{T}_{\mathbf{B}}$ has the form $\begin{array}{c} \overline{\text{Int}} \quad \overline{\text{Bool}} \\ \searrow \quad \swarrow \\ \perp \end{array}$. Then,

the circularity we must consider is expressed as follows:

$$(1) \quad \begin{array}{ccc} & \xrightarrow{\Phi^{\mathcal{T}}} & \\ \mathcal{T} & \xrightarrow{\quad} & \mathcal{T}_{\mathbf{B}} + [\mathcal{T} \xrightarrow{\text{Fin}} \mathcal{T}] \\ & \xleftarrow{\Psi^{\mathcal{T}}} & \end{array}$$

Here, $+$ is the smashed sum which identifies the least element.

We can solve this equation algebraically. Consider the following sequence of posets:

$$\begin{aligned} \mathcal{T}_0 &= \{\perp_{\mathcal{T}}\}, \\ \mathcal{T}_1 &= \mathcal{T}_{\mathbf{B}} + [\mathcal{T}_0 \xrightarrow{\text{Fin}} \mathcal{T}_0], \\ &\dots \\ \mathcal{T}_{n+1} &= \mathcal{T}_{\mathbf{B}} + [\mathcal{T}_n \xrightarrow{\text{Fin}} \mathcal{T}_n], \\ &\dots \end{aligned}$$

There are embedding-projection pairs $\mathcal{T}_n \xrightleftharpoons[\psi_n^{\mathcal{T}}]{\phi_n^{\mathcal{T}}} \mathcal{T}_{n+1}$ in the standard way.

Define \mathcal{T} as the colimit of $\phi_n^{\mathcal{T}}$. Then, \mathcal{T} satisfies (1). \mathcal{T} becomes a consistently complete poset with only finite elements. From now on, we denote by \mathcal{T} this particular poset and we write $\mathcal{T}_{\mathbf{F}}$ for the poset $[\mathcal{T} \xrightarrow{\text{Fin}} \mathcal{T}]$. We write $s \uparrow t$ when s and t have an upper bound, and write $s \bowtie t$ for the least upper bound of s and t .

This poset reflects the structure of the set of type expressions of λ_m in that \mathcal{T} is isomorphic to the set of type expressions modulo equivalence plus the bottom element as we will see below. When V is a type expression, we write \overline{V} for the corresponding element in \mathcal{T} .

The type expressions of λ_m are defined as follows. Suppose that a finite set of basic types like **Int** and **Bool** (ranged over by B) is given. We define pretypes (ranged over by U and V) as follows:

$$V ::= B \mid \overbrace{[V \rightarrow V, \dots, V \rightarrow V]}^{\text{at least one}}.$$

$[V_1 \rightarrow V'_1, \dots, V_n \rightarrow V'_n]$ denotes the least upper bound of the step functions $\text{Step}(\overline{V_1}, \overline{V'_1}), \dots, \text{Step}(\overline{V_n}, \overline{V'_n})$. Among pretypes, we define those appropriate as finite functions as types. For instance, $[\mathbf{Int} \rightarrow \mathbf{Int}, \mathbf{Bool} \rightarrow \mathbf{Int}]$ is a type but $[\mathbf{Int} \rightarrow \mathbf{Int}, \mathbf{Int} \rightarrow \mathbf{Bool}]$ is not, because the step functions $\text{Step}(\overline{\mathbf{Int}}, \overline{\mathbf{Int}})$ and $\text{Step}(\overline{\mathbf{Int}}, \overline{\mathbf{Bool}})$ do not have a least upper bound. We also define syntactically the relations $U \uparrow V$ meaning that U and V are compatible, and $U \leq V$ meaning that U is a subtype of V , and an abbreviation $U \bowtie V$ for the greatest lower bound type of U and V . The definitions are given in the Appendix, and the proofs that the poset of type expressions modulo equivalence extended with the bottom element is isomorphic to \mathcal{T} , that $U \leq V$ is decidable, that $U \uparrow V$ iff $\overline{U} \uparrow \overline{V}$, that $U \leq V$ iff $\overline{U} \sqsubseteq \overline{V}$, and that $\overline{U \bowtie V} = \overline{U} \bowtie \overline{V}$ are given in [Tsu98].

We only define here the following syntactic notions.

Definition 4.1 *Let F be $[V_1 \rightarrow V'_1, \dots, V_n \rightarrow V'_n]$.*

1) *We say that F is applicable to U if at least one of the V_i ($i = 1, \dots, n$) satisfies $U \leq V_i$.*

2) When F is applicable to U , define $\text{cod}(F, U)$ as $V'_{\phi(1)} \bowtie \dots \bowtie V'_{\phi(l)}$, where $\phi(i)$ ($i = 1, \dots, l$) satisfy $U \leq V_{\phi(i)}$. Note that $\overline{\text{cod}(F, U)}$ is $\Psi^T(\overline{F})(\overline{U})$.

One thing to note about \mathcal{T} is that though a generic function type may be applicable to itself, it cannot return itself. We define the degree of a type as follows:

$$\text{degree}(B) = 1,$$

$$\text{degree}([V_1 \rightarrow V'_1, \dots, V_n \rightarrow V'_n]) = \max(\text{degree}(V'_1), \dots, \text{degree}(V'_n)) + 1.$$

We also define the degree of a term as the degree of its type. The degree of a term expresses the maximal number of arguments applicable to it. For example, when the degree of M is n , there are no terms N_1, \dots, N_n which make $M N_1 N_2 \dots N_n$ well typed. This fact also supports the use of operational equivalence instead of bisimulation to compare terms in Section 10.

4.2 Terms of λ_m

Before presenting the definition of terms, we give the fundamental idea of terms of λ_m , in particular, the way we define a lax generic function. A lax generic function is composed using the merge operator \otimes , which takes two terms of compatible types V_1 and V_2 , and produces a term of type $V_1 \bowtie V_2$. First, a monomorphic function from V to V' is identified with a generic function of type $[V \rightarrow V']$, which is applicable to subtypes of V through coercions. And then, when $M_1 : F_1$ and $M_2 : F_2$ are lax generic functions with compatible types, then $M_1 \otimes M_2$ becomes a lax generic function of type $F_1 \bowtie F_2$.

In this first step, the argument is coerced to V and then the function is applied. For this purpose, we add an expression $M|_V$ denoting the coercion of M to V . For the second step, we intuitively give the meaning of merge of terms inductively on their degrees. For a basic type B , we define that the merge of two terms denoting the same value is itself and the merge of two terms denoting different values is \mathbf{Top}^B , the constant denoting $\top^{\overline{B}}$. Note that a basic type is compatible only with itself and therefore there is no term like $1 \otimes \text{true}$. For generic function types, a lax generic function $M_1 \otimes M_2$, when applied to an argument, activates M_1 if M_1 is applicable, activates M_2 if M_2 is applicable, and activates both and *merges* the results if both are applicable. Note that the degree of the result is smaller than the maximum of the degrees of M_1 and M_2 , and thus the meaning of \otimes is well defined.

