Characterizing strong normalization in a language with control operators.

Dan Dougherty, Silvia Ghilezan, Pierre Lescanne

▶ **To cite this version:**

Dan Dougherty, Silvia Ghilezan, Pierre Lescanne. Characterizing strong normalization in a language with control operators.. [Research Report] LIP RR-2004-29, Laboratoire de l'informatique du parallélisme. 2004, 2+26p. hal-02101950

HAL Id: hal-02101950

https://hal-lara.archives-ouvertes.fr/hal-02101950v1

Submitted on 17 Apr 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON-UCBL
n° 5668

Characterizing strong normalization in a language with control operators

Dan Dougherty
Silvia Ghilezan
Pierre Lescanne

May 2004

Research Report N° RR2004-29

École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : lip@ens-lyon.fr

Characterizing strong normalization in a language with control operators

Dan Dougherty
Silvia Ghilezan
Pierre Lescanne

May 2004

Abstract

We investigate some fundamental properties of the reduction relation in the untyped term calculus derived from Curien and Herbelin's $\overline{\lambda}\mu\tilde{\mu}$. The original $\overline{\lambda}\mu\tilde{\mu}$ has a system of simple types, based on sequent calculus, embodying a Curry-Howard correspondence with classical logic; the significance of the untyped calculus of raw terms is that it is a Turing-complete language for computation with explicit representation of control as well as code. We define a type assignment system for the raw terms satisfying: a term is typable if and only if it is strongly normalizing. The intrinsic symmetry in the $\overline{\lambda}\mu\tilde{\mu}$ calculus leads to an essential use of both intersection and union types; in contrast to other union-types systems in the literature, our system enjoys the Subject Reduction property.

Keywords: classical logic, sequent calculus, semantic, lambda calculus, continuations, strong normalization

Résumé

Nous examinons quelques propri´et´es fondamentales de la relation de r´eduction dans le calcul sans types d´eriv´e du calcul $\overline{\lambda}\mu\tilde{\mu}$ de Curien et Herbelin. Le calcul originel $\overline{\lambda}\mu\tilde{\mu}$ possède un système de types simple fond´es sur le calcul des s´equents et impliquant une correspondance de Curry-Howard avec la logique classique. Le calcul sans type, dit des termes bruts, est un langage complet au sens de Turing qui contient une repr´esentation explicite du contrôle aussi bien que du code et c'est cela sa sp´ecificit´e. Nous d´efinissons sur ces termes bruts un système de types qui satisfait la condition suivante: un terme est typable si et seulement si il est fortement normalisable. La sym´etrie intrinsèque de $\overline{\lambda}\mu\tilde{\mu}$ conduit à une utilisation essentielle de l'union et de l'intersection dans les types. Contrairement à cuex que rapporte la litt´erature, notre système a la propri´et´e de r´eduction du sujet.

Mots-clés: logique classique, calcul des s´equentss´emantique, lambda calcul, continuation, normlisation forte

1 Introduction

The Curry-Howard correspondence has long been a linchpin of the connection between logic and computer science. It was originally articulated in the context of intuitionistic logic, but Griffin extended the Curry-Howard correspondence to classical logic in his seminal 1990 POPL paper [14], by observing that classical tautologies suggest typings for certain control operators. This initiated an active line of research; in particular the $\lambda\mu$ calculus of Parigot [21] has been the foundation of a number of investigations [22, 12, 19, 6, 1] into the relationship between classical logic and theories of control in programming languages.

Meanwhile Curien and Herbelin [11], building on earlier work in [15], defined the system $\overline{\lambda}\mu\tilde{\mu}$. In contrast to Parigot's $\lambda\mu$-calculus, which bases its type system on a natural deduction system for classical logic, terms in $\overline{\lambda}\mu\tilde{\mu}$ represent derivations in a *sequent calculus* proof system and reduction reflects the process of cut-elimination. As described in [11], the sequent calculus basis for $\overline{\lambda}\mu\tilde{\mu}$ supports an interpretation of the reduction rules of the system as operations of an abstract machine. In particular, the right- and left-hand sides of a sequent directly represent the *code* and *environment* components of the machine. This perspective is elaborated more fully in [10].

From logic to computation: two roles of typing

In this paper we relax the connection with logic, and explore the computational properties of pure (untyped) $\overline{\lambda}\mu\tilde{\mu}$.

A useful perspective emerges if we compare the present project to the study of types in the standard λ-calculus. Type systems have been used in order to interpret λ-terms as defining set-theoretic functions (simple types) and later to enforce data abstraction (dependent and polymorphic types). Roughly, this use of types enables the λ-calculus to be used as an *applied* calculus. But in another direction, type systems were developed to study the reduction behavior of λ-terms and the structure of models. Intersection types play a central role in this analysis. Key results here are the characterizations of terms that are solvable, normalizing, and strongly normalizing in terms of their possible typings [8, 26], and the completeness results for set-theoretic semantics [4]. In a precise sense the paradigms of types-as-propositions and types for operational and denotational semantics are skew to each other: as pointed out by Hindley [16], there does not seem to be any standard logical notion that corresponds to intersection.

The current work is a first step in a programme that is firmly in the second tradition: we want to explore the basic combinatorial properties of the reduction-relation in the $\overline{\lambda}\mu\tilde{\mu}$ calculus. In this paper we present an analysis of strong normalization.

It has often been observed that an advantage the sequent calculus has over natural deduction is the the fact that it better exhibits the *symmetries* inherent in logic. This is most apparent in classical logic, and indeed this symmetry is at the heart of the duality between call-by-name and call-by-value that motivated the definition of $\overline{\lambda}\mu\tilde{\mu}$ originally; Wadler has recently clarified and strengthened this duality in [29]. In this paper we are not interested in a *logical* interpretation of $\overline{\lambda}\mu\tilde{\mu}$. Still, the importance of symmetries in the analysis of $\overline{\lambda}\mu\tilde{\mu}$ is manifested in the fact that union types are an essential feature of our type system. See [9] for a discussion of the importance of symmetries in computation. A discussion of the intuitions behind the type system of $\overline{\lambda}\mu\tilde{\mu}$ is at the beginning of Section 3.

