プログラミング-言語・基礎・実践 14－6
（1993．10．29）

# 多重定義と部分型が存在する計算の強正規性と型保存性について

立木秀樹
慶應義塾大学環境情報学部

最近、単純型付きλ計算に多重定義と部分型を付け加えた計算λ&-calculus が Castagna, Ghelli, Longo により提案された。λ&-calculus では、多重定義と部分型の相互作用のため、強正規性がなりたたない、計算のモデルが考えにくいなどの好ましくない性質を持っている。本論文では、多重定義と部分型の相互作用について研究する。型の非保存性がこの計算の型理論的研究を困難にしている。そこで、型変換を導入することにより、型の保存性の成り立つように λ&-calculus を変更した二つの計算λ&$C$と λ&$C^*$を考える。これらは、多重定義された関数に対する 2 つの異なる意味の与え方に対応している。両者は、正規性、型変換の推移性などに異なる性質を持ち、異なるモデルを持つ。

# A normalizing calculus with overloading and subtyping.

Hideki Tsuiki
Faculty of Environmental Information
Keio University
tsuiki@sfc.keio.ac.jp

Recently, λ&-calculus, an extension of the simply typed lambda calculus with overloading and subtyping was presented by Castagna, Ghelli, and Longo. In λ&-calculus, the interaction of overloading and subtyping causes some unexpected properties such as non-normalization and difficulty in constructing a natural model. One reason for this is that the type of a term is not preserved but is reduced by reduction. In this paper, we present modifications of this calculus so that they have the Subject Reduction property, and investigate syntactic and semantic properties in detail. We define two calculi λ&$C$ and λ&$C^*$ corresponding to two possible meanings of an overloaded type. They have completely different properties with respect to normalization property and transitivity of coerions property, and have different semantics.

# 1 Introduction.

Recently, an extension of the simply typed lambda calculus ($\lambda^{\to}$) with overloading and subtyping was presented by Castagna, Ghelli, and Longo([CGL94]). In this calculus, $\lambda\&$-calculus (or $\lambda\&$ in short), an overloaded function is defined by putting different branches of code together, and when it is applied, the branch to execute is chosen according to the type of the argument. Very roughly, when $e_1 : V_1 \to V_1', \ldots, e_n : V_n \to V_n'$ are function terms, $e_1\& \ldots \& e_n$ is an overloaded function with an overloaded type $\{V_1 \to V_1', \ldots, V_n \to V_n'\}$, and an overloaded application $(e_1\& \ldots \& e_n) \bullet f$ is reduced to a function application $e_i \cdot f$ where $e_i$ is selected based on the type of $f$. This calculus provides a foundation for typed object-oriented languages by a novel way: it models a message as an overloaded function defined by combining all the methods with the same name, and message sending as an application of the overloaded function. This is actually the way CLOS implements object orientedness and $\lambda\&$ is an attempt to add type structures to the core of CLOS. Though some aspects of object orientedness like encapsulation cannot be modeled, this new calculus is promising in that it explains the computational mechanism of object oriented languages from a new viewpoint.

On the other hand, $\lambda\&$ behaves differently from many typed calculi in that it is not Strong Normalizing [1]. In many calculi, adding subtyping does not affect the normalizing property of a calculus. For example, $\lambda_{\leq}$, the extension of $\lambda^{\to}$ with subtyping, and $F_{\leq}$, the extension of system $F$ with subtyping and bounded quantification, are Strong Normalizing. In the case of $\lambda\&$, $\lambda\&^{-\leq}$: the extension of $\lambda^{\to}$ only with overloading is Strong Normalizing, but $\lambda\&$: its extension with subtyping is not. This is related to a property of the translation into a calculus without subtyping defined by inserting coercion functions. In the case of $\lambda_{\leq}$ and $F_{\leq}$, the translations into $\lambda^{\to}$ and $F$ respectively preserve reductions ([BTCGS91], [BTGS90]). In $\lambda\&$, though we can define a translation to $\lambda\&^{-\leq}$, reduction is not preserved by this translation. In this paper, we will study syntax and semantics of calculi with subtyping and overloading in detail.

Crucial features of $\lambda\&$ are that the type of a term changes to a subtype of the statically given type according to the reduction, and that the branch selection depends not on the compile-time type (the type of a term) but on the run-time type (the type of the normal form of a term) of the argument. The former feature is called generalized Subject Reduction in [CGL94], and the latter feature is called late binding in object oriented terminology. Late binding is one of the characteristics of object orientedness. However, in order to express this, $\lambda\&$ becomes rather complicated. For example, in order to select a branch according to the run-time type, there is an restriction that an overloaded application $e \bullet f$ can only be reduced when $f$ is a normal form. Moreover, late binding makes it difficult to consider a model because we need to consider two notions of types.

Thus, in our study of the relation between subtyping and overloading, we will consider modifications of $\lambda\&$ so that Subject Reduction holds. It is done by inserting coercions to the argument when a term is $\beta$-reduced. That is, we add $e|_V$ to terms and use the following rule instead of the $\beta$ rule.

$$(\beta_C) \quad (\lambda x^V.e) \cdot f \rhd e[x := f|_V]$$

Here, $f|_V$ means the term $f$ but is considered as a term of type $V$. Thus, the calculi we present have Subject Reduction property, and therefore method selection is based on the static type of the argument (early binding).

Even though these calculi are Subject Reduction and early binding, the interaction of subtyping and overloading makes them non-trivial and worth studying. We present two calculi $\lambda\&C$ and $\lambda\&C^*$, which only differ in the reduction rule of an application of a coerced function. Surprisingly, they behave completely differently as in listed in Figure 1. Strong Normalization holds in $\lambda\&C^*$ though it is not the case in $\lambda\&C$, and transitivity of coercions holds in $\lambda\&C$ though it is not the case in $\lambda\&C^*$. The translation of $\lambda\&C^*$ into $\lambda\&^{-\leq}$ preserves reductions, and the index of the branch selected in each overloaded application can be determined statically. However, it is not the case in $\lambda\&C$. We will also study the semantics of both calculi, which correspond to two possible meanings of an overloaded function under the existence of subtyping.