We define the pre-terms of λ_m as follows:

$$M ::= K^B \mid x^V \mid \lambda x^V. M \mid M N \mid M|_V \mid M_1 \otimes M_2$$

Here, K^B denotes a constant of basic type B and x^V means a variable of type V . K^B includes a special constant \mathbf{Top}^B for each basic type B . The typing rules of λ_m are defined as follows:

$$\begin{array}{l}
 \text{(T-CONST)} \frac{}{K^B : B} \qquad \text{(T-FUN)} \frac{M : V'}{\lambda x^V. M : [V \rightarrow V']} \\
 \text{(T-VAR)} \frac{}{x^V : V} \qquad \text{(T-APP)} \frac{M : F \quad N : V \quad F \text{ applicable to } V}{M N : \text{cod}(F, V)} \\
 \text{(T-COE)} \frac{M : U \quad U \leq V}{M|_V : V} \qquad \text{(T-MERGE)} \frac{M : V \quad M' : V' \quad V \uparrow V'}{M \otimes M' : V \bowtie V'}
 \end{array}$$

We further define a pre-term to be a term if it is typable. The type of a term is uniquely defined by this type system. The meanings of these terms are determined by the following reduction rules:

$$\begin{array}{l}
 \text{(E-APP)} \quad (\lambda x^V. M) N \triangleright M[x^V := N|_V] \\
 \text{(E-APPL)} \quad (M_1 \otimes M_2) N \triangleright \left. \begin{array}{l} M_1 N \quad (F_1 \text{ is applicable to } V) \\ M_2 N \quad (F_2 \text{ is applicable to } V) \\ M_1 N \otimes M_2 N \quad (\text{both } F_1 \text{ and } F_2 \text{ are applicable to } V) \end{array} \right\} \\
 \qquad \qquad \qquad (M_1 : F_1, M_2 : F_2, \text{ and } N : V) \\
 \text{(E-CONST1)} \quad K^B \otimes K'^B \triangleright \mathbf{Top}^B \quad (K^B \text{ and } K'^B \text{ are different constants.}) \\
 \text{(E-CONST2)} \quad K^B \otimes K^B \triangleright K^B \\
 \text{(E-CONST3)} \quad \mathbf{Top}^B \otimes M \triangleright \mathbf{Top}^B \\
 \text{(E-COE)} \quad M|_B \triangleright M \quad (B \text{ is a basic type}) \\
 \text{(E-APPC)} \quad (M|_F) N \triangleright (M N)|_{\text{cod}(F, V)} \quad (V \text{ is the type of } N) \\
 \text{(E-COMM)} \quad M \otimes N \triangleright N \otimes M \quad (M \text{ and } N \text{ have the same basic type}) \\
 \text{(E-ASSO)} \quad (M_1 \otimes M_2) \otimes M_3 \triangleright M_1 \otimes (M_2 \otimes M_3) \\
 \qquad \qquad \qquad (M_1, M_2, \text{ and } M_3 \text{ have the same basic type})
 \end{array}$$

As we explained before, $\lambda x^V. M$ denotes a generic function applicable to subtypes of V through the coercion functions. It is realized by inserting a coercion to the argument in (E-APP), the β -reduction. The rules (E-APPL), (E-CONST1), (E-CONST2), and (E-CONST3) determine the meaning of the merge operator, and reflect the intuitive explanation at the beginning of this section. (E-APPC) determines the meaning of coercion between generic functions, and corresponds to pointwise coercion as we will see in Section 9. (E-COMM) and (E-ASSO) are not essential; they are added only to express the syntactic properties simpler.

Compared with λ_{\otimes} : a calculus with ad-hoc generic functions in [Tsu98], the only differences are (E-CONST1) to (E-CONST3). In λ_{\otimes} , the value on the right hand side is given the higher priority. Therefore, in λ_{\otimes} , we do not have the term \mathbf{Top}^B and we have the following rule instead of these three,

(E-CONST0) $M \otimes N \triangleright N$ (M and N have the same basic type).

(E-ASSO) and (E-COMM) are also eliminated in λ_{\otimes} because the order of arguments to \otimes is important.

5 Syntactic Properties of λ_m

We can prove, by checking the rules, that λ_m has the unicity of type and the subject reduction properties. We can also prove the Church-Rosser property using parallel reduction ([Tak95]).

However, λ_m is not weak normalizing; we can form a non-normalizing term on each type. A non-normalizing term of type \mathbf{Int} is given as follows. Let $\mathbf{S} = [\mathbf{Int} \rightarrow \mathbf{Int}, [\mathbf{Int} \rightarrow \mathbf{Int}] \rightarrow \mathbf{Int}]$. Then, a term of type \mathbf{S} is applicable to itself with the result type \mathbf{Int} . Therefore, we can define $M = (\lambda x^{\mathbf{Int}}.x) \otimes (\lambda x^{[\mathbf{Int} \rightarrow \mathbf{Int}]} .1) \otimes (\lambda x^{\mathbf{S}}.x x)$. The type of M is $\mathbf{T} = [\mathbf{Int} \rightarrow \mathbf{Int}, [\mathbf{Int} \rightarrow \mathbf{Int}] \rightarrow \mathbf{Int}, \mathbf{S} \rightarrow \mathbf{Int}]$. Note that $\mathbf{T} \simeq \mathbf{S}$; we have $\mathbf{T} \leq \mathbf{S}$ immediately, and we have $\mathbf{S} \leq [\mathbf{Int} \rightarrow \mathbf{Int}]$, $[[\mathbf{Int} \rightarrow \mathbf{Int}] \rightarrow \mathbf{Int}] \leq [\mathbf{S} \rightarrow \mathbf{Int}]$, and thus $\mathbf{S} \leq \mathbf{T}$. Therefore, M is applicable to itself and $M M$ has type \mathbf{Int} . $M M$ reduces infinitely as follows.

$$\begin{aligned}
M M &= ((\lambda x^{\mathbf{Int}}.x) \otimes (\lambda x^{[\mathbf{Int} \rightarrow \mathbf{Int}]} .1) \otimes (\lambda x^{\mathbf{S}}.x x)) M \\
&\triangleright^+ (\lambda x^{[\mathbf{Int} \rightarrow \mathbf{Int}]} .1) M \otimes (\lambda x^{\mathbf{S}}.x x) M \\
&\triangleright^+ 1 \otimes M|_{\mathbf{S}} M|_{\mathbf{S}} \\
&\triangleright 1 \otimes (M M)|_{\mathbf{S}}|_{\mathbf{Int}} \\
&\triangleright 1 \otimes M M|_{\mathbf{S}} \\
&\triangleright^+ 1 \otimes 1 \otimes M|_{\mathbf{S}}|_{\mathbf{S}} M|_{\mathbf{S}}|_{\mathbf{S}} \\
&\dots
\end{aligned}$$

Here, \triangleright^+ means one or more reduction steps. A non-normalizing term can be formed in any type in a similar way. The existence of a non-normalizing term is connected, in many calculus, to the expressiveness of the paradoxical operator and the existence of an unsolvable term. However, the situation is different in this calculus. We can show the following:

Theorem 5.1 *A term M of a basic type is reduced to K^B or $K^B \otimes M'$ for some constant K^B .*

The syntactic proof of this theorem is rather long, relating a reduction in λ_m with a reduction in another calculus, and we omit it here. Instead, this theorem is derived, in Section 10, as a corollary to the adequacy property of our semantics.

This theorem shows that though non-normalizing terms exist, every term of a basic type has a meaning bigger than some constant K^B , and therefore,

there is no unsolvable term. It also shows that a fixpoint operator like Y does not exist because, if it did, we could form an unsolvable term $Y(\lambda x.x)$. As for a generic function type F , there is no unsolvable term either, because every term of type F forms, when enough arguments are added, a basic type term, which is not unsolvable.

For comparison, this same term $M M$ is reduced in λ_{\otimes} as follows: $M M \triangleright^+ K^B \otimes M |_{\mathbf{s}} M |_{\mathbf{s}} \triangleright^+ M |_{\mathbf{s}} |_{\mathbf{s}} M |_{\mathbf{s}} |_{\mathbf{s}} \triangleright^+ \dots$. That is, by the reduction rule (E-CONST0), the information that it is bigger than K^B is lost. Thus, it becomes an unsolvable term in λ_{\otimes} .