Summary of results

The main contribution of this paper is the characterization of the strongly normalizing terms of pure $\overline{\lambda}\mu\tilde{\mu}$ under both the call-by-name and call-by-value disciplines. In particular, the set of terms that are strongly normalizing in $\overline{\lambda}\mu\tilde{\mu}$ under the call-by-name and the call-by-value disciplines coincide. In fact our proof applies to a rather general class of reduction systems, namely any reduction discipline expressing a notion of *priority* (see Definition 6.1) in which individual *terms* each consistently act as call-by-name or as call-by-value (different terms may adopt different "modes").

The characterization is in terms of a type system incorporating both intersection and union types: the strongly normalizing terms are precisely the typable ones. Such a characterization has been done for traditional λ-calculus using intersection types only but thus far no attempt has been made for extending it to the classical-logic inspired languages with control operators.

Type systems with union types have been studied before, but the systems we are aware of suffer from a severe handicap: the Subject Reduction property fails for these systems, at least without some constraints on the reductions. In contrast, the calculus here enjoys Subject Reduction for unrestricted reduction.

The type-system we define is itself presented in sequent style; in particular we use only rules for intersection- and union-introduction.

Related work

Curien and Herbelin [11] encode simply-typed call-by-name and call-by-value $\overline{\lambda}\mu\tilde{\mu}$ into the simply-typed λ-calculus via CPS translations: this implies strong normalization for these reductions. In [18] Lengrand shows how simply-typed $\overline{\lambda}\mu\tilde{\mu}$ and the calculus of Urban and Bierman [28] are mutually interpretable, so that the strong normalization proof of the latter calculus yields another proof of strong normalization for simply-typed $\overline{\lambda}\mu\tilde{\mu}$. Our proof of strong normalization is direct, using a natural adaptation of the reducibility method for the λ-calculus.

The unrestricted reduction relation in $\overline{\lambda}\mu\tilde{\mu}$ has a critical pair, and indeed this system is not confluent. This critical pair is a reflection of the inherent symmetry in the system, but it complicates reasoning about reduction. Polonovski [24] presents a proof of SN for the unrestricted calculus with a method based on the "symmetric candidates" idea of Barbanera and Berardi [2] (actually Polonovski treats a version of $\overline{\lambda}\mu\tilde{\mu}$ with explicit substitutions). The interaction between intersection types and symmetric candidates is problematic, and strong normalization for intersection-typable terms under reduction that reflects no notion of priority is at present still a conjecture.

Remarkably, Laurent [17] has recently and independently set out to analyze the denotational semantics of the $\lambda\mu$ calculus by defining a type system quite similar to ours: in particular his system involves both intersection and union types. Since he is interested in semantics his system naturally has the property that typings are closed under subject conversion (for example there is a universal type). It will be interesting to investigate the relationship between these systems.

The larger context of related research includes a wealth of work in logic and programming languages. Several term-assignment systems for sequent calculus have been proposed as a tool for studying the process of cut-elimination [25, 5, 28]. In these systems — with the exception of the one in [28] — terms do not unambiguously encode sequent derivations.

We described above the fundamental importance of intersection types for λ-calculus.

In the 1980's and early 1990's Reynolds explored the role that intersection types can play in a practical programming language (see for example the report [27] on the language Forsythe). Pierce [23] explored the use of union types in programming, for example as a generalization of variant records. Buneman and Pierce [7], have shown how union types can play a key role in the design of query languages for semistructured data union types. The motivation for the work in Barbanera et al. [3] was foundational, motivated by the observation that union types arise naturally in denotational semantics and that they can generate more informative types for some terms. This latter work highlighted the failure of Subject Reduction in the presence of union types and showed how to recover this property by suitably restricting the notion of reduction. Union and intersection types have recently been used by Palsberg and Pavlopoulou [20] and subsequently by Wells, Dimock, Muller, and Turbak [30] in a type system involving *flow types* for encoding control and data flow information in typed program representations. The system in [30] obeys the Subject Reduction for a certain call-by-value version of the β-rule (in which variables are not considered values).

2 Syntax

In this section we present the untyped version of Curien and Herbelin's $\overline{\lambda}\mu\widetilde{\mu}$ calculus introduced in [11]. We identify three syntactic categories: the set of *callers*, the set of *callees*, and the set of *capsules* (in [11] these are referred to respectively as *terms*, *contexts* and *commands*). We changed the terminology because the word *terms* seems inappropriate to qualify only some of the expressions of the language, because *commands* are in no way "commands". And last a terminology that emphasizes the inherent symmetry of the language is more than welcome.

There are two kinds of variables in this system: (i) the set Var_r of *caller variables* denoted by Latin variables x, y,..., which can be bound by λ-abstractions or $\widetilde{\mu}$-abstractions, and (ii) the set Var_e of *callee variables* denoted by Greek variables α, β,... which can be bound by μ-abstractions. Letting r, e, and c range over callers, callees and capsules respectively, we have

$$r ::= x \mid \lambda x.r \mid \mu\alpha.c \qquad e ::= \alpha \mid r \bullet e \mid \widetilde{\mu}x.c \qquad c ::= \langle r \parallel e \rangle$$

The reduction rules of the calculus are

$$\begin{array}{lrcl} (\lambda) & \langle \lambda x.r \parallel r' \bullet e \rangle & \longrightarrow & \langle r' \parallel \widetilde{\mu}x.\langle r \parallel e \rangle \rangle \\ (\mu) & \langle \mu\alpha.c \parallel e \rangle & \longrightarrow & c[\alpha \leftarrow e] \\ (\widetilde{\mu}) & \langle r \parallel \widetilde{\mu}x.c \rangle & \longrightarrow & c[x \leftarrow r] \end{array}$$

Of course the substitutions above are defined so as to avoid variable capture. The formal definitions of free and bound variables are as expected. In this paper, we use the usual convention on variables that in a statement or an expression, there is no subexpression in which a variable is both free and bound.