# 2 Review of the $\lambda\&$-calculus.

The formal system of $\lambda\&$is listed in Figure 2. We use metavariables $U$, $V$ for types, $A$, $B$ for ground types, $F$,$G$ for overloaded types, and $e$, $f$ for expressions. Note that not all the lists $\{V_1 \to V_1', \ldots, V_n \to V_n'\}$ are allowed as types: it is subject to the restrictions listed in Figure 2. $at(F, U)$ is the index of the branch selected when the argument has type $U$, and the second condition ensures that the index is uniquely determined. Since these conditions depend on the subtyping rules, we first defined pretypes and subtype relations over pretypes, and then defined types as pretypes which satisfy

[1] The current author has also found this fact independently after $\lambda\&$ was first presented in [CGL92].

| | Unicity of Type | Subject Reduction | Church Rosser | Transitivity of Coercions | Strong Normalization | Translation to $\lambda\&^{-\leq}$ |
|---|---|---|---|---|---|---|
| $\lambda\&C$ | ○ | ○ | ○ | ○ | × | × |
| $\lambda\&C^*$ | ○ | ○ | ○ | × | ○ | ○ |

Figure 1: Properties of Calculi.

the conditions. The subtype relation of overloaded types says that $F$ is a subtype of $G$ when every component of $G$ is a supertype of a component of $F$.

As for terms, $e \cdot e'$ is an application of a function $e$ to $e'$, and $e \bullet e'$ is an application of an overloaded function $e$ to $e'$. $e\&^F e'$ is an overloaded function composed by adding a function $e'$ to an overloaded function $e$, and $\epsilon$ is the empty overloaded function from which every overloaded function is composed. Since only one typing rule is applicable to each term, every well-typed term has a unique type. That is, Unicity of Type property holds.

As for reduction rules, $(\beta)$ is for a function application, and $(\beta_\&)$ is for an overloaded function application. Note that the type $V$ of the argument is used in $(\beta_\&)$. That is, type information is used in reduction, whereas types are erased and untyped terms are reduced in other typed calculi. The restriction that $f$ is a normal form is required because the type of $f$, on which the reduction depends, may decrease according to the reduction. Thus, we first reduce the argument to a normal form and use the type of the normal form in selecting a branch. Note that the type of $e_1\&^F e_2$ do not change though the types of $e_1$ and $e_2$ may change according to the reduction because we have a superscript $F$.

$\lambda\&$ has Church-Rosser and Unicity of Type properties. It is easy to see that $\lambda\&$ does not have the Subject Reduction property. For example, when $U \leq V$ and $e : U$, $(\lambda x^V.x) \cdot e$ has type $V$, and reduces by $(\beta)$ to $e$ of type $U$. Only the *generalized* Subject Reduction property holds: $e : U$ and $e \triangleright^* e'$ implies that the type $U'$ of $e'$ is a subtype of $U$.

We will show, in the rest of this section, that there is a non-normalizing term in $\lambda\&$. This example is essentially the same as the one given in [CGL94]. We assume that there exists a ground type $i$, say *int*, and a constant 1 of type $i$.

First, we consider a self application and find a type $S = \{V_1 \longrightarrow V_1', \ldots, V_n \longrightarrow V_n'\}$ for which $(\lambda x^S.x \bullet x)$ is well-typed. For this, at least one of the $V_i$, say $V_1$, is a supertype of $S$. That is, the inequality

$$\{V_1 \longrightarrow V_1', \ldots, V_n \longrightarrow V_n'\} \leq V_1$$

holds. It holds for $S = \{i \longrightarrow i, \{i \longrightarrow i\} \longrightarrow i\}$ because $\{i \longrightarrow i, \{i \longrightarrow i\} \longrightarrow i\} \leq \{i \longrightarrow i\}$. In the rest of this paper, we write $S$ for this type. Thus, $\lambda x^S.x \bullet x$ is a well typed term.

Next, we try to form a term

$$y = (e\&^T(\lambda x^S.x \bullet x)) \bullet (e\&^T(\lambda x^S.x \bullet x)),$$

for which the branch $(\lambda x^S.x \bullet x)$ is selected in $(\beta_\&)$ reduction over $y$. For this, $T$ must satisfy that $T \leq S$. One of the candidates for $T$ is to take $T = \{i \longrightarrow i, \{i \longrightarrow i\} \longrightarrow i, S \longrightarrow i\}$. Since $T$ includes all the components of $S$, it is easy to show that $T \leq S$. Note that $S \leq T$ is also true because $\{i \longrightarrow i\} \geq S$, and therefore $\{i \longrightarrow i\} \longrightarrow i \leq S \longrightarrow i$.

Let $e$ be a term of type $S$ like $\epsilon\&^{i\longrightarrow i}(\lambda x^i.x)\&^S(\lambda x^{i\longrightarrow i}.1)$, and $z$ be $(e\&^T(\lambda x^S.x \bullet x))$. Then $y = z \bullet z$ is a well typed term of type $i$, which reduces infinitely.

$$\begin{aligned} z \bullet z &= (e\&^T(\lambda x^S.x \bullet x)) \bullet z \\ &\triangleright (\lambda x^S.x \bullet x) \cdot z \\ &\triangleright (z \bullet z) \\ &\triangleright \ldots. \end{aligned}$$

# 3 $\lambda\&^{-\leq}$: A calculus without subtyping

We start our study with a calculus $\lambda\&^{-\leq}$ with overloading but without subtyping. The types of $\lambda\&^{-\leq}$ are the pretypes of $\lambda\&$ with the restriction that the domain types of an overloaded function are different. The terms of $\lambda\&^{-\leq}$ are those of $\lambda\&$ with $e\&^F e'$ replaced by $e\&e'$. Since there is no subtype relation, the typing rules become simple. [$\longrightarrow$ Elim], [{}Intro], and [{}Elim] are replaced by the following rules.

$$[\longrightarrow \text{Elim'}] \quad \frac{e : V \longrightarrow V' \quad f : V}{e \cdot f : V'}$$

$$[\{\}\text{Elim'}] \quad \frac{e : F = \{V_i \longrightarrow V_i'\}_{i\leq n} \quad f : V_j}{e \bullet f : V_j'}$$

$$[\{\}\text{Intro'}] \quad \frac{e : \{V_i \longrightarrow V_i'\}_{i\leq(n-1)} \quad f : V_n \longrightarrow V_n'}{e\&f : \{V_i \longrightarrow V_i'\}_{i\leq n}}$$

The $(\beta_\&^-)$ rule is also replaced by the following simplar one.