6 A Construction in a Functor Category

In the rest of this work, we give a denotational semantics to λ_m . Since λ_m has a hierarchical type structure, we need to construct domains which are connected by coercion functions. Therefore, we construct a functor D from \mathcal{T} to a suitable category \mathcal{C} as the semantics of λ_m . Usually, a domain equation is solved in the category of pointed cpo's. However, as we have shown, each type of λ_m does not have an unsolvable term. Therefore, it is not appropriate to construct a domain with the bottom element as the interpretation of a type. Thus, we consider a construction with the category of (not always pointed) cpo's for \mathcal{C} .

For a basic type B , we define $D(\overline{B})$ to be the co-flat poset of constants of B like $\begin{array}{c} \overline{\mathbf{Int}} \\ \swarrow \downarrow \searrow \\ 1 \quad 2 \quad 3 \dots \end{array}$. Note that $D(\overline{B})$ does not have a least element.

For a generic function type F , the natural interpretation of $D(\overline{F})$ is the set of lax transformations from D to $D \circ \Psi^{\mathcal{T}}(\overline{F})$ with the order of $D(\overline{F})$ defined pointwisely. Therefore, we construct a functor D so that

$$(2) \quad D(\overline{F}) = \{f \in \Pi_{s \in \mathcal{T}} D(\Psi^{\mathcal{T}}(\overline{F})(s))^{D(s)} \mid f \text{ lax commutes with coercions}\}$$

Note that it is not a definition but an equation that D must satisfy because s ranges over \overline{F} . Thus, we have an infinite number of equations between infinitely many domains $\{D(s) \mid s \in \mathcal{T}\}$.

These equations can be expressed as one equation between functors. We write $\mathbf{Fun}(\mathcal{T}, \mathcal{C})$ for the functor category from \mathcal{T} to \mathcal{C} . We define a functor $[- \xrightarrow{\text{Lax}} -]$ from $\mathbf{Fun}(\mathcal{T}, \mathcal{C})^{op} \times \mathbf{Fun}(\mathcal{T}, \mathcal{C})$ to $\mathbf{Fun}(\mathcal{T}, \mathcal{C})$ as follows:

$$\begin{aligned} [E \xrightarrow{\text{Lax}} E'](t) &= \mathbf{1} && (t \in \mathcal{T}_{\mathbf{B}}), \\ [E \xrightarrow{\text{Lax}} E'](t) &= \{f \in \Pi_{s \in \mathcal{T}} E'(\Psi^{\mathcal{T}}(t)(s))^{E(s)} \mid f \text{ lax commutes with coercions}\} \\ &&& (t \in \mathcal{T}_{\mathbf{F}}, t \neq \perp). \end{aligned}$$

The condition “ f lax commutes with coercions” can also be expressed as

follows:

$$E'(\Psi^{\mathcal{T}}(t)(s_1) \sqsupseteq \Psi^{\mathcal{T}}(t)(s_2))(f_{s_1}(x)) \preceq_{E'(\Psi^{\mathcal{T}}(t)(s_2))} f_{s_2}(E(s_1 \sqsupseteq s_2)(x))$$

and for all $s_1, s_2 \in \mathcal{T}$ and $x \in E(s_1)$.

The operation of $[E \xrightarrow{\text{Lax}} E']$ on morphisms is as follows:

$$\begin{aligned} [E \xrightarrow{\text{Lax}} E'](t \sqsupseteq t') &= id_{\mathbf{1}} && (t, t' \in \mathcal{T}_{\mathbf{B}}), \\ [E \xrightarrow{\text{Lax}} E'](t \sqsupseteq t') &= \lambda f \in \prod_{s \in \mathcal{T}} E'(\Psi^{\mathcal{T}}(t)(s))^{E(s)}. \\ &\langle s \in \mathcal{T}. E'(\Psi^{\mathcal{T}}(t)(s)) \sqsupseteq \Psi^{\mathcal{T}}(t')(s) \circ f_s \rangle && (t, t' \in \mathcal{T}_{\mathbf{F}}). \end{aligned}$$

Here, $\langle s \in \mathcal{T}. f_s \rangle$ denotes an infinite product and f_s is the projection of f to the s part. The operation of the functor $[- \xrightarrow{\text{Lax}} -]$ on morphisms (ie. natural transformations) can also be defined naturally.

Let $D_{\mathbf{B}} \in \mathbf{Fun}(\mathcal{T}, \mathcal{C})$ be the following functor:

$$\begin{aligned} D_{\mathbf{B}}(t) &= \text{the co-flat poset of constants of type } t \quad (t \in \mathcal{T}_{\mathbf{B}}, t \neq \perp), \\ D_{\mathbf{B}}(t) &= \mathbf{1} && (t \in \mathcal{T}_{\mathbf{F}}). \end{aligned}$$

Then, the equations we need to solve can be expressed as the following equation between functors:

$$(3) \quad E = D_{\mathbf{B}} \times [E \xrightarrow{\text{Lax}} E].$$

Here, the product of functors is defined pointwisely. Note that only one component of \times is not $\mathbf{1}$ for each $t \in \mathcal{T}$ in (3). When D satisfies (3), then D satisfies (2), and thus we can give semantics to λ_m .

[SP82] gives sufficient conditions under which such a categorical equation is solvable. Roughly, the condition says that the category is enriched with pointed ω -cpo's and the functor is a locally continuous functor. In [Tsu98], the author has solved, to give a domain-theoretic semantics to λ_{\otimes} , the equation

$$(4) \quad E = D_{\mathbf{B}} \times [E \Rightarrow E]$$

in $\mathbf{Fun}(\mathcal{T}, \mathcal{C})$ with \mathcal{C} the category of pointed ω -cpo's and strict continuous functions, and $[- \Rightarrow -]$ being a functor forming the ad-hoc generic function space. That is, $[E \Rightarrow E'](t)$ is defined to be $\{f \in \prod_{s \in \mathcal{T}} E'(\Psi^{\mathcal{T}}(t)(s))^{E(s)}\}$ when $t \in \mathcal{T}_{\mathbf{F}}$. For the case of (3), if we use the category of pointed cpo's with only strict morphisms for \mathcal{C} , then the conditions of [SP82] are satisfied and (3) is solvable in $\mathbf{Fun}(\mathcal{T}, \mathcal{C})$. Over the solution, we can give a semantics to λ_m . However, since an object of \mathcal{C} is pointed, $D(\mathbf{Int})$ needs to have the

form $\begin{array}{c} \mathbf{Int} \\ \swarrow \downarrow \searrow \\ 1 \quad 2 \quad 3 \dots \\ \swarrow \downarrow \searrow \\ \perp \quad \mathbf{Int} \end{array}$ instead of $\begin{array}{c} \mathbf{Int} \\ \swarrow \downarrow \searrow \\ 1 \quad 2 \quad 3 \dots \end{array}$. Since there is no term whose meaning

is $\perp^{\mathbf{Int}}$, this semantics does not reflect the structure of λ_m , and is far from sufficient. However, in standard theories, the existence of the bottom element plays an essential role in solving a domain equation ([SP82], [Fre91], [Fio94]), and therefore they are not applicable directly to this problem.

We solve this problem by considering a pair (\mathcal{I}, E) of a poset \mathcal{I} and a functor E from \mathcal{I} to \mathcal{C} as an object and construct both \mathcal{T} and D simultaneously. Then, the pair consisting of the one point poset for \mathcal{I} and one point cpo for the image of E becomes the least element. Instead of constructing such a pair, we construct an opfibration satisfying an equation equivalent to (3). It makes the presentation simpler, and enables the proof of the adequacy theorem in Section 10.

7 A Construction Over Opfibrations

In this section, we shall consider that all the values of all the types constitute a large domain \mathcal{D} and that all the components of a lax generic function form one function from \mathcal{D} to \mathcal{D} . It is known that we can construct a split opfibration from an indexed category by the Grothendieck construction. (see [Pho92], [BW90], or [Jac99] for references.) Since we can view each cpo as a category, we can apply this construction to $\mathbf{Fun}(\mathcal{T}, \mathcal{C})$. We give here the definition of an opfibration for the case that the base space and the target spaces are both posets. Note that all the opfibrations split in this case.