As a rewriting calculus $\overline{\lambda}\mu\widetilde{\mu}$ has an essential critical pair between the μ and the $\widetilde{\mu}$ redexes. That is to say, on a term of the form $\langle \mu\alpha.c \parallel \widetilde{\mu}x.c \rangle$ rules (μ) and $(\widetilde{\mu})$ can be applied ambiguously. If one gives priority to (μ) over $(\widetilde{\mu})$ this corresponds to a *call-by-value* discipline, otherwise it is *call-by-name*. Indeed the calculus is inherently not confluent. As a simple example observe that the capsule $\langle \mu\alpha.\langle z_1 \parallel \beta_1 \rangle \parallel \widetilde{\mu}x.\langle z_2 \parallel \beta_2 \rangle \rangle$ reduces to each of $\langle z_1 \parallel \beta_1 \rangle$ and $\langle z_2 \parallel \beta_2 \rangle$.

This is more than simply a reflection of the well-known fact that the equational theories of call-by-name and call-by-value differ. It is a reflection of the great expressive

power of the language: a single term containing several capsules can encompass several complete computational processes, and the μ and $\tilde{\mu}$ reductions allow free transfer of control between them.

So the combinatorics of pure reduction is very complex. In this light it is perhaps slightly surprising that the strongly normalizing computations can so readily be characterized, via the type system we present later.

When reduction in $\overline{\lambda}\mu\tilde{\mu}$ is constrained to commit to the call-by-name discipline or to the call-by-value, the system is confluent.

It is not hard to see that pure $\overline{\lambda}\mu\tilde{\mu}$ is Turing-complete as a programming language, since the untyped λ-calculus can be coded easily into it. Space does not permit a formal development here, but it is instructive to note the following general example.

Notation. If m and n are callers, let $m * n$ denote the caller term $\mu\alpha.\langle m \parallel n \bullet \alpha\rangle$. (Of course α is not free in m or n here.)

Example. (Classical beta-reduction)

$$\begin{array}{lcl} (\lambda x.r) * s & \equiv & \mu\alpha.\langle \lambda x.r \parallel s \bullet \alpha\rangle \\ & \twoheadrightarrow & \mu\alpha.\langle r[x \leftarrow s] \parallel \alpha\rangle \end{array}$$

using a λ-step followed immediately by a $\tilde{\mu}$-step.

From the reduction rules, one deduces easily that the *normal forms* are generated by the following abstract syntax.

$$\begin{array}{lcl} r_{nf} & ::= & x \mid \lambda x.r_{nf} \mid \mu\alpha.c_{nf} \\ e_{nf} & ::= & \alpha \mid r_{nf} \bullet e_{nf} \mid \tilde{\mu}x.c_{nf} \\ c_{nf} & ::= & \langle x \parallel \alpha\rangle \mid \langle x \parallel r_{nf} \bullet e_{nf}\rangle \mid \langle \lambda x.r_{nf} \parallel \alpha\rangle \end{array}$$

3 Type assignment with intersection and union

The form of classical sequent calculus provides the framework for the definition of a type-assignment system for $\overline{\lambda}\mu\tilde{\mu}$ using simple types. This is precisely the type system of Curien and Herbelin [11], which will be the foundation upon which we build our intersection types. Indeed, the system of Curien and Herbelin comprises precisely the rules in Figure 1 that do not mention intersections or unions.

When the sequent-based type system is viewed a means of assigning terms to sequents, we are led to the notion of *the active proposition* in a sequent, that is, the proposition (type) that "carries" the term. This is the proposition in *the stoup* in Girard's sense [13]. In each rule of Figure 1 , the active proposition is boxed. When applying a rule, one has to take into account where the active proposition is.

It is well-known that the distinction between intuitionistic and classical logic is reflected in sequent calculus by the admission, in the classical case, of multiple formulas on the right side of a sequent. But the use of sequents as typings for $\overline{\lambda}\mu\tilde{\mu}$ terms — μ-terms in particular — yields another insight. For the purposes of this discussion let us agree to introduce the propositional constant $\perp$ to denote the space of "answers" or "responses". The informal interpretation of a caller typing judgment $r : A$ is that r denotes a *value* in type A; correspondingly the informal interpretation of a callee typing judgment $e : A$ is that e denotes a *continuation* that expects a value of type A and returns an answer. Under this reading a judgment such as $\mu\alpha.c : A$ says that $\mu\alpha.c$

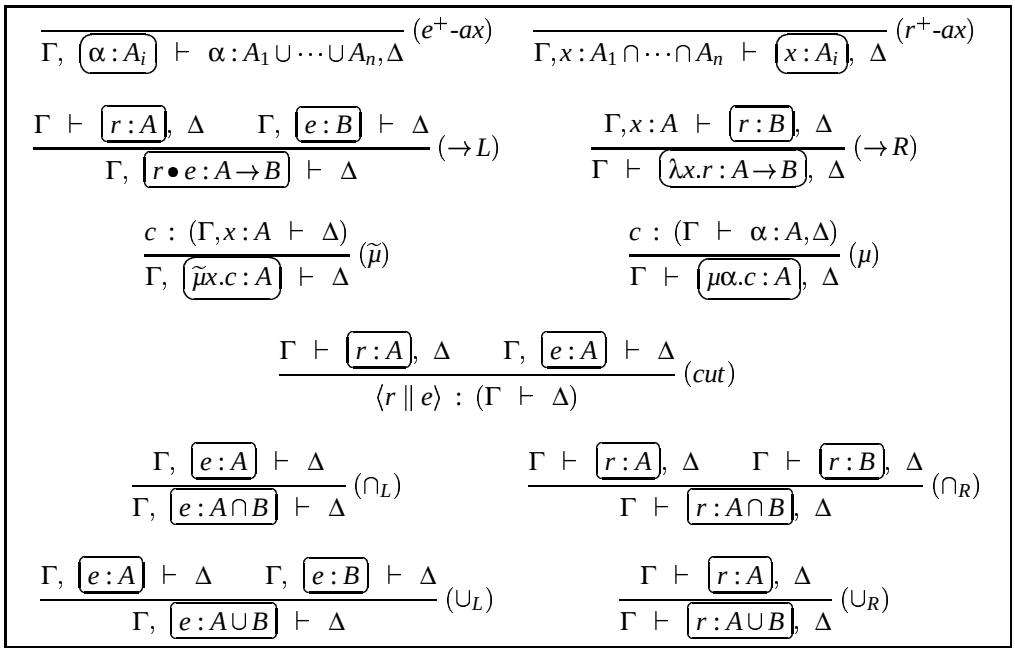

Figure 1: The type system $\mathcal{M}^{\cap\cup}$

takes as parameter an A-continuation and returns a value. If we were to informally denote the space of A-continuations as the set $(A \Rightarrow \bot)$ then this $\mu\alpha.c$ inhabits the set $((A \Rightarrow \bot) \Rightarrow \bot)$, and the fact that such terms are assigned the *type* A is exactly the embodiment of the equivalence of a proposition with its double-negation.