## λ&-calculus

**Pre-Types:**

$$V ::= A \mid V \longrightarrow V' \mid \{V_1 \longrightarrow V_1', \ldots, V_n \longrightarrow V_n'\}$$

**Subtype Relation:**

$$\frac{}{A \leq A} \qquad \frac{U \geq V \quad U' \leq V'}{U \longrightarrow U' \leq V \longrightarrow V'}$$

$$\frac{\forall j \leq m, \exists i \leq n \text{ s.t. } U_i \longrightarrow U_i' \leq V_j \longrightarrow V_j'}{\{U_1 \longrightarrow U_1', \ldots, U_n \longrightarrow U_n'\} \leq \{V_1 \longrightarrow V_1', \ldots, V_m \longrightarrow V_m'\}}$$

**Restriction on Types:**

A pretype $F = \{V_1 \longrightarrow V_1', \ldots, V_n \longrightarrow V_n'\}$ is a type only when

1. If $V_i \leq V_j$, then $V_i' \leq V_j' (i, j \leq n)$.
2. For all pretype $U$ such that $U \leq V_i$ for some $i \leq n$, there is a unique $h \leq n$ such that $V_h$ is the least element of $\{V_i \mid V_i \geq U\}$. We write $at(F, U)$ for this $h$.

**Terms:**

$$e ::= x^V \mid d(constants) \mid \lambda x^V.e \mid e \cdot e \mid \epsilon \mid e \&^F e \mid e \bullet e$$

**Typing rules:**

[Taut] $\dfrac{}{x^V : V}$ [Taut$_\epsilon$] $\dfrac{}{\epsilon : \{\}}$

[$\longrightarrow$ Intro] $\dfrac{e : V'}{\lambda x^V.e : V \longrightarrow V'}$ [{}Intro] $\dfrac{e : F_1 \leq \{V_i \longrightarrow V_i'\}_{i \leq (n-1)} \quad f : U \leq V_n \longrightarrow V_n'}{e \&^{\{V_i \longrightarrow V_i'\}_{i \leq n}} f : \{V_i \longrightarrow V_i'\}_{i \leq n}}$

[$\longrightarrow$ Elim] $\dfrac{e : V \longrightarrow V' \quad f : U \leq V}{e \cdot f : V'}$ [{}Elim] $\dfrac{e : F = \{V_i \longrightarrow V_i'\}_{i \leq n} \quad f : U}{e \bullet f : V'_{at(F,U)}}$

**Reduction Rules:**

($\beta$) $(\lambda x^V.e) \cdot f \triangleright e[x := f]$

($\beta_\&$) $(e_1 \&^F e_2) \bullet f \triangleright \begin{cases} e_1 \bullet f & (f : U, f \text{ is a normal form}, at(F, U) < n) \\ e_2 \cdot f & (f : U, f \text{ is a normal form}, at(F, U) = n) \end{cases}$

Figure 2:

$(\beta_{\&}^{-})$ $\quad (e_1 \& e_2) \bullet f \triangleright$
$e_1 \bullet f \ (e_1 \& e_2 : F = \{V_i \longrightarrow V_i'\}_{i \leq n},\ f : U,\ U \neq V_n)$
$e_2 \cdot f \ (e_1 \& e_2 : F = \{V_i \longrightarrow V_i'\}_{i \leq n},\ f : U,\ U = V_n)$

The calculus $\lambda\&^{-\leq}$ thus defined has good properties:

**Theorem 1** *$\lambda\&^{-\leq}$ has Unicity of Type, Subject Reduction, Church-Rosser, and Strong Normalization properties.*

Church Rosser can be proved by using the notion of parallel reduction due to Tait and Martin-Löf following the construction in [Tak89]. Strong Normalization can be proved by extending the Tait's method for the simply type lambda calculus ([GLT89]).

From this theorem, we can construct a term model of $\lambda\&^{-\leq}$ in which the meaning of a type is the set of closed normal forms of the type. It is also easy to encode $\lambda\&^{-\leq}$ into $\lambda^{\rightarrow}$ with product types by considering $\{V_1 \longrightarrow V_1', \ldots, V_n \longrightarrow V_n'\}$ as a $n$-product of $V_1 \longrightarrow V_1', \ldots, V_n \longrightarrow V_n'$. Therefore, when a model $[\![\_]\!]$ of $\lambda^{\rightarrow}$ on a cartesian closed category $\mathcal{C}$ is given, we can easily construct a model of $\lambda\&^{-\leq}$ on $\mathcal{C}$ by considering $\{V_1 \longrightarrow V_1', \ldots, V_n \longrightarrow V_n'\}$ as a $n$-product $[\![V_1 \longrightarrow V_1']\!] \times \ldots \times [\![V_n \longrightarrow V_n']\!]$, $e\&f$ as a pair $\langle [\![e]\!], [\![f]\!] \rangle$, and when $e : \{V_i \longrightarrow V_i'\}_{i \leq n}$ and $f : V_j$, $e \bullet f$ as $\mathbf{APP} \circ \langle \mathbf{snd} \circ \mathbf{fst}^{n-j} \circ [\![e]\!], [\![f]\!] \rangle$.

Now, we consider the translation from $\lambda\&$ to $\lambda\&^{-\leq}$. We first define a coercion term $c_{[U,V]}$ of type $U \longrightarrow V$ in $\lambda\&^{-\leq}$ when $U \leq V$ is inferred in $\lambda\&$.

$$c_{[U \longrightarrow U', V \longrightarrow V']} = \lambda f^{U \longrightarrow U'}.c_{[V,U]}; f; c_{[U'V']},$$
$$c_{[\{U_i \longrightarrow U_i'\}_{i \leq n}, \{V_j \longrightarrow V_j'\}_{j \leq m}]} = \lambda f^{\{U_i \longrightarrow U_i'\}_{i \leq n}}.$$
$$\&_{j \leq m}(c_{[U_{\phi(j)} \longrightarrow U'_{\phi(j)}, V_j \longrightarrow V_j']} \cdot (\lambda x^{U_{\phi(j)}}.f \bullet x)).$$

Here, $e; f$ means the composition of $e$ and $f$, and $\&_{j \leq m} e_j$ means $\epsilon \& e_1 \ldots \& e_m$. $\phi(j)$ $(j \leq m)$ are defined as follows: from the rules of subtypes, there exists for each $j \leq m$, an $i \leq n$ which satisfies $U_i \longrightarrow U_i' \leq V_j \longrightarrow V_j'$. The set of $U_i$ which satisfy this has the smallest element because of the restriction over the form of the type $\{U_i \longrightarrow U_i'\}_{i \leq n}$ in $\lambda\&$, and define $\phi(j)$ as the index of the smallest element.