Definition 7.1 *An opfibration is a monotonic function \mathcal{A} from a poset (\mathcal{E}, \preceq) to another poset $(\mathcal{I}, \sqsubseteq)$ such that, for $x \in \mathcal{E}$ and $s \sqsubseteq \mathcal{A}(x)$, there exists $y \in \mathcal{E}$ such that $y \preceq x$ and $\mathcal{A}(y) = s$ with the following universality: for any $z \preceq x$ such that $\mathcal{A}(z) \sqsubseteq s$, we have $z \preceq y$. We write $x|_s$ for this y .*

We call $x|_s$ the coercion of x to s . When $\mathcal{A}(x) = t$, we say that the type of x is t .

Proposition 7.2 *Let \mathcal{I} be a poset and E be a functor from \mathcal{I} to \mathbf{POSET} . Then, we define a poset $\mathcal{E} = \bigcup_{t \in \mathcal{I}} E(t)$ of disjoint union of $\{E(t) \mid t \in \mathcal{I}\}$ with the order relation \preceq defined as $(t, x) \preceq (s, y)$ iff $t \sqsubseteq s$ and $x \preceq_{E(t)} E(t \sqsubseteq s)(y)$. Let \mathcal{A} to be the first projection from \mathcal{E} to \mathcal{I} . Then, $\mathcal{A} : \mathcal{E} \rightarrow \mathcal{I}$ is an opfibration.*

Note that the opfibration constructed in this way from the solution D of (3) has a least element because \mathcal{T} has a least element $\perp_{\mathcal{T}}$ and $D(\perp_{\mathcal{T}})$ is the one-point poset. However, \mathcal{E} is not a cpo because there can be an ω -chain $(t_1, d_1) \preceq (t_2, d_2) \preceq \dots$ with $t_1 \sqsubseteq t_2 \sqsubseteq \dots$ so that this ω -chain does not have a least upper bound in \mathcal{T} .

Conversely, when an opfibration $\mathcal{A} : \mathcal{E} \rightarrow \mathcal{I}$ is given, we can form a functor D from \mathcal{I} to \mathbf{POSET} . In particular, if $\mathcal{A}^{-1}(t)$ is a cpo for every $t \in \mathcal{I}$ and coercion functions are continuous, then D is a functor to \mathbf{CPO} . Therefore, we construct an opfibration $\mathcal{A} : \mathcal{D} \rightarrow \mathcal{T}$ with this property instead of $D : \mathcal{T} \rightarrow \mathbf{CPO}$.

Definition 7.3 We call a triple $\mathcal{O} = (\mathcal{E}, \mathcal{A}, \mathcal{I})$ an *M-domain* if $\mathcal{A} : \mathcal{E} \rightarrow \mathcal{I}$ is an opfibration, \mathcal{A} is surjective, $\mathcal{A}^{-1}(t)$ is a cpo for every $t \in \mathcal{I}$, $\downarrow_s : \mathcal{A}^{-1}(t) \rightarrow \mathcal{A}^{-1}(s)$ is continuous for $s \sqsubseteq t$, \mathcal{I} has the least element $\perp_{\mathcal{I}}$, and $\mathcal{A}^{-1}(\perp_{\mathcal{I}}) = \{\perp_{\mathcal{E}}\}$ is the one-point cpo.

Suppose that $\mathcal{O} = (\mathcal{D}, \mathcal{A}, \mathcal{T})$ is an M-domain constructed from the solution D of (3) by Proposition 7.2. Then, a generic function m of type F defines a function $m^\#$ from \mathcal{D} to \mathcal{D} , which satisfies $\mathcal{A} \circ m^\# = F \circ \mathcal{A}$. One of the benefits of considering opfibrations instead of indexed categories is that the characterization of a lax generic function is simplified as follows:

Proposition 7.4 A generic function m lax commutes with coercions iff $m^\#$ is a monotone function.

Regarding the requirement that each component of a generic function should be continuous, we give the following definition:

Definition 7.5 Let $\mathcal{O} = (\mathcal{E}, \mathcal{A}, \mathcal{I})$ and $\mathcal{O}' = (\mathcal{E}', \mathcal{A}', \mathcal{I}')$ be M-domains. A morphism from \mathcal{O} to \mathcal{O}' is a pair (F, f) where $f : \mathcal{E} \rightarrow \mathcal{E}'$ is a continuous function and $F : \mathcal{I} \rightarrow \mathcal{I}'$ is a finite function such that $\mathcal{A}' \circ f = F \circ \mathcal{A}$.

Here, we define a function from a poset to another poset to be continuous if it preserves all the existing limits of directed sets. The f part determines F because \mathcal{A} is surjective. Since identity on \mathcal{I} is not a finite function, the identity on $\mathcal{O} = (\mathcal{E}, \mathcal{A}, \mathcal{I})$ is not a morphism. Therefore, M-domains, with M-domain morphisms, do not form a category. When ϕ is a morphism between M-domains, we write $\phi^{\mathcal{T}}$ and $\phi^{\mathcal{D}}$ for the first and the second component of ϕ , respectively.

Proposition 7.6 When $\mathcal{O} = (\mathcal{E}, \mathcal{A}, \mathcal{T})$ and $\mathcal{O}' = (\mathcal{E}', \mathcal{A}, \mathcal{T})$ are M-domains, there is a one to one correspondence between morphisms from \mathcal{O} to \mathcal{O}' and generic functions between the corresponding functors which lax commutes with coercions.

Proof. As we have noted, we can form a lax generic function from an M-domain morphism. For the converse, let m be such a lax generic function and d be the l.u.b. of an ω -chain $d_1 \preceq d_2 \preceq \dots$ in \mathcal{E} . Then, from the laxness of m , $m^\#(d_1) \preceq m^\#(d_2) \preceq \dots$ becomes an ω -chain in \mathcal{E} . On the other hand, since $\mathcal{A}(d_1) \sqsubseteq \mathcal{A}(d_2) \sqsubseteq \dots$ is an ω -chain in \mathcal{T} with the l.u.b. $\mathcal{A}(d)$, we have $\mathcal{A}(d_n) = \mathcal{A}(d_{n+1}) = \dots = \mathcal{A}(d) = t$ for some n since all the elements of \mathcal{T} are finite. Therefore, $d_n \preceq d_{n+1} \preceq \dots$ becomes an ω -chain in $\mathcal{A}^{-1}(t)$, and since each component of m is continuous, $m^\#(d)$ is the l.u.b. of $m^\#(d_n) \preceq m^\#(d_{n+1}) \preceq \dots$

Theorem 7.7 Let $\mathcal{O} = (\mathcal{E}, \mathcal{A}, \mathcal{I})$ and $\mathcal{O}' = (\mathcal{E}', \mathcal{A}', \mathcal{I}')$ be M-domains. Let $[\mathcal{E} \rightarrow \mathcal{E}']_{\mathcal{O}, \mathcal{O}'}$ be the set of M-domain morphisms from \mathcal{O} to \mathcal{O}' and let $\mathcal{A}'' : [\mathcal{E} \rightarrow \mathcal{E}']_{\mathcal{O}, \mathcal{O}'} \rightarrow [\mathcal{I} \xrightarrow{Fin} \mathcal{I}']$ be the first projection. Then, $[\mathcal{O} \rightarrow \mathcal{O}'] = ([\mathcal{E} \rightarrow \mathcal{E}']_{\mathcal{O}, \mathcal{O}'}, \mathcal{A}'', [\mathcal{I} \xrightarrow{Fin} \mathcal{I}'])$ is also an M-domain.