The need for unions General consideration of symmetry should lead us to consider union types together with intersection types in our system. If a caller r can have type $A \cap B$, meaning that it denotes values that inhabit both A and B then it can interact with any callee that can receive an A-value *or* a B-value: such a callee will naturally be expected to have the type $A \cup B$. Thus far we have only argued that having intersection types for values suggests having union types for callees, which is in itself not a real extension of the intersection-types paradigm. But any type that can be the type of a caller-variable can be the type of a callee term (via the $\tilde{\mu}$-construction) and any type that can be the type of a callee-variable can be the type of a caller term (via the μ-construction). So we are committed to having intersections *and* unions for callers *and* callees.

A specific technical consideration gives another argument for having unions. If our goal is to characterize strongly normalizing terms then certainly we need to type all normal forms. As usual in traditional intersection types, when a term t has more than one occurrence of a caller-variable x, this variable will, in general, have a type of the form $A_1 \cap \cdots \cap A_n$, the different A_i reflecting various constraints on the type of x arising from its interaction with different subterms t. Similarly, if t has more than one occurrence of a callee-variable α, it will, in general, receive a type $A_1 \cup \cdots \cup A_n$, the type of continuations that can produce an answer given a value of any of the types $A_1, \ldots, A_n$

Definition 3.1 - *The set of* types *is generated from a set of* type variables *by closing under* $\cap$ *and* $\cup$*. We will always consider types to be defined modulo commutativity and associativity for* $\cap$ *and* $\cup$*.*

- *A* caller basis *is a set of statements of the form* $x : A$*, a* callee basis *is a set of statements of the form* $\alpha : A$*; in each case we stipulate that all variables are distinct.*

- *There are three types of* typing judgments*:*

$$\Gamma \vdash \boxed{r : A},\ \Delta \qquad \Gamma,\ \boxed{e : A} \vdash \Delta \qquad c : (\Gamma \vdash \Delta)$$

where Γ *is a caller basis and where* Δ *is a callee basis.*

□

3.1 Discussion

In order to motivate the technical condition on bases in Definition 3.2 below we show why it is that a naive attempt to define a type system with unions types leads to difficulties in Subject Reduction. More specifically, it seems difficult to prove a substitution lemma of the sort that is crucial to Subject Reduction.

A first attempt Consider a standard intersection type system for λ-calculus. At the level of the natural deduction, the rules for $\cap$ are just the rules for $\wedge$ (that is, if one erases the terms and looks just at the formulas). As we know, the difference between the formula $A \wedge B$ and the type $A \cap B$ is that in the latter we require the same term to witness A, B, and $A \cap B$.

We can imagine a type system for $\overline{\lambda}\mu\tilde{\mu}$ derived using the same principle, applied to sequent calculus deduction. So the $\cap$ rules would be based on the shape of the logic rules for $\wedge$:

$$\frac{\Gamma, A \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta} \qquad \frac{\Gamma \vdash \Delta, A \quad \Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \wedge B}$$

A key point is that, since introducing an intersection on the left or the right is *not* a logical inference, these type inferences are completely orthogonal to the notion of the stoup, i.e. of the active formula. This means that to write down the typing judgments corresponding to these rules in the context of $\overline{\lambda}\mu\tilde{\mu}$, we should write down several different $\overline{\lambda}\mu\tilde{\mu}$ sequents whose "erasure" looks like a given logic inference.

Of course the same principle holds for union types, whose rules are suggested by analogy with disjunction.

For example, consider

$$\frac{\Gamma, A \vdash \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta}$$

The formula $A \vee B$ can be the type of an active formula (i.e. the type of a callee), or it can be the type of a caller-variable, and in the latter case the judgment can be typing a caller or a callee. Thus, in terms of $\overline{\lambda}\mu\tilde{\mu}$, this one logic inference would yield three different typing rules, as follows.

$$\frac{\Gamma,\ \boxed{e : A} \vdash \Delta \qquad \Gamma,\ \boxed{e : B} \vdash \Delta}{\Gamma,\ \boxed{e : A \cup B} \vdash \Delta}\ (\cup_L)$$

$$\frac{\Gamma, x:A \vdash \boxed{r:T}, \Delta \qquad \Gamma, x:B \vdash \boxed{r:T}, \Delta}{\Gamma, x:A \cup B \vdash \boxed{r:T}, \Delta}\,(\cup_{L-Var})$$

$$\frac{\Gamma, x:A, \boxed{e:T} \vdash \Delta \qquad \Gamma, x:B \vdash e:T\Delta}{\Gamma, x:A \cup B, \boxed{e:T} \vdash \Delta}\,(\cup_{L-Var})$$

It turns out that the latter two rules above prove to be an obstacle to proving a substitution lemma of the kind that is necessary to support Subject Reduction. We require the following property:

$$\textit{If} \quad \Gamma, x:A \vdash \boxed{r:T}, \Delta \quad \textit{and} \quad \Gamma \vdash \boxed{s:A}, \Delta \quad \textit{then} \quad \Gamma \vdash \boxed{r[x \leftarrow s]:T}, \Delta$$

But knowing $\Gamma \vdash \boxed{s:A_1 \cup A_2}, \Delta$ doesn't allow us to use the induction hypothesis on the given $\Gamma, x:A_i \vdash \boxed{r:T}, \Delta$, and the proof cannot be completed.