We define a type-preserving translation $(\_)^{\diamond}$ from $\lambda\&$ to $\lambda\&^{-\leq}$ by inserting coercion terms where subtype relations are used. This is done following the typing rules of $\lambda\&$. Here, we only omit the translations of variables and constants which are identity.

$$\begin{array}{lcl} (\lambda x^V.e)^{\diamond} & = & \lambda x^V.e^{\diamond} \\ (e \cdot f)^{\diamond} & = & e^{\diamond} \cdot (c \cdot f^{\diamond}) \\ (e \bullet f)^{\diamond} & = & e^{\diamond} \bullet (c \cdot f^{\diamond}) \\ (e \&^F f)^{\diamond} & = & (c \cdot e^{\diamond}) \& (c \cdot f^{\diamond}) \end{array}$$

Here, the subscripts of $c$ are omitted because they are clear from the corresponding typing rules. Note that we do not need to care about the coherence problem ([BTCGS91], [BTGS90]) because there is only one proof of type assignment to each term and thus the translation is uniquely determined.

It is expected that the operational semantics of $\lambda\&$ is preserved by this translation; when $e : V$ and $e \triangleright^* e'$ in $\lambda\&$, then $e^{\diamond}$ and $c_{[U,V]} \cdot e'^{\diamond}$ are equivalent in the theory of $\lambda\&^{-\leq}$ for $U$ the type of $e'$. However, it does not hold because branch selection is based on the runtime type of the argument in $\lambda\&$, whereas it is based on the static type of the argument in $\lambda\&^{-\leq}$. It means that we cannot give semantics to $\lambda\&$ via this translation as was done in [BTCGS91] for $F_{\leq}$.

The translation of the term $y$ into $\lambda\&^{-\leq}$ is $y^{-} = z^{-} \bullet (c_{[T,S]} \cdot z^{-})$, where $z^{-} = e\&(\lambda x^S.x \bullet (c_{[S,\{i \longrightarrow i\}]} \cdot x)) : T$ with $e = \epsilon\&(\lambda x^i.x)\&(\lambda x^{i \longrightarrow i}.1)$. Here we omited coercions of the form $c_{[V,V]}$. The reduction of $y^{-}$ terminates, and it reduces to 1.

# 4 $\lambda\&C$ and $\lambda\&C^*$.

As we have noted in the introduction, Non-Subject-Reduction and late binding make the formal system of $\lambda\&$ rather complicated. Firstly, there is a restriction that an overloaded application $e \bullet f$ can only be reduced when $f$ is a normal form. Secondly, the form of an overloaded function $e_1 \&^F e_2$ is a bit tricky in that the type $F$ of this overloaded function is included in the term. Moreover, late binding makes type-theoretic study of the calculus difficult because the result of computation is affected by the type of a normal form which is not a part of the static type system. Therefore, we will consider modifications of $\lambda\&$ so that Subject Reduction holds. It is done by inserting a coercion to the argument when a term is $\beta$-reduced. We define coercions as syntactic objects and add $e|_V$ to the set of terms (Figure 3) [2].

Since the calculi we construct have the Subject Reduction property, we do not need the condition in $(\beta_{\&})$ that the argument is a normal form. We need to define reductions of coerced terms. First, since a ground type is ordered only with itself, we reduce the coercion of a ground-type term to itself[3].

As for a coercion of a function, there are two alternative ways of defining the reduction of an application of a coerced function, which result in two calculi $\lambda\&C$ and $\lambda\&C^*$. These rules reflect the meaning of an overloaded type in each system. In

[2] $e|_V$ is similar to $c_V(e)$ in $\lambda\&$-early[CGL93] . We will come back to the relation with this calculus in Section 7.

[3] It is easy to add subtyping between ground types and atomic coercions.

## $\lambda\&C$ and $\lambda\&C^*$

**Pre-Types, Subtype Relation, Restriction on Types:** Same as $\lambda\&$.

**Terms:**

$$e ::= x^V \mid d(constants) \mid \lambda x^V.e \mid e \cdot e \mid \epsilon \mid e\&e \mid e \bullet e \mid e|_V$$

**Typing rules:** Same as $\lambda\&$ with [{}Intro] replaced by [{}Intro'], plus [Coerce].

$$[\{\}\text{Intro'}]\ \frac{e : \{V_i \longrightarrow V_i'\}_{i<(n-1)} \quad f : V_n \longrightarrow V_n'}{e\&f : \{V_i \longrightarrow V_i'\}_{i\leq n}}$$

$$[\text{Coerce}]\ \frac{e : U \quad U \leq V}{e|_V : V}$$

**Reduction Rules:** $(\beta_C)$ $(\lambda x^V.e) \cdot f \triangleright e[x := f|_V]$

$$(\beta_\&')\ (e_1\&e_2) \bullet f \triangleright \begin{array}{ll} e_1 \bullet f & (e_1\&e_2 : F = \{V_i \longrightarrow V_i'\}_{i\leq n},\ f : U,\ at(F,U) < n) \\ e_2 \cdot f & (e_1\&e_2 : F = \{V_i \longrightarrow V_i'\}_{i\leq n},\ f : U,\ at(F,U) = n) \end{array}$$

$(Ground)$ $e|_A \triangleright e$ ($A$ is a ground type)

In $\lambda\&C$:

(Coerce) $e|_{V\longrightarrow V'} \cdot f \triangleright (e \cdot f)|_{V'}$

(Coerce$_\&$) $e|_F \bullet f^U \triangleright (e \bullet f)|_{V'_{at(F,U)}}$ $\quad (F = \{V_i \longrightarrow V_i'\}_{i\leq n})$

In $\lambda\&C^*$:

(Coerce$^*$) $e|_{V\longrightarrow V'} \cdot f \triangleright (e \cdot f|_V)|_{V'}$

(Coerce$^*_\&$) $e|_F \bullet f^U \triangleright (e \bullet f|_{V_{at(F,U)}})|_{V'_{at(F,U)}}$ $\quad (F = \{V_i \longrightarrow V_i'\}_{i\leq n})$

Figure 3:

$\lambda\&C$, when $e$ is a function of type $U \longrightarrow U'$, we view $e$ a function which is applicable to subtypes of $U$ and returns a value in $U'$. Therefore, we consider the function $e|_{V\longrightarrow U'}$ with $V \leq U$ as a restriction of $e$ to subtypes of $V$. Thus, $e|_{V\longrightarrow V'} \cdot f^U$ is equivalent to $(e \cdot f)|_{V'}$, and the rule (Coerce) is justified.

In $\lambda\&C^*$, we view $e : U \longrightarrow U'$ a function defined on $U$, which is also applicable to subtypes of $U$ through coercion functions. Therefore, we consider $e|_{V\longrightarrow U'}$ as a function defined on $V$, whose behavior is that of $e$ to $V$. Thus, when $e|_{V\longrightarrow V'}$ is applied to $f : U$, coerce $f$ to $V$, and apply $e|_{V\longrightarrow V'}$ to $f|_V$, which is equal to $e \cdot f|_V$. Thus the rule (Coerce$^*$) is justified. These informal semantics are naturally extended to overloaded types, which justify (Coerce$_\&$) and (Coerce$^*_\&$). The detailed explanation are given in the following sections.