Let $\mathcal{O}_{\mathbf{B}} = (\mathcal{D}_{\mathbf{B}}, \mathcal{A}_{\mathbf{B}}, \mathcal{T}_{\mathbf{B}})$ be the M-domain composed of $\mathcal{D}_{\mathbf{B}} = \bigcup_{t \in \mathcal{T}_{\mathbf{B}}} D_{\mathbf{B}}(t)$. Let $\mathcal{O} = (\mathcal{E}, \mathcal{A}, \mathcal{I})$ and $\mathcal{O}' = (\mathcal{E}', \mathcal{A}', \mathcal{I}')$ be M-domains. We define the sum of \mathcal{O} and \mathcal{O}' as $(\mathcal{E} + \mathcal{E}', \mathcal{A} + \mathcal{A}', \mathcal{I} + \mathcal{I}')$. Here, $+$ is the smashed sum of posets, and define an isomorphism Φ from \mathcal{O} to \mathcal{O}' be a pair of isomorphisms $\Phi^{\mathcal{D}}$ from \mathcal{E} to \mathcal{E}' and $\Phi^{\mathcal{T}}$ from \mathcal{I} to \mathcal{I}' , such that $\mathcal{A}' \circ \Phi^{\mathcal{D}} = \Phi^{\mathcal{T}} \circ \mathcal{A}$.

Theorem 7.8 *If a functor $D \in \mathbf{Fun}(\mathcal{T}, \mathcal{C})$ satisfies Equation (3), then the op-fibration \mathcal{O} constructed by Proposition 7.2 is a M-domain which is isomorphic to $\mathcal{O}_{\mathbf{B}} + [\mathcal{O} \rightarrow \mathcal{O}]$.*

$$(5) \quad \mathcal{O} \begin{array}{c} \xrightarrow{\Phi} \\ \xleftarrow{\Psi} \end{array} \mathcal{O}_{\mathbf{B}} + [\mathcal{O} \rightarrow \mathcal{O}].$$

Conversely, if an M-domain \mathcal{O} is isomorphic to $\mathcal{O}_{\mathbf{B}} + [\mathcal{O} \rightarrow \mathcal{O}]$, the corresponding functor satisfies (3).

When $\mathcal{O} = (\mathcal{D}, \mathcal{A}, \mathcal{T})$ is a solution of equation (5), it can be decomposed as follows:

$$(6) \quad \begin{array}{ccc} \mathcal{D} & \begin{array}{c} \xrightarrow{\Phi^{\mathcal{D}}} \\ \xleftarrow{\Psi^{\mathcal{D}}} \end{array} & \mathcal{D}_{\mathbf{B}} + [\mathcal{D} \rightarrow \mathcal{D}]_{\mathcal{O}, \mathcal{O}'} \\ \downarrow \mathcal{A} & & \downarrow \mathcal{A}_{\mathbf{B}} + \mathcal{A}'' \\ \mathcal{T} & \begin{array}{c} \xrightarrow{\Phi^{\mathcal{T}}} \\ \xleftarrow{\Psi^{\mathcal{T}}} \end{array} & \mathcal{T}_{\mathbf{B}} + [\mathcal{T} \xrightarrow{\text{Fin}} \mathcal{T}]. \end{array}$$

Note that the bottom line of (6) is (1).

8 Solving the Equation

We outline how to solve equation (5). For this purpose, we extend the definition of an M-domain so that the type part also has limit elements.

Definition 8.1 *An M-domain $\mathcal{O} = (\mathcal{E}, \mathcal{A}, \mathcal{I})$ is continuous if it satisfies the followings:*

- 1) \mathcal{E} and \mathcal{I} are cpo.
- 2) when $t_0 \sqsubseteq t_1 \sqsubseteq \dots$ is an ω -sequence in \mathcal{I} with the l.u.b. t and $\mathcal{A}(d) = t$, then $d = \sqcup d|_{t_i}$.

The continuity of \mathcal{A} follows easily from this definition.

Definition 8.2 *Let $\mathcal{O} = (\mathcal{E}, \mathcal{A}, \mathcal{I})$ and $\mathcal{O}' = (\mathcal{E}', \mathcal{A}', \mathcal{I}')$ be continuous M-domains. A continuous morphism from \mathcal{O} to \mathcal{O}' is a pair (F, f) where $f : \mathcal{E} \rightarrow \mathcal{E}'$ and $F : \mathcal{I} \rightarrow \mathcal{I}'$ are continuous functions such that $\mathcal{A}' \circ f = F \circ \mathcal{A}$. We write $[\mathcal{E} \Rightarrow \mathcal{E}']_{\mathcal{O}, \mathcal{O}'}$ for the set of continuous morphisms.*

One can show that the set of continuous morphisms form a continuous M-domain $[\mathcal{O} \Rightarrow \mathcal{O}'] = ([\mathcal{E} \Rightarrow \mathcal{E}']_{\mathcal{O}, \mathcal{O}'}, \mathcal{A}'', [\mathcal{I} \rightarrow \mathcal{I}'])$. Thus, continuous M-domains, with continuous morphisms, form an \mathcal{O} -category. That is, every hom-set $[\mathcal{E} \Rightarrow \mathcal{E}']_{\mathcal{O}, \mathcal{O}'}$ becomes a ω -cpo with the limit preserved by morphism compositions. Other conditions of [SP82] are also satisfied. For example, the following M-domain \mathcal{O}_0 becomes the terminal object.

Definition 8.3 Let $\mathcal{O}_0 = (\mathcal{D}_0, \mathcal{A}_0, \mathcal{T}_0)$ be the M-domain with \mathcal{D}_0 and \mathcal{T}_0 be the one-point poset.

The limit of an ω^{op} chain $\mathcal{O}_1 \rightarrow \mathcal{O}_2 \rightarrow \dots$ is computed by taking the limits of the ω^{op} chains of cpo's for both components. The functor $id_{\mathcal{O}_{\mathbf{B}}} + [- \Rightarrow _]$ is shown to be locally continuous. Therefore, we can form the invariant of this functor.

Theorem 8.4 There is a continuous M-domain $\hat{\mathcal{O}} = (\hat{\mathcal{D}}, \hat{\mathcal{A}}, \hat{\mathcal{T}})$ which satisfies the following isomorphism:

$$\hat{\mathcal{O}} \begin{array}{c} \xrightarrow{\Phi} \\ \xleftarrow{\Psi} \end{array} \mathcal{O}_{\mathbf{B}} + [\hat{\mathcal{O}} \Rightarrow \hat{\mathcal{O}}].$$

Moreover, it is minimal in that the l.u.b. of the morphisms $p_n : \hat{\mathcal{O}} \rightarrow \hat{\mathcal{O}}$ defined by $p_0 = \perp_{\hat{\mathcal{O}}, \hat{\mathcal{O}}}$, $p_{n+1} = \Psi \circ id_{\mathcal{O}_{\mathbf{B}}} + [p_n \Rightarrow p_n] \circ \Phi$ is the identity on $\hat{\mathcal{O}}$.

Note that $\hat{\mathcal{T}}$ satisfies

$$\hat{\mathcal{T}} \begin{array}{c} \xrightarrow{\Phi^{\mathcal{T}}} \\ \xleftarrow{\Psi^{\mathcal{T}}} \end{array} \mathcal{T}_{\mathbf{B}} + [\hat{\mathcal{T}} \rightarrow \hat{\mathcal{T}}]$$

and it is the minimal invariant of the functor $\mathcal{T}_{\mathbf{B}} + [- \rightarrow _]$. From this and that $\mathcal{T}_{\mathbf{B}}$ is a finite poset, \mathcal{T} becomes the set of finite elements of $\hat{\mathcal{T}}$.