A second attempt A first idea for fixing this problem might be to forbid union types for callers and forbid intersection types for callees. But as we observed earlier, in the presence of μ and $\tilde{\mu}$, any type that can be the type of a callee variable can arise as the type of a caller term, and any type that can be the type of a caller variable can arise as the type of a callee term. So we cannot hope to make any distinction between "caller types" "callee types."

A third and successful attempt However it turns out, as we will see, that we get a successful system if we simply forbid typing judgments whose bases contain variable-typings of the form $x:A \cup B$ and $\alpha:A \cap B$. In particular we reject the latter two rules above.

Definition 3.2
- *A type A is* $\cap$-definite *if it is a type variable, an arrow-type or it is $A_1 \cap A_2$, with each A_i a $\cap$-definite type. A type A is* $\cup$-definite *if it is a type variable, an arrow-type or it is $A_1 \cup A_2$, with each A_i a $\cup$-definite type.*
- *A basis Γ is* $\cap$-definite, *if in each binding $x:A$ in Γ, A is a $\cap$-definite type. A basis Δ is* $\cup$-definite *if in each binding $\alpha:A$ in Δ, A is a $\cup$-definite type.*
- *A typing judgment is* definite *if its typing bases Γ and Δ are $\cap$- and $\cup$-definite, respectively.*

□

Note that in a definite typing judgment we do not insist that the type of the active formula be definite. We will restrict attention to definite typing judgments in the type system we define.

A further simplification of the system we define will be for convenience only. The analogy with the logical rules described above suggest the following rules for variables:

$$\frac{\Gamma, x:A \vdash \Delta}{\Gamma, x:A \cap B \vdash \Delta} \qquad \frac{\Gamma \vdash \Delta, \alpha:A}{\Gamma \vdash \Delta, \alpha:A \cup B}$$

These rules are perfectly sound, but it turns out that their presence complicates reasoning about this system. On the other hand it is not hard to see that application of these

rules can always be pushed towards the leaves of a typing tree. In fact an equivalent formulation of the system removes these rules completely and replaces them with more flexible axiom schemas.

The system in Definition 3.3 is the system obtained from the rules naively generated by analogy with the logical inference rules by

- restricting to definite type judgments and
- replacing the variable rules above by the more flexible axioms e^+-*ax* and r^+-*ax*.

This system fits well with the sequent calculus based on active formulas, since all the rules concern the terms associated with the active formula, as opposed to the earlier two systems, whose var-rules changed the type basis for an active term while keeping the type of the term the same.

Note that although we have forbidden caller variables to have union types, we **do** have caller terms with union types, due to their typing rule μ.

3.2 The types system $\mathcal{M}^{\cap\cup}$

Definition 3.3 (The type system $\mathcal{M}^{\cap\cup}$) *The typing judgements of $\mathcal{M}^{\cap\cup}$ are those derivable by the rules in Figure 1. In each rule we assume that the bases are definite.*
□

The first property we will need is that the intersection and union rules with two premises can be "inverted" in the sense that if the judgment in the conclusion of the rule is derivable then each of the judgments in the hypotheses are derivable.

It is precisely here that we reap the benefit of our restriction to definite bases. The following lemma is false without this restriction. Indeed

$$x : A \cap A \to (B \cap C) \;\vdash\; \boxed{\mu\alpha.\langle x \parallel x \bullet \alpha\rangle : B \cap C},$$

since

$$\langle x \parallel x \bullet \alpha\rangle : \quad (x : A \cap A \to (B \cap C) \quad \vdash \quad \alpha : B \cap C)$$

where $\alpha : B \cap C$ **is not** $\cup$-definite. An easy check shows that one has

$$\begin{aligned} \text{neither} \quad & x : A \cap A \to (B \cap C) \;\vdash\; \boxed{\mu\alpha.\langle x \parallel x \bullet \alpha\rangle : B}, \\ \text{nor} \quad & x : A \cap A \to (B \cap C) \;\vdash\; \boxed{\mu\alpha.\langle x \parallel x \bullet \alpha\rangle : C}, \ . \end{aligned}$$

Lemma 3.4 (Elimination)

i. If $\Gamma \vdash \boxed{r : A_1 \cap A_2}, \Delta$ *then for* $i = 1, 2$, $\Gamma \vdash \boxed{r : A_i}, \Delta$.

ii. If $\Gamma, \boxed{e : A_1 \cup A_2} \vdash \Delta$ *then for* $i = 1, 2$, $\Gamma, \boxed{e : A_i} \vdash \Delta$.

Proof. For part 1, we just observe that the only rules that could be used to derive $\Gamma \vdash \boxed{r : A_1 \cap A_2}, \Delta$ are r^+-*ax* and $\cap_R$. In the latter case the result is immediate; and in the former case the result is a consequence of the fact that we are considering types modulo associativity and commutativity of $\cap$.

The fact that the last inference cannot be a μ is a direct consequence of our assumption that bases are definite. That is, since bases cannot have assumptions of the form $\alpha : A_1 \cap A_2$ a derivation such as the one below is *not possible*:

$$\dfrac{\dfrac{\Gamma \vdash \boxed{r : T},\ \Delta, \alpha : A_1 \cap A_2 \quad \Gamma,\ \boxed{e : T} \vdash \Delta, \alpha : A_1 \cap A_2}{\langle r \parallel e \rangle \ :\ (\Gamma \vdash \Delta, \alpha : A_1 \cap A_2)}\ (cut)}{\Gamma \vdash \boxed{\mu\alpha.\langle r \parallel e \rangle : A_1 \cap A_2},\ \Delta}$$

the assumed derivation must look like

$$\dfrac{\Gamma \vdash \boxed{\mu\alpha.\langle r \parallel e \rangle : A_1},\ \Delta \quad \Gamma \vdash \boxed{\mu\alpha.\langle r \parallel e \rangle : A_2},\ \Delta}{\Gamma \vdash \boxed{\mu\alpha.\langle r \parallel e \rangle : A_1 \cap A_2},\ \Delta}$$

that is, an instance of $\cap_R$. This completes the proof of part 1 of the lemma. The proof of part 2 is similar. □