Note that the coercions are inserted in the reduction rules, and therefore, the term given by a user does not include them; the same set of terms as $\lambda\&$ is used in writing programs.

**Theorem 2** *Both calculi have Subject Reduction, Unicity of Type, and the Church-Rosser properties.*

*Proof:* Unicity of Type is obvious because only one typing rule can be applied to each term. Subject Reduction can be proved by checking each rule. We will give the proof of Church Rosser property of $\lambda\&C$ using the notion of parallel reduction due to Tait and Martin-Löf following the construction in [Tak89]. The Church Rosser property of $\lambda\&C^*$ can be proved in the same way. ∎

We will study the other properties of both calculi in detail. Surprisingly, $\lambda\&C$ and $\lambda\&C^*$ have completely different properties.

# 5 Properties of $\lambda\&C^*$.

First, we show that $\lambda\&C^*$ has the Strong Normalization property.

We will define the translation $(\_)^\diamond$ from $\lambda\&C^*$ to $\lambda\&^{-\leq}$ by modifying that for $\lambda\&$ in Section 2 with the following:

$$\begin{array}{rcl} (e\&f)^\diamond & = & e^\diamond\&f^\diamond \\ e|_V & \Rightarrow & c \cdot e^\diamond \end{array}$$

We will show that this translation $(\_)^\diamond$ preserves the reduction.

**Lemma 3** *When $e \triangleright f$ in $\lambda\&C^*$, $e^\diamond$ reduces to $f^\diamond$ in one or more steps.*

*Proof.* This is proved by checking each reduction rule, using induction on the form of a term. ∎

**Theorem 4** *$\lambda\&C^*$ is Strong Normalizing.*

*Proof.* If there is an infinite reduction sequence $e_0 \triangleright e_1 \triangleright \ldots$ in $\lambda\&C^*$, then $e_0^\diamond \triangleright^+ e_1^\diamond \triangleright^+ \ldots$ in $\lambda\&^{-\leq}$. Here $\triangleright^+$ means one or more reductions. This contradicts the fact that $\lambda\&^{-\leq}$ is Strong Normalizing. ∎

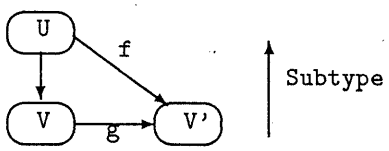
The term $y$ in Section 2 reduces as follows:

$$\begin{aligned} yz \bullet z &= (e\&(\lambda x^S.x \bullet x)) \bullet z \\ &\triangleright (\lambda x^S.x \bullet x) \cdot z \\ &\triangleright (z|_S \bullet z|_S) \\ &\triangleright (z \bullet z|_S|_{\{i \longrightarrow i\}})|_i \\ &\triangleright (\lambda x^{\{i \longrightarrow i\}}.1 \cdot z|_S|_{\{i \longrightarrow i\}})|_i \\ &\triangleright 1|_i \triangleright 1 \end{aligned}$$

Lemma 3 shows that the meaning of a term does not change by this translation. Therefore, when an extensional model of $\lambda\&^{-\leq}$ is given, we can give semantics to $\lambda\&C^*$ via this translation. From the semantics of $\lambda\&^{-\leq}$ on any cartesian closed category given in Section 3, we can consider $\{V_1 \longrightarrow V_1', \ldots, V_n \longrightarrow V_n'\}$ as a $n$-record of $V_1 \longrightarrow V_1', \ldots, V_n \longrightarrow V_n'$. From the translation $(\_)^\diamond$, we can consider $e|_V$ with $e : U$ as an application of a coercion function $c_{[U,V]}$ to $e$, and $e \cdot f$ and $e \bullet f$ as first applying a coercion function on the argument $f$ to a proper type and then apply $e$ in the sense of $\lambda\&^{-\leq}$.

Therefore, though an overloaded function of type $\{V_1 \longrightarrow V_1', \ldots, V_n \longrightarrow V_n'\}$ is applicable to all the subtypes of $V_1, \ldots, V_n$, its behavior is determined by only its behaviors on $V_1, \ldots, V_n$.

We note that even equivalent types have different meanings. For example, $\{U \longrightarrow V', V \longrightarrow V'\} \simeq \{V \longrightarrow V'\}$ when $U \leq V$. Though both types are sets of overloaded functions which are applicable to subtypes of $V$, $\{V \longrightarrow V'\}$ only include uniform functions which first coerce the argument to $V$ and then apply one function from $V$ to $V'$, whereas a function of type $\{U \longrightarrow V', V \longrightarrow V'\}$ is an ad-hoc function whose effect on $f : U$ may be different from the effect on $f|_V$. For a concrete example, consider $V' = int$ and $\lambda x^U.1\&\lambda x^V.2 : \{U \longrightarrow V', V \longrightarrow V'\}$.




Though $\lambda\&C^*$ has some good properties and clear semantics, it does not have one property which is common to calculi with coercions. That is, coercion is not transitive in $\lambda\&C^*$. For example, let $U \leq V$, $f : U \longrightarrow V'$, and $g : V \longrightarrow V'$, and consider the function $(f\&g)|_{\{V \longrightarrow V'\}}|_{\{U \longrightarrow V'\}}$ and $(f\&g)|_{\{U \longrightarrow V'\}}$. As we have seen, the behavior of this function is determined by its behavior on $U$. Let $e : U$. then

$$\begin{aligned} &(f\&g)|_{\{V \longrightarrow V'\}}|_{\{U \longrightarrow V'\}} \bullet e \\ &\quad \triangleright ((f\&g)|_{\{V \longrightarrow V'\}} \bullet e|_U)|_{V'} \\ &\quad \triangleright ((f\&g) \bullet e|_U|_V)|_{V'}|_{V'} \\ &\quad \triangleright (g \cdot e|_U|_V)|_{V'}|_{V'}. \end{aligned}$$

On the other hand,

$$\begin{aligned} &(f\&g)|_{\{U \longrightarrow V'\}} \bullet e \\ &\quad \triangleright ((f\&g) \bullet e|_U)|_{V'} \\ &\quad \triangleright (f \cdot e|_U)|_{V'}. \end{aligned}$$

Therefore, these two functions have different behaviors. This fact shows that we cannot add the following reduction rule:

$$\text{(E-CC)} \quad e|_U|_V \triangleright e|_V.$$

If we add this reduction rule, not only the Church-Rosser property, but also the Strong Normalization property does no longer hold. Consider the term $y$ with $\lambda x^S.x \bullet x$ replaced by $\lambda x^S.x|_T \bullet x$.