It also follows that $[\hat{\mathcal{T}} \xrightarrow{\text{Fin}} \hat{\mathcal{T}}]$ is isomorphic to $[\mathcal{T} \xrightarrow{\text{Fin}} \mathcal{T}]$.

We define $\mathcal{O} = (\mathcal{D}, \mathcal{A}, \mathcal{T})$ to be the restriction of $\hat{\mathcal{O}}$ to \mathcal{T} . That is, we take $\mathcal{D} = \bigcup_{t \in \mathcal{T}} \hat{\mathcal{A}}^{-1}(t)$.

Since $[\hat{\mathcal{O}} \Rightarrow \hat{\mathcal{O}}]$ restricted to $[\hat{\mathcal{T}} \xrightarrow{\text{Fin}} \hat{\mathcal{T}}]$ is $[\hat{\mathcal{O}} \rightarrow \hat{\mathcal{O}}]$, we have

$$\mathcal{O} \begin{array}{c} \xrightarrow{\Phi} \\ \xleftarrow{\Psi} \end{array} \mathcal{O}_{\mathbf{B}} + [\hat{\mathcal{O}} \rightarrow \hat{\mathcal{O}}].$$

Lemma 8.5 The behavior of a generic function is determined by its effects on finite elements.

Lemma 8.6 All the finite elements of $\hat{\mathcal{D}}$ belong to \mathcal{D} .

Therefore, $[\hat{\mathcal{O}} \rightarrow \hat{\mathcal{O}}]$ is isomorphic to $[\mathcal{O} \rightarrow \hat{\mathcal{O}}]$. And then to $[\mathcal{O} \rightarrow \mathcal{O}]$ because the result of a finite function is a finite element. Thus, we have proved the following:

Theorem 8.7 The M-domain \mathcal{O} constructed above satisfies (5).

9 Denotational Semantics of λ_m

Let $\mathcal{O} = (\mathcal{D}, \mathcal{A}, \mathcal{T})$ be an M-domain satisfying (5). We give a denotational semantics of λ_m on \mathcal{O} . The only difficult part is the treatment of the merge operator. We interpret the merge operator as the least upper bound operator in \mathcal{D} .

Proposition 9.1 For $e, f \in \mathcal{D}$, if $\mathcal{A}(e)$ and $\mathcal{A}(f)$ are compatible in \mathcal{T} , then the least upper bound $e \sqcup f$ of e and f exists in \mathcal{D} and $\mathcal{A}(e \sqcup f) = \mathcal{A}(e) \boxtimes \mathcal{A}(f)$.

Proof. It is proved by induction on the degree on \mathcal{T} as follows: When $\mathcal{A}(e)$ and $\mathcal{A}(f)$ are basic types, then the existence of a least upper bound is ensured by its co-flat structure. When $\mathcal{A}(e)$ and $\mathcal{A}(f)$ are function types, we have $e \in \Pi_{s \in \mathcal{T}} E'(\Psi^{\mathcal{T}}(\mathcal{A}(e))(s))^{E(s)}$ and $f \in \Pi_{s \in \mathcal{T}} E'(\Psi^{\mathcal{T}}(\mathcal{A}(f))(s))^{E(s)}$. Since $\Psi^{\mathcal{T}}(\mathcal{A}(e))(s)$ and $\Psi^{\mathcal{T}}(\mathcal{A}(f))(s)$ have smaller degree than those of $\mathcal{A}(e)$ and $\mathcal{A}(f)$ for all $s \in \mathcal{T}$, we have the least upper bound $e_s \sqcup f_s$ of the s -components. Thus, we can form $e \sqcup f$ as $\langle s \in \mathcal{T}. e_s \sqcup f_s \rangle$. \square

Though D is constructed as the minimal invariant of an equation, we can prove, as in Prop. 9.1, a lot of properties of D using the inductive structure of \mathcal{T} .

Proposition 9.2 \mathcal{D} is consistently complete.

Proof. From Prop. 9.1 and the consistently completeness of \mathcal{T} .

An environment ρ is an assignment of an element of $\mathcal{A}^{-1}(\overline{V})$ to each free variable x^V . When $M : V$ is a term of λ_m , we define $\mathcal{E}[[M]](\rho) \in \mathcal{D}$ so that $\mathcal{A}(\mathcal{E}[[M]](\rho)) = \overline{V}$ as follows:

$$\begin{aligned} \mathcal{E}[[x^V]](\rho) &= \rho(x^V) \\ \mathcal{E}[[K^B]](\rho) &= \Psi^{\mathcal{D}}(K^B \in \mathcal{D}_{\mathbf{B}}) \\ \mathcal{E}[[\lambda x^V. M]](\rho) &= \Psi^{\mathcal{D}}(\text{lambda}(d). \text{if}(\mathcal{A}(d) \supseteq \overline{V}) \text{ then } \mathcal{E}[[M]](\rho[d|_{\overline{V}}/x^V]) \\ &\quad \text{else } \perp_{\mathcal{D}}) \\ \mathcal{E}[[M N]](\rho) &= \Phi^{\mathcal{D}}(\mathcal{E}[[M]](\rho)) \mathcal{E}[[N]](\rho) \\ \mathcal{E}[[M|_V]](\rho) &= \mathcal{E}[[M]](\rho)|_{\overline{V}} \\ \mathcal{E}[[M_1 \otimes M_2]](\rho) &= \mathcal{E}[[M_1]](\rho) \sqcup \mathcal{E}[[M_2]](\rho) \end{aligned}$$

Lemma 9.3 When f is a continuous function from $\mathcal{A}^{-1}(\overline{V})$ to $\mathcal{A}^{-1}(\overline{V}')$, then the following function from \mathcal{D} to \mathcal{D} belongs to $[\mathcal{D} \rightarrow \mathcal{D}]_{\mathcal{O}, \mathcal{O}}$,

$$\text{lambda}(d). \text{if}(\mathcal{A}(d) \supseteq \overline{V}) \text{ then } f(d|_{\overline{V}}) \text{ else } \perp_{\mathcal{D}}.$$

Theorem 9.4 (Soundness) if $M \triangleright N$, then $\mathcal{E}[[M]](\rho) = \mathcal{E}[[N]](\rho)$.

Proof. By checking each reduction rule. \square

10 Computational Adequacy

Usually, computational adequacy property means that if a closed term M of basic type B does not have a normal form, then $\mathcal{E}[[M]]$ is \perp^B . The computational adequacy property of λ_m has a different form because the semantic domain $D(\overline{B})$ of B does not have a least element.

Theorem 10.1 (Computational Adequacy) *If a closed term M of basic type B has the denotation $\mathcal{E}[[M]] = \Psi^{\mathcal{D}}(K^B)$, then M is reduced to a form $K^B[\otimes M']$.*

Here, $K^B[\otimes M']$ means K^B or $K^B \otimes M'$ for some M' . Since the poset $D(\overline{B})$ is composed of only constants of type B , every term M of type B has the denotation $\Psi^{\mathcal{D}}(K^B)$ for some K^B . Therefore, Theorem 5.1 is easily derived as a corollary to this theorem.

Theorem 10.1 is proved by constructing the formal approximation relation. Recently, [Pit94,Pit96] has developed a technique for proving the adequacy of a language with respect to the model constructed over the minimal invariant of a recursive domain equation, by defining a formal approximation relation as a fixed point of a constructor of mixed variance over relations. This construction is applicable to our case. The proof of the existence of a formal approximation relation relies on the minimality, that is, representability of the identity on \mathcal{D} as a least upper bound of its projections. Therefore, we consider a relation between $\hat{\mathcal{D}}$ and $Prog$ instead of a relation between \mathcal{D} and $Prog$.

We write $Prog(V)$ for the set of closed expressions of type V , and $Prog$ for the set of all closed expressions. We use meta variables P and Q for closed expressions.