Definition 3.5 *If Γ_1 and Γ_2 are caller bases, define $\Gamma_1 \sqcap \Gamma_2$ to be*

$$\begin{aligned}\Gamma_1 \sqcap \Gamma_2 = \{x : A \mid (x : A) \in \Gamma_1 \text{ and } x \notin \Gamma_2\} \quad &\cup \\ \{x : A \mid (x : A) \in \Gamma_2 \text{ and } x \notin \Gamma_2\} \quad &\cup \\ \{x : A \cap B \mid (x : A) \in \Gamma_1 \text{ and } (x : B) \in \Gamma_2\}&\end{aligned}$$

If Δ_1 and Δ_2 are callee bases, define $\Delta_1 \sqcup \Delta_2$ to be

$$\begin{aligned}\Delta_1 \sqcup \Delta_2 = \{x : A \mid (x : A) \in \Delta_1 \text{ and } x \notin \Delta_2\} \quad &\cup \\ \{x : A \mid (x : A) \in \Delta_2 \text{ and } x \notin \Delta_1\} \quad &\cup \\ \{x : A \cup B \mid (x : A) \in \Delta_1 \text{ and } (x : B) \in \Delta_2\}&\end{aligned}$$

□

Lemma 3.6 *Let Γ and Γ' be $\cap$-definite; let Δ and Δ' be $\cup$-definite. Then $\Gamma_1 \sqcap \Gamma_2$ is $\cap$-definite and $\Delta_1 \sqcup \Delta_2$ is $\cup$-definite. Furthermore:*

i. If $\Gamma \vdash \boxed{r : A},\ \Delta$ *then* $\Gamma \sqcap \Gamma' \vdash \boxed{r : A},\ \Delta \sqcup \Delta'$.

ii. If $\Gamma,\ \boxed{e : A} \vdash \Delta$ *then* $\Gamma \sqcap \Gamma' \vdash \boxed{e : A},\ \Delta \sqcup \Delta'$.

Lemma 3.7 (Context expansion lemma) *Let $\Gamma \subseteq \Gamma'$ and $\Delta \subseteq \Delta'$.*

i. If $\Gamma \vdash \boxed{r : A}, \Delta$, then $\Gamma' \vdash \boxed{r : A}, \Delta'$.

ii. If $\Gamma, \boxed{e : A} \vdash \Delta$, then $\Gamma', \boxed{e : A} \vdash \Delta'$.

iii. If $c : (\Gamma \vdash \Delta)$, then $c : (\Gamma' \vdash \Delta')$.

Lemma 3.8 (Generation lemma)

i. If $\Gamma \vdash \boxed{\lambda x.r : \bigcap_{i \in I} A_i \to B_i}, \Delta$, then $\Gamma, x : A_i \vdash \boxed{r : B_i}, \Delta$.

ii. If $\Gamma, \boxed{r \bullet e : \bigcup_{i \in I} A_i \to B_i} \vdash \Delta$, then $\Gamma \vdash \boxed{r : A_i}, \Delta$ and $\Gamma, \boxed{e : B_i} \vdash \Delta$.

iii. If $\Gamma \vdash \boxed{\mu\alpha.c : \bigcap_{i \in I} A_i}, \Delta$, then $c : (\Gamma \vdash \alpha : A_i, \Delta)$.

iv. If $\Gamma, \boxed{\widetilde{\mu}x.c : \bigcup_{i \in I} A_i} \vdash \Delta$, then $c : (\Gamma, x : A_i \vdash \Delta)$.

4 Subject Reduction

In this section we show that our type system enjoys the Subject Reduction property, for the the calculus with unrestricted reduction, that is, even in the presence of the $(\mu, \tilde{\mu})$ critical pair. As mentioned in the introduction and shown in [3] subject reduction is difficult to achieve in a system with union types.

As usual the key property to verify to ensure Subject Reduction is that substitution behaves well. So the next lemma is the heart of the argument.

Lemma 4.1 (Substitution) *Let us suppose that all judgments are definite.*

1. *If* $\Gamma, x:S \vdash \boxed{t:T}, \Delta$ *and* $\Gamma \vdash \boxed{s:S}, \Delta$ *then* $\Gamma \vdash \boxed{t[x \leftarrow s]:T}, \Delta$.
2. *If* $\Gamma \vdash \boxed{t:T}, \Delta, \alpha:S$ *and* $\Gamma, \boxed{f:S} \vdash \Delta$ *then* $\Gamma \vdash \boxed{t[\alpha \leftarrow f]:T}, \Delta$.
3. *If* $\Gamma, x:S, \boxed{g:T} \vdash \Delta$ *and* $\Gamma \vdash \boxed{s:S}, \Delta$ *then* $\Gamma, \boxed{g[x \leftarrow s]:T} \vdash \Delta$.
4. *If* $\Gamma, \boxed{g:T} \vdash \Delta, \alpha:S$ *and* $\Gamma, \boxed{f:S} \vdash \Delta$ *then* $\Gamma, \boxed{g[\alpha \leftarrow f]:T} \vdash \Delta$.

Proof. We prove the parts simultaneously by induction on the length of the derivation of the first indicated judgment in each part. We can remark that Lemma 3.4 — and so in turn the assumption that all bases are definite — is crucial to the argument.

We organize the proof by considering the case of each typing rule in turn.

Case: r^+**-***ax***.** We must (only) address parts 1 and 2. Part 2 is trivial. For part 1, we are given

$$\frac{}{\Gamma, x:A_1 \cap \cdots \cap A_n \vdash \boxed{x:A_i}, \Delta}\ (r^+\text{-}ax)$$

and $\Gamma \vdash \boxed{s:A_1 \cap \cdots \cap A_n}, \Delta$; we want to show that $\Gamma \vdash \boxed{s:A_i}, \Delta$. This is immediate from Lemma 3.4.1.

Case: e^+**-***ax***.** We must address parts 3 and 4. Part 3 is trivial. For part 4, the argument is similar to the previous case, using Lemma 3.4.2.