# 6 Properties of $\lambda\&C$.

Compared with $\lambda\&C^*$, $\lambda\&C$ has complicated syntactic and semantic properties. As was the case in $\lambda\&$, $\lambda\&C$ does not have the strong normalization property.

The term $y$ defined for $\lambda\&$ reduces as follows:

$$\begin{aligned} y = z \bullet z &= (e\&(\lambda x^S.x \bullet x)) \bullet z \\ &\triangleright (\lambda x^S.x \bullet x) \cdot z \\ &\triangleright (z|_S \bullet z|_S) \\ &\triangleright (z \bullet z|_S)|_i \\ &\triangleright \ldots. \\ &\triangleright (z|_S|_S \bullet z|_S|_S)|_i \\ &\triangleright \ldots. \end{aligned}$$

Comparing this reduction with that in $\lambda\&C^*$, it is clear that the reduction in $\lambda\&C$ is not preserved by the translation $(\_)^\diamond$ into $\lambda\&^{-\leq}$, and it is impossible to give semantics to $\lambda\&C$ through this translation.

Actually, the meaning of $\{V \longrightarrow V'\}$ is different from that in $\lambda\&C^*$. In $\lambda\&C^*$, $\{V \longrightarrow V'\}$ is the set of overloaded functions with only one branch of type $V \longrightarrow V'$, and thus the behavior of $e : \{V \longrightarrow V'\}$ is determined by its behavior on $V$. However,

it is not the case in $\lambda\&C$. Consider the case $U \leq V$. $\{U \longrightarrow V', V \longrightarrow V'\} \leq \{V \longrightarrow V'\}$ (actually, they are equivalent). Therefore, for $f : U \longrightarrow V'$ and $g : V \longrightarrow V'$, $(f\&g)|_{\{V \longrightarrow V'\}}$ is a term of type $\{V \longrightarrow V'\}$. From the reduction rule, $(f\&g)|_{\{V \longrightarrow V'\}} \bullet e \triangleright ((f\&g) \bullet e)|_{V'}$. Therefore, $f$ is applied when the argument type is a subtype of $U$, and $g$ is applied if the argument type is a subtype of $V$. It means that the behavior of a function of type $\{V \longrightarrow V'\}$ is not determined by its behavior on $V$.

Thus a question arises how an overloaded function behaves in $\lambda\&C$.

In $\lambda\&C^*$, the transitivity of coercions holds. That is, $e|_U|_V$ and $e|_V$ are equivalent. Note that our calculi are not extensional, and we do not intend to extend them to extensional calculi. For example, $(\lambda x^V.x)|_{V \longrightarrow V}$ and $\lambda x^V.x$ are different normal forms with the same functionality. Instead, we observe equivalence on ground types.

**Definition 1** *Two closed terms $e$ and $f$ are congruent ($e \approx f$) if for every closed context $C[X]$ of a ground type, $C[e]$ is well-typed and reduces to a normal form $c$ iff $C[f]$ is well-typed and reduces to $c$ ([Blo90]).*

**Theorem 5** *Coercion is transitive in $\lambda\&$. That is, when $F \leq G \leq U$ and $e$ is a closed term of type $F$, $e|_G|_U$ and $e|_U$ are congruent.*

The proof is rather long. We need to introduce the notion of applicable congruence and prove Operationally Extensionality ([Blo90]) that congruence and applicable congruence coinside, and then prove the applicable congruence of $e|_G|_U$ and $e|_U$. For the proof of Operationally Extensionality, we need to use induction on the length of the leftmost reduction, for which we also need to prove the leftmost reduction theorem of this calculus. The details are given in [Tsu93].

We can also prove that when two types $U$ ang $V$ are equivalent (i.e., $U \leq V$ and $U \geq V$, written $U \simeq V$), we can give the same meaning to $U$ and $V$.

**Theorem 6** *When $U \simeq V$ and $e$ is a closed term of type $U$, $e$ and $e|_V$ are congruent.*

Now, we can discuss the semantics of $\lambda\&$. We will consider, as the meaning of $V$, $\mathcal{D}(V)$: the quotient by $\approx$ of closed terms of type $V$ [4]. We will write $\overline{e}$ for the equivalence class of $e$.

We write $\mathcal{T}$ for the set of all type expressions, and $\overline{\mathcal{T}}$ for the partial ordered set obtained by taking the quotient by $\simeq$. We write $\overline{V}$ for the equivalence class of $V$. From Theorem 6, we know that when $U \simeq V$, we can identify $\mathcal{D}(U)$ and $\mathcal{D}(V)$. Therefore, from now on, we consider $\mathcal{D}$ a function from $\overline{\mathcal{T}}$.

Thus, we have constructed a poset $\overline{\mathcal{T}}$ and a function $\mathcal{D}$ from $\overline{\mathcal{T}}$ to **Sets**: the class of all sets. When $e \approx f$ are two terms of type $U$, then $e|_V \approx f|_V$ for $U \leq V$ from the definition of congruence. Therefore, we can define a function $\mathcal{D}(|_V)$ from $\mathcal{D}(\overline{U})$ to $\mathcal{D}(\overline{V})$ when $U \leq V$. Moreover, from Theorem 5, we have that $\mathcal{D}$ is a contravariant functor from the poset $\overline{\mathcal{T}}$ considered as a category to **Sets**. That is, $\mathcal{D}$ is a coindexed category. Here, we define the direction of an arrow as $\overline{U} \leftarrow \overline{V}$ when $\overline{U} \leq \overline{V}$

Now, we consider the meaning of an overloaded type $F$ on this indexed category. For an overloaded type $F = \{V_1 \longrightarrow V_1', \ldots, V_n \longrightarrow V_n'\}$, we can construct a partial function $\mathcal{G}(F)$ from $\mathcal{T}$ to $\mathcal{T}$ which sends $U \in \mathcal{T}$ to $V'_{at(F,U)}$. That is, a function which maps the argument type to the result type. The first condition on the form of an overloaded type ensures that it is a monotonic function by considering $\mathcal{T}$ as a pre-ordered set. This monotonicity ensures that we can consider $\mathcal{G}(F)$ as a monotonic function from the poset $\overline{\mathcal{T}}$ to $\overline{\mathcal{T}}$.

Though the subtyping relation between overloaded types seems complicated, it has the following soundness and completeness theorem for this interpretation.