Definition 10.2 *We define \mathcal{R} as the set of all binary relations $\{R \subset \hat{\mathcal{D}} \times Prog\}$ satisfying the followings:*

- (i) $\perp_{\mathcal{D}} R P$ for all $P \in Prog$,
- (ii) when $d_0 \preceq d_1 \preceq \dots \preceq d_n \preceq \dots$ is an increasing ω -sequence in \mathcal{D} for which $d_n R P$ ($n = 0, 1, \dots$) holds, then $(\sqcup_n d_n) R P$.

Definition 10.3 $\lesssim \in \mathcal{R}$ is a formal approximation if the followings hold:
 $d \lesssim P$ iff $d = \perp_{\mathcal{D}}$ or $(P \triangleright^+ K^B[\otimes M']$ and $\mathcal{E}[[K^B]] = d)$ or $(P \triangleright^+ \mathbf{Top}^B$ and $\mathcal{A}(d) = \overline{B}$ for a basic type $B)$ or $(\mathcal{A}(d) \in \mathcal{T}_{\mathbf{F}}$ and $\Phi^{\mathcal{D}}(d)(d') \lesssim P P'$ for all (d', P') such that $d' \lesssim P'$ and P is applicable to $P')$.

We omit the details, but we can show the existence of a formal approximation relation by applying essentially the same construction as in [Pit94].

Theorem 10.4 *There exists a formal approximation relation \lesssim .*

We can show, by induction on the degree of types, the following lemmas.

Lemma 10.5 *When $P \triangleright^+ Q$, we have $d \lesssim P$ iff $d \lesssim Q$.*

Lemma 10.6 *Suppose that $d \lesssim P$ and $d' \lesssim P'$ with the types of P and P' compatible. Then, $d \sqcup d'$ exists and $d \sqcup d' \lesssim P \otimes P'$.*

Lemma 10.7 *Suppose that $d \lesssim P$ and $\overline{V} \sqsubseteq \mathcal{A}(d)$. Then $d|_{\overline{V}} \lesssim P|_{\overline{V}}$.*

By induction on the formation of terms, we can prove the following.

Proposition 10.8 *Suppose that \lesssim is a formal approximation and $M : V$ is a term with free variables $x_1^{V_1}, \dots, x_n^{V_n}$ such that $\rho(x_i^{V_i}) \lesssim P_i$ and $P_i : V_i$ for $i = 1, \dots, n$. Then $\mathcal{E}[[M]](\rho) \lesssim M[x_1^{V_1} := P_1, \dots, x_n^{V_n} := P_n]$.*

From this proposition, when M is a closed expression of a basic type B , we have $\mathcal{E}[[M]] \lesssim M$. Since $\mathcal{E}[[M]] \neq \perp_{\mathcal{D}}$, it means that $(\mathcal{E}[[M]] = \Psi^{\mathcal{D}}(K^B))$ and $M \triangleright^+ K^B[\otimes M']$ or $M \triangleright^+ \mathbf{Top}^B$. From the soundness property, we have $\mathcal{E}[[M]] = \Psi^{\mathcal{D}}(\top^B)$ when $M \triangleright^+ \mathbf{Top}^B$. Thus, we have proved the adequacy property.

From this adequacy property, we can prove some equivalences between terms.

Definition 10.9 *When M and N are terms of equivalent types, M operationally approximates N ($M \preceq N$) iff $C[M] \triangleright^+ K[\otimes M']$ implies $C[N] \triangleright^+ K[\otimes N']$ or $C[N] \triangleright^+ \mathbf{Top}^B$ for any closed context $C[\]$ of a basic type B . When $M \preceq N$ and $N \preceq M$, we write $M \approx N$ and say that M and N are (operationally) equivalent.*

Proposition 10.10 1) *When $U \simeq V$ and $M : U$, we have $M \approx M|_V$. In particular, $M \approx M|_U$ when $M : U$.*

2) *When $W \leq V \leq U$ and $M : W$, we have $M|_V|_U \approx M|_U$.*

This justifies our functorial semantics of λ_m . Finally, we have the laxness of our generic functions as a corollary to the adequacy property.

Proposition 10.11 *A generic function lax commutes with coercions. That is, when $V \leq U$, $N : U$, and $M : F$ is applicable to V , then $M N|_U \preceq M N$.*

11 Conclusion

We have studied the syntactic and semantic properties of a calculus λ_m , which is a polymorphic calculus defined over a hierarchical type structure. In λ_m , though a generic function can be composed, like an overloaded function, from more than one lambda expression, the ways it behaves on each type are related in that it lax commutes with coercion functions. This laxness condition can also be stated, when considered as one function defined over the opfibration composed of all the values of all the types, as preserving the information order between values.

This calculus has the syntactic property that, though it is not normalizing, it does not have an unsolvable term. Therefore, the recursive equations expressing the circular structure causing non-normalization need to be solved in a category with non-pointed domains. This is realized by considering an opfibration composed of all the values of all the types, and expressing the equations as one equation between opfibrations. A kind of adequacy property is also proved applying Pitts' technique, and some syntactic properties are derived using this.

The author has studied, in previous works, two related calculi: a calculus

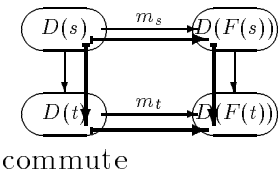
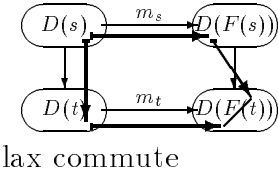
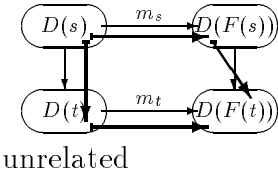
	Generic Functions	Relation with Coercion Functions	Syntactic Properties	Domain Construction
λ_{\otimes}^p	Coherent	 commute	Normalizing	Finite product
λ_m	Lax	 lax commute	Non-normalizing, but every term is reduced to K or $K \otimes M$	Domain equation over op-fibrations
λ_{\otimes}	Ad-hoc	 unrelated	Non-normalizing and unsolvable terms exist	Domain equation over functors

Fig. 2. Properties of λ_{\otimes}^p , λ_m , and λ_{\otimes} .

of ad-hoc generic functions λ_{\otimes} and a calculus of coherent generic functions λ_{\otimes}^p . The three calculi, λ_{\otimes} , λ_{\otimes}^p , and λ_m , have the same hierarchical type structure, but have different syntactic and semantic properties according to the slightly different typing rules and reduction rules. Properties of these three calculi are listed in Figure 2.

As is shown in this figure, when we require more relations among the branches of a generic function, better syntactic and semantic properties are derived. Though one may find the construction of the semantic domain for λ_m be more complicated than that for λ_{\otimes} , it is mainly due to the lack of limit elements in the type structure.

It is usual that a calculus is designed to give a foundation for a new programming language mechanism. Though we have drawn perspective for error handling in Section 2, this calculus has some difficulty if viewed as a fundamental calculus of a programming language in that it is not efficient to execute all the applicable branches of a generic function. The author considers that the importance of this calculus exists not as a fundamental calculus of a programming language, but rather for the model it presents. We will explain this point as the final remark.

Though the calculus λ_{\otimes}^p has good properties, the type system restricts the way a generic function is defined so that no overriding is expressible in this

language. On the other hand, λ_{\otimes} is more close to a programming language design in that one can express both inheritance and overriding in writing a subtype method. However, unrestricted use of overriding allows one to write a program very difficult to understand. The complicated semantic structure of λ_{\otimes} can be considered as a reflection of such complication.