Case: $\rightarrow L$ We consider parts 3 and 4. For part 3 we are given

$$\frac{\Gamma, x:S \vdash \boxed{r:A}, \Delta \quad \Gamma, x:S, \boxed{e:B} \vdash \Delta}{\Gamma, x:S, \boxed{r \bullet e : A \rightarrow B} \vdash \Delta}\ (\rightarrow L)$$

and

$$\Gamma \vdash \boxed{s:S}, \Delta$$

and we note that

$$(r \bullet e)[x \leftarrow s] = (r[x \leftarrow s]) \bullet (e[x \leftarrow s])$$

so the result follows by induction hypothesis, parts 1 and 3.

For part 4 we are given

$$\frac{\Gamma \vdash \boxed{r : A},\ \Delta, \alpha : S \quad \Gamma,\ \boxed{e : B} \vdash \Delta, \alpha : S}{\Gamma,\ \boxed{r \bullet e : A \to B} \vdash \Delta, \alpha : S}\ (\to L)$$

and

$$\Gamma,\ \boxed{f : S} \vdash \Delta.$$

and we note that

$$(r \bullet e)[\alpha \leftarrow f] = (r[\alpha \leftarrow f]) \bullet (e[\alpha \leftarrow f])$$

so the result follows by induction hypothesis, parts 2 and 4.
Case: $\to R$**.** We address parts 1 and 2. For part 1 we are given

$$\frac{\Gamma, x : S, y : A \vdash \boxed{r : B},\ \Delta}{\Gamma, x : S \vdash \boxed{\lambda y.r : A \to B},\ \Delta}\ (\to R)$$

and

$$\Gamma \vdash \boxed{s : S},\ \Delta$$

and we note that

$$(\lambda y.r)[x \leftarrow s] = \lambda y.(r[x \leftarrow s])$$

so the result follows by induction hypothesis, part 1, and an application of rule $\to R$.

For part 2 we are given

$$\frac{\Gamma, y : A \vdash \boxed{r : B},\ \Delta, \alpha : S}{\Gamma \vdash \boxed{\lambda y.r : A \to B},\ \Delta, \alpha : S}\ (\to R)$$

and

$$\Gamma,\ \boxed{f : S} \vdash \Delta.$$

and we note that

$$(\lambda y.r)[\alpha \leftarrow f] = \lambda y.(r[\alpha \leftarrow f])$$

so the result follows by induction hypothesis, part 2, and an application of rule $\to R$.
Case: $\widetilde{\mu}$**.** We address parts 3 and 4. For part 3 we are given

$$\frac{\dfrac{\Gamma, y : T, x : S \vdash \boxed{r : B},\ \Delta \qquad \Gamma, y : T, x : S,\ \boxed{e : B} \vdash \Delta}{\langle r \parallel e \rangle\ :\ (\Gamma, y : T, x : S \vdash \Delta)}}{\Gamma, x : S,\ \boxed{\widetilde{\mu} y.\langle r \parallel e \rangle : T} \vdash \Delta}\ (\widetilde{\mu})$$

and

$$\Gamma \vdash \boxed{s : S},\ \Delta$$

and we seek to show that

$$\Gamma,\ \boxed{(\widetilde{\mu} y.\langle r \parallel e \rangle)[x \leftarrow s] : T} \vdash \Delta$$

Since $(\widetilde{\mu} y.\langle r \parallel e \rangle)[x \leftarrow s] = \widetilde{\mu} y.\langle r[x \leftarrow s] \parallel e[x \leftarrow s] \rangle$, the result follows from the induction hypothesis, parts 1 and 3, applied to the leaves of the derivation shown, followed by (cut) and $(\widetilde{\mu})$.

Part 4 is similar, using the induction hypothesis, parts 2 and 4.

Case: μ**.** We address parts 1 and 2. For part 1 we are given

$$\frac{\Gamma, x:S \;\vdash\; \boxed{r:B},\ \alpha:T,\Delta \qquad \Gamma, x:S,\ \boxed{e:B} \;\vdash\; \alpha:T,\Delta}{\dfrac{\langle r \parallel e\rangle \;:\; (\Gamma, x:S \;\vdash\; \alpha:T,\Delta)}{\Gamma, x:S \;\vdash\; \boxed{\mu\alpha.\langle r \parallel e\rangle : T},\ \Delta}\,(\mu)}$$

and

$$\Gamma \;\vdash\; \boxed{s:S},\ \Delta$$

and we seek to show that

$$\Gamma, x:S \;\vdash\; \boxed{(\mu\alpha.\langle r \parallel e\rangle)[x \leftarrow s] : T},\ \Delta$$

Since $(\mu\alpha.\langle r \parallel e\rangle)[x \leftarrow s] = \mu\alpha.\langle r[x \leftarrow s] \parallel e[x \leftarrow s]\rangle$, the result follows from the induction hypothesis, parts 1 and 3, applied to the leaves of the derivation shown, followed by (cut) and (μ).

Part 2 is similar, using the induction hypothesis, parts 2 and 4.

Case: $\cap_L$**.** We address parts 3 and 4. For part 3 we are given

$$\frac{\Gamma, x:S,\ \boxed{e:A} \;\vdash\; \Delta}{\Gamma, x:S,\ \boxed{e:A \cap B} \;\vdash\; \Delta}\,(\cap_L)$$

and we wish to show that

$$\Gamma,\ \boxed{e[x \leftarrow s] : A \cap B} \;\vdash\; \Delta$$

This is an easy application of the induction hypothesis, part 3.

Part 4 is similar.

Case: $\cap_R$**.** We address parts 1 and 2. For part 1 we are given

$$\frac{\Gamma, x:S \;\vdash\; \boxed{r:A},\ \Delta \quad \Gamma, x:S \;\vdash\; \boxed{r:B},\ \Delta}{\Gamma, x:S \;\vdash\; \boxed{r:A \cap B},\ \Delta}\,(\cap_R)$$

and we wish to show that

$$\Gamma \;\vdash\; \boxed{r[x \leftarrow s] : A \cap B},\ \Delta$$

This is an easy application of the induction hypothesis, part 1.

Part 2 is similar.