**Theorem 7** *$F \geq G$ iff $\mathcal{G}(F) \geq \mathcal{G}(G)$ by the pointwise order on the function space from $\overline{\mathcal{T}}$ to $\overline{\mathcal{T}}$.*

From this theorem, two overloaded types $F$ and $G$ are equivalent iff $\mathcal{G}(F)$ and $\mathcal{G}(G)$ are the same function. Thus we can consider $\mathcal{G}$ a function from $\overline{\mathcal{T}_O}$, where $\mathcal{T}_O$ is the set of overloaded types. We add a least element $\bot$ which means type error to the poset $\overline{\mathcal{T}}$ and define $\overline{\mathcal{T}}_\bot$. Thus we consider $\mathcal{G}(\overline{F})$ as a total monotonic function by assigning $\bot$ when $\mathcal{G}(\overline{F})$ is undefined. It is obvious that $\mathcal{G}(\overline{F})$ is a step function from $\overline{\mathcal{T}}_\bot$ to $\overline{\mathcal{T}}_\bot$, and every step function from $\overline{\mathcal{T}}_\bot$ to $\overline{\mathcal{T}}_\bot$ is $\mathcal{G}(\overline{F})$ for some overloaded type $F$. Therefore, we can identify $\overline{\mathcal{T}_O}$ with the set of all step functions from $\overline{\mathcal{T}}_\bot$ to $\overline{\mathcal{T}}_\bot$. We write $[\overline{\mathcal{T}}_\bot \longrightarrow \overline{\mathcal{T}}_\bot]$ for this set.

As for the meaning of terms, we define $\mathcal{D}(\bot) = \{*\}$; where $*$ is the term which denote a type error, and consider $\mathcal{D}$ a functor from $\overline{\mathcal{T}}_\bot$. We consider the meaning of an overloaded function over the Grothendieck Construction of the indexed category ($\overline{\mathcal{T}}_\bot \xrightarrow{\mathcal{D}}$ **Sets**). That is, consider the disjoint union $\mathcal{U}$ of all $\mathcal{D}(t)$ with $t \in \overline{\mathcal{T}}_\bot$, and a projection $\mathcal{A}$ from $\mathcal{U}$ to $\overline{\mathcal{T}}_\bot$. $\mathcal{U}$ is intuitively the set of all values of all types and $\mathcal{A}$ is a type assignment function. The order (category) structure of $\mathcal{U}$ is induced by $\mathcal{D}(|_V)$; that is, the coercible relation on $\mathcal{U}$.

[4] Though we can consider on $\mathcal{D}(V)$ an order structure derived from the termination property, we do not consider it here.

When $e : F$ is an overloaded function, $e$ determines a function from $\mathcal{U}$ to $\mathcal{U}$, which maps an argument to the result and an unapplicable element to $*$. We write $\mathcal{G}(\overline{e})$ for this function. $\mathcal{G}(\overline{e})$ makes the following diagram commute:

$$\begin{array}{ccc} \mathcal{U} & \xrightarrow{\mathcal{G}(\overline{e})} & \mathcal{U} \\ \downarrow{\scriptstyle \mathcal{A}} & & \downarrow{\scriptstyle \mathcal{A}} \\ \overline{T} & \xrightarrow{\mathcal{G}(\overline{F})} & \overline{T} \end{array}$$

We will write $\lceil \mathcal{U} \longrightarrow \mathcal{U} \rceil_{\mathcal{G}(\overline{F})}$ for the set of functions from $\mathcal{U}$ to $\mathcal{U}$ which make the diagram commute. Note that, $\mathcal{G}(\overline{e})$ is a function from $\mathcal{U}$ to $\mathcal{U}$, but *not a functor from $\mathcal{U}$ to $\mathcal{U}$*. That is, $\mathcal{G}(\overline{e})$ *does not preserve the order structure on $\mathcal{U}$ which is the coercibleness relation between values.* This is the dirty side of overloading.

Thus, we may consider the meaning of an overloaded type $F$ as $\lceil \mathcal{U} \longrightarrow \mathcal{U} \rceil_{\mathcal{G}(\overline{F})}$. However, *it is not a denotational model* in the true sense because $\mathcal{D}(\overline{F})$, which is a subset of $\mathcal{U}$, and $\lceil \mathcal{U} \longrightarrow \mathcal{U} \rceil_{\mathcal{G}(\overline{F})}$ are different. Now a question arises whether there is a truly denotational model of $\lambda\&C$. In [Tsu92], the author has studied $\lambda_m$, another calculus with overloading and subtyping. In $\lambda_m$, an overloaded function works more uniformly in that when more than two branches are applicable, it applies all the branches and then merge the result. In this way, $\mathcal{G}(e)$ becomes a functor, and he has succeeded in constructing a semantic domain by solving a domain equation over opfibrations. In the case of $\lambda\&C$, it seems that a similar construction is almost impossible. This is why we considered a term model and study its property. [5] We leave open whether there is a true denotational model for $\lambda\&C$.

# 7 Related Works and Further Works

We will summarize the properties of calculi with overloading.

Castagna, Ghelli, and Longo has also presented $\lambda\&$-early as well as $\lambda\&$[CGL93]. This calculus is an early binding version of $\lambda\&$-calculus, which uses the notion of coercions to freeze the type of a term. To use our syntax, the terms of $\lambda\&$-early have the forms:

$$e ::= x^V \mid \lambda x^V.e \mid e \cdot e \mid e|_V \mid \epsilon \mid e\&^F e \mid e \bullet e|_V$$

and in addition to $(\beta)$ and $(\beta_\&)$, it has the reduction rule

$$(coerce) \quad e|_V \circ f \rhd e \circ f$$

where $\circ$ is a $\bullet$ or a $\cdot$. The crucial difference between $\lambda\&$-early and $\lambda\&C$ is that a programmer is expected to insert coercions to overloaded applications in $\lambda\&$-early, where coercion is inserted when a term is $\beta$-reduced in $\lambda\&C$.

$\lambda\&$-early does not have the Subject Reduction property, nor Strong Normalization. A non-normalizing term is $y' = z' \bullet z'|_S$, with $z' = (e\&^T(\lambda x^S.x \bullet x|_S))$.

We will define a translation $(\_)^\diamond$ from $\lambda\&$-early to $\lambda\&C$ which inserts coercions on each component of $\&$.

$$(e_0\&^F e_1)^\diamond = (e_0^\diamond|_{V_0})\&(e_1^\diamond|_{V_1})$$

The other cases are trivially defined. This translation preserves reductions in the following sense.