The author considers that though overriding is essential in object oriented programming, a programmer is expected to ensure that a generic function written using overriding behaves coherently to every type. When we express it in our calculi, though the expressive power of λ_{\otimes} is required, a programmer is expected to realize the semantic structure of λ_{\otimes}^p . However, the two languages have different syntax and thus a λ_{\otimes} program cannot be interpreted in λ_{\otimes}^p . We consider that λ_m fills this gap between the syntax of λ_{\otimes} and the semantics of λ_{\otimes}^p . Since λ_{\otimes} and λ_m has the same syntax, we can consider the λ_m -semantics to a λ_{\otimes} program. Then, a generic function with non-coherent behavior produces an erroneous value and thus lax commutes with coercions, whereas a generic function with coherent behavior is interpreted as a natural transformation.

Though this paper is written mainly with theoretical interest on domain theory over (op)fibred structure, the author thinks coherency is an important property in real object oriented programming, and he expects that this theoretical study helps understanding the subject.

Acknowledgement

The author thanks Masami Hagiya, Andreas Knobel, and Masahito Hasegawa for fruitful discussions and invaluable comments.

References

- [BMM88] K. B. Bruce, A. R. Meyer, and J. C. Mitchell. The semantics of second-order lambda calculus. *Information and Computation*, 59:76–134, 1988.
- [BW90] M. Barr and C. Wells. *Category Theory for Computing Science*. Prentice Hall, 1990.
- [CGL95] Giuseppe Castagna, Giorgio Ghelli, and Giuseppe Longo. A calculus for overloaded functions with subtyping. *Information and Computation*, 117(1):115–135, 1995.
- [Fio94] Marcelo P. Fiore. *Axiomatic Domain Theory in Categories of Partial Maps*. PhD thesis, University of Edinburgh, 1994.
- [Fre91] P. J. Freyd. Algebraically complete categories. In *Proc. 1990 Como Category Theory Conference, Lec. Notes in Math. 1488*, pages 95–104. Springer-Verlag, 1991.
- [Gir71] J. Y. Girard. Une extension de l’interprétation de gödel á l’analyse et son application á l’élimination des coupures dans l’analyse et la théorie des

- types. In *Proc. 2nd Scandinavian Logic Symposium*, pages 63–92. North Holland, 1971.
- [GJGL96] James Gosling, Bill Joy, and Jr. Steele Guy L. *The Java Language Specification*. Addison-Wesley, 1996.
- [Has94] R. Hasegawa. Categorical data types in parametric polymorphism. *Math.Struct. in Comp. Science*, 4:71–109, 1994.
- [Jac99] B. Jacobs. *Categorical Logic and Type Theory*. North Holland, Elsevier, 1999.
- [Pho92] W. Phoa. An introduction to fibrations, topos theory, the effective topos and modest sets. Technical Report ECS-LFCS-92-208, Edinburgh University, 1992.
- [Pit94] A. M. Pitts. Computational adequacy via ‘mixed’ inductive definitions. In *Mathematical Foundations of Programming Semantics, Proc. 9th Int. Conf., New Orleans, LA, USA, April 1993*, volume 802 of *Lecture Notes in Computer Science*, pages 72–82. Springer-Verlag, Berlin, 1994.
- [Pit96] A. M. Pitts. Relational properties of domains. *Information and Computation*, 127:66–90, 1996.
- [Rey74] J. C. Reynolds. Towards a theory of type structure. In *Proceedings, Colloque sur la Programmation, LNCS Vol19*, pages 408–425, 1974.
- [Rey81] J. C. Reynolds. Using category theory to design implicit conversions and generic operators. In *Lecture Notes in Computer Science, 94*. Springer-Verlag, 1981.
- [Rey83] J. C. Reynolds. Types, abstraction, and parametric polymorphism. In R. E. A. Mason, editor, *Information Processing 83*, pages 513–523. North-Holland, 1983.
- [SP82] M. Smyth and G. Plotkin. The category-theoretic solution of recursive domain equations. *SIAM J. Comput.*, 11(4):761–783, 1982.
- [Tak95] M. Takahashi. Parallel reduction in lambda-calculus. *Information and Computation*, 118:120–127, 1995.
- [Tsu94] Hideki Tsuiki. On typed calculi with a merge operator. In *14th Conference on Foundations of Software Technology and Theoretical Computer Science, LNCS 880*, pages 101–112, 1994.
- [Tsu95] Hideki Tsuiki. A categorical model of overloading via a domain theory over a functor category. An earlier version available as a technical report KSU/ICS-95-01 of Institute of Computer Science, Kyoto Sangyo University, with the title “A Denotational Model of Overloading”, 1995.
- [Tsu98] Hideki Tsuiki. A computationally adequate model for overloading via domain-valued functors. *Math.Struct. in Comp. Science*, 8:321–349, 1998.

A Types of λ_m

Suppose that a finite set of basic types like **Int** and **Bool** (ranged over by B) is given. We first define pretypes (ranged over by U and V) and function pre-components (ranged over by H) as follows:

$$\begin{aligned} V &::= B \mid [\overbrace{H, \dots, H}^{\text{at least one}}], \\ H &::= V \rightarrow V . \end{aligned}$$

We call pretypes of the form $[H_1, \dots, H_n]$ function (pre)types, which is ranged over by F and G . Note that not all the lists of function components are allowed as generic function types. We define the following compatibility relation \uparrow on pretypes.

$$\begin{aligned} \text{(m-REFL)} \quad & \frac{}{B \uparrow B} & \text{(m-FUN)} \quad & \frac{(\text{not } V_1 \uparrow V_2) \text{ or } (V_1 \uparrow V_2 \text{ and } V'_1 \uparrow V'_2)}{V_1 \rightarrow V'_1 \uparrow V_2 \rightarrow V'_2} \\ & & \text{(m-FLIST)} \quad & \frac{H_i \uparrow H'_j \quad (i = 1, \dots, n, j = 1, \dots, m)}{[H_1, \dots, H_n] \uparrow [H'_1, \dots, H'_m]} \end{aligned}$$

Note that \uparrow is well defined though a negation of \uparrow appears in the precondition of (m-FUN) (See [Tsu94]). We impose the condition on $[H_1, \dots, H_n]$ that H_1, \dots, H_n are pairwise compatible (i.e. $H_i \uparrow H_j$ for $i, j = 1, \dots, n$), and define type expressions.

When $V_1 \uparrow V_2$, we define $V_1 \bowtie V_2$ as an abbreviation for a type as follows:

$$\begin{aligned} B \bowtie B &= B, \\ [H_1, \dots, H_m] \bowtie [H_{m+1}, \dots, H_n] &= [H_1, \dots, H_n]. \end{aligned}$$

The subtype relation is defined as follows:

$$\begin{aligned} \text{(I-REFL)} \quad & \frac{}{V \leq V} & \text{(I-FLIST1)} \quad & \frac{}{[H_1, \dots, H_n] \leq [H_i]} \\ \text{(I-TRAN)} \quad & \frac{V \leq V' \quad V' \leq V''}{V \leq V''} & \text{(I-FLIST2)} \quad & \frac{V_1 \uparrow V_2}{[V_1 \rightarrow V'_1, V_2 \rightarrow V'_2] \leq [V_1 \bowtie V_2 \rightarrow V'_1 \bowtie V'_2]} \\ \text{(I-FUN)} \quad & \frac{V \leq U \quad U' \leq V'}{[U \rightarrow U'] \leq [V \rightarrow V']} & \text{(I-FLIST3)} \quad & \frac{[H_1, \dots, H_n] \leq [H'_i] \quad (i = 1, \dots, m)}{[H_1, \dots, H_n] \leq [H'_1, \dots, H'_m]} \end{aligned}$$

Note that we do not consider a subtype relation between basic types, and therefore a basic type is only comparable with itself. We call V a subtype of

U when $V \leq U$, and we call V and U are equivalent and write $V \simeq U$ when $V \leq U$ and $U \leq V$.