Case: $\cup_L$**.** Similar to case $\cap_R$.

Case: $\cup_R$**.** Similar to case $\cap_L$. □

Theorem 4.2 (Subject Reduction)
If $c : (\Gamma \vdash \Delta)$ *and* $c \to c'$ *then* $c' : (\Gamma \vdash \Delta)$

Proof. There are three cases, corresponding to the reductions for μ, $\tilde{\mu}$, and λ.

Case: μ**.** We have

$$\langle \mu\alpha.\langle r \parallel e\rangle \parallel f\rangle : (\Gamma \vdash \Delta)$$

and we wish to show

$$\langle r[\alpha \leftarrow f] \parallel e[\alpha \leftarrow f]\rangle : (\Gamma \vdash \Delta)$$

By hypothesis, for some type T,

$$\Gamma \vdash \boxed{r : T},\ \Delta \quad \text{and} \quad \Gamma,\ \boxed{e : T} \vdash \Delta.$$

But by the Substitution Lemma 4.1,

$$\Gamma \vdash \boxed{r[\alpha \leftarrow f] : T},\ \Delta \quad \text{and} \quad \Gamma,\ \boxed{e[\alpha \leftarrow f] : T} \vdash \Delta$$

and so the result follows.

Case: $\tilde{\mu}$**.** Similar to case μ.

Case: λ**.** We have $\langle \lambda x.r \parallel s \bullet e\rangle : (\Gamma \vdash \Delta)$ and we wish to show $\langle s \parallel \tilde{\mu}x.\langle r \parallel e\rangle\rangle : (\Gamma \vdash \Delta)$. By hypothesis, for some type T,

$$\Gamma \vdash \boxed{\lambda x.r : T},\ \Delta \quad \text{and} \quad \Gamma,\ \boxed{s \bullet e : T} \vdash \Delta.$$

We perform a sub-induction on the size of the type T. The type T is of one of the forms: $A \to B$, $T_1 \cap T_2$, or $T_1 \cup T_2$.

case: T *is* $A \to B$*.* Then $\lambda x.r$ must have been typed by an application of rule $\to R$,

$$\frac{\Gamma, x : A \vdash \boxed{r : B},\ \Delta}{\Gamma \vdash \boxed{\lambda x.r : A \to B},\ \Delta}\ (\to R)$$

and $s \bullet e$ must have been typed by an application of rule $\to L$:

$$\frac{\Gamma \vdash \boxed{s : A},\ \Delta \quad \Gamma,\ \boxed{e : B} \vdash \Delta}{\Gamma,\ \boxed{s \bullet e : A \to B} \vdash \Delta}\ (\to L)$$

We can rearrange these typing judgments to obtain the desired typing.

$$\frac{\Gamma \vdash \boxed{s : A},\ \Delta \qquad \dfrac{\dfrac{\Gamma, x : A \vdash \boxed{r : B},\ \Delta \qquad \Gamma,\ \boxed{e : B} \vdash \Delta}{\langle r \parallel e\rangle : (\Gamma, x : A \vdash \Delta)}}{\Gamma,\ \boxed{\tilde{\mu}x.\langle r \parallel e\rangle : A} \vdash \Delta}}{\langle s \parallel \tilde{\mu}x.\langle r \parallel e\rangle\rangle : (\Gamma \vdash \Delta)}$$

case: T *is* $T_1 \cap T_2$*.* Then $s \bullet e$ must have been typed by an application of rule $\cap_L$, so that

$$\frac{\Gamma,\ \boxed{s \bullet e : T_i} \vdash \Delta}{\Gamma,\ \boxed{s \bullet e : T_1 \cap T_2} \vdash \Delta}\ (\cap_L)$$

for either $i = 1$ or $i = 2$ in $\cap_L$. But by Lemma 3.4, we have $\Gamma \vdash \boxed{\lambda x.r : T_i},\ \Delta$ and so the result follows by application of the induction hypothesis at type T_i.

case: T is $T_1 \cup T_2$. Then $\lambda x.r$ must have been typed by an application of rule $\cup_R$

$$\frac{\Gamma \vdash \boxed{\lambda x.r : T_i}, \Delta}{\Gamma \vdash \boxed{\lambda x.r : T_1 \cup T_2}, \Delta} (\cup_R)$$

for either $i = 1$ or $i = 2$ in $\cup_R$. But by Lemma 3.4, we have $\Gamma, \boxed{s \bullet e : T_i} \vdash \Delta$ and so the result follows by application of the induction hypothesis at type T_i. □

5 Strongly normalizing terms are typable

In this section we show that our type system assigns a type to any normal form and that a Subject Expansion result holds, so that any term that is strongly normalizing (under any of the reduction disciplines considered) is typable.

5.1 Typing normal forms

Theorem 5.1 *Every normal form is typable.*

Proof. It is here that the unions first play a crucial role, in assigning types to callee variables that occur more than once in a normal form. We prove by induction on terms that

i. If r is a caller normal form then there are Γ, Δ, and T such that $\Gamma \vdash \boxed{r : T}, \Delta$

ii. If e is a callee normal form then there are Γ, Δ and T such that $\Gamma, \boxed{e : T} \vdash \Delta$

iii. If c is a capsule normal form then there are Γ and Δ such that $c : (\Gamma \vdash \Delta)$

Of course in each case we assert that Γ comprises only $\cap$-definite bindings and that Δ comprises only $\cup$-definite bindings.

Now suppose r is a caller normal form. Of course if r is a variable we may type r with a type variable. If r is $\lambda x.s$ then by induction we have

$$\Gamma \vdash \boxed{s : T}, \Delta$$

where, without loss of generality we have assumed a binding $x : X$ in the typing for s. Then we have

$$\Gamma \setminus x : X \vdash \boxed{\lambda x.s : X \to T}, \Delta$$

If r is $\mu\alpha.\langle r \parallel e \rangle$ then by induction we have

$$\langle r \parallel e \rangle : (\Gamma \vdash \alpha : X, \Delta)$$

where, without loss of generality we have assumed a binding for α. Then

$$\Gamma \vdash \boxed{\mu\alpha.\langle r \parallel e \rangle : X}