**Theorem 8** *If $e \rhd^* e'$ in $\lambda\&$-early with $e : V$, $e' : U$, then $e^\diamond$ and $e'^\diamond|_V$ are congruent in $\lambda\&C$.*

Now, the relations between the calculi are listed in Figure 4. Here, SN means Strong Normalization, and SR means Subject Reduction. Calculi on the same column means that there is a translation from the above calculus to the calculus below.

| | | Late Binding | Early Binding | |
|---|---|---|---|---|
| | | | Not SN | SN |
| Not SR | Subtyping | $\lambda\&$ | $\lambda\&$-early | |
| SR | | | $\lambda\&C$ | $\lambda\&C^* = \lambda\&G$ |
| | No Subtyping | | | $\lambda\&^{-\leq}$ |

Figure 4: Table of calculi

From this table, one may expect that we can construct a normalizing calculus $\lambda\&^*$-early at the location above $\lambda\&G$. Consider a calculus $\lambda\&^*$-early which is the same as $\lambda\&$-early except that the rule (coerce) is replaced by the following two rules.

$$(\text{coerce}^*) \quad e|_{V \longrightarrow V'} \cdot f \rhd (e \cdot f|_V)$$
$$(\text{coerce}^*_\&) \quad e|_F \bullet f^U \rhd (e \bullet f|_{V_{at(F,U)}}) \quad (F = \{V_i \longrightarrow V_i'\}_{i \leq n})$$

It is expected that $\lambda\&^*$-early can be translated into $\lambda\&C^*$. However, it is not the case. Consider the term $(\lambda x^S.x \bullet e|_S) \cdot z$ with $z = \epsilon\&(\lambda x^i.x)\&(\lambda x^{\{i \longrightarrow i\}}.1)\&(\lambda x^S.2)$, and $e$ any term of type $S$. It reduces in $\lambda\&^*$-early to $z \bullet e|_S$ and then to 2, where the same term reduces in $\lambda\&C^*$ to $z|_S \bullet e|_S \rhd z \bullet e|_S|_{\{i \longrightarrow i\}}$, and then to 1. It is left open whether $\lambda\&^*$-early is normalizing or not.

[5] In [CGL93], they tried to give a per model of $\lambda\&$-early, which is, as we will see, closely related to $\lambda\&C$. However, their semantics construction only applies to a stratified subcalculus of $\lambda\&$-early, and does not apply to $\lambda\&$-early.

In [CGL94], in order to settle the non-normalizing problem of $\lambda\&$, they introduced stratified subcalculus of $\lambda\&$ which is Strong Normalizing. In this paper, we tackled this problem differently; instead of restricting the terms, we modified the reduction rule and gave different semantics, so that it is Strong Normalizing. There are some more topics that the author could not expand due to the restriction of the space.

One thins is that branch selected in overloaded applications in $\lambda\&C$ can be determined statically, which is not the case in $\lambda\&C^*$. Therefore, we can drop type information from $\lambda\&C^*$ at runtime, and can compile $\lambda\&C^*$ into an untyped calculus. This enables efficient implimemtation of the language.

Another things is that, the semantics of $\lambda\&C^*$ ensures that we can identify $\{V \longrightarrow V'\}$ with $V \longrightarrow V'$. It is practically important because one can pass an overloaded function whose type is a subtype of $\{V \longrightarrow V'\}$ to a function which requires an argument of type $V \longrightarrow V'$. In the full version of the paper, we have simplified the syntax of $\lambda\&C^*$, by this unification, and defined a calculus $\lambda\&G$.

The third point is late bindings. Late binding is one of the characteristics of object orientedness, and actually, we cannot express many useful features of object orientedness like virtual classes without this. In $\lambda\&$, late binding was implemented with the loss of subject reduction, which means that the type system does not know exact type of a term, and does not know which branch will be executed. The author is trying to add late binding to $\lambda\&G$, in a way that Subject Reduction also holds, by expressing explicitly that the argument type of a function is a subtype of some type and bind the exact type of the argument to a type variable when the function is called. For this purpose, we need to restrict inheritance relation to a kind of record extensions of Ohori and others([OB89]). The fact that it has Unicity of Type and Subject Reduction properties means that we can infer the exact type of a term, and when we infer a type which do not include a type variable, we can avoid dynamic method search and statically bind the method at compile-time. Many practical typed object oriented languages like C++ and Kuno's Misty combine dynamic method search and compile-time method binding. The author thinks that this will lead to a strong type system for this kind of languages.

**Acknowledgement** The author thanks Benjamin Pierce for valuable comments and stimulating discussions. Thanks also to Giuseppe Castagna and Giorgio Ghelli for discussions over E-mail about normalization properties of $\lambda\&$-calculus and $\lambda\&$-early.

# References

[Blo90] Bard Bloom. Can lcf be topped?, flat lattice models of typed $\lambda$-calculus. *Info. and Comput.*, Vol. 87, pp. 264–301, 1990.

[BTCGS91] V. Breazu-Tannen, T. Coquand, C. A. Gunter, and A. Scedrov. Inheritance as implicit coercion. *Info. and Comput.*, Vol. 93, pp. 172–221, 1991.

[BTGS90] V. Breazu-Tannen, C. A. Gunter, and A. Scedrov. Computing with coercions. In *Proc. ACM Conf. on Lisp and Functional Prog.*, pp. 44–59, 1990.

[CGL92] G. Castagna, G. Ghelli, and G. Longo. A calculus for overloaded functions with subtyping. In *Proc. ACM Conf. on LISP and Functional Prog.*, 1992. Extended abstract of [CGL94].

[CGL93] G. Castagna, G. Ghelli, and G. Longo. The semantic of $\lambda\&$-early: a calculus with overloading and early binding. In *Int. conf. on typed lambda calculi and applications. LL-NCS664*, 1993.

[CGL94] G. Castagna, G. Ghelli, and G. Longo. A calculus for overloaded functions with subtyping. *Info. and Comput.*, Vol. to appear,, 1994.

[GLT89] J. Y. Girard, Y. Lafont, and P. Taylor. *Proof and Types.* Cambridge University Press, 1989.

[OB89] A. Ohori and P. Buneman. Static type inference for parametric classes. In *Proc., OOPSLA89*, pp. 445–455, 1989.

[Tak89] Masako Takahashi. Parallel reductions in $\lambda$-calculus. *J. Symbolic Computation*, Vol. 7, pp. 113–123, 1989. Also *Parallel Reductions in $\lambda$-Calculus*, Revised version (Research Report, Tokyo Institute of Technology, 1992).

[Tsu92] Hideki Tsuiki. *A Record Calculus with a Merge Operator.* PhD thesis, Keio University, 1992.

[Tsu93] Hideki Tsuiki. A normalizing calculus with overloading and subtyping. Technical report, Keio University, SFC, 1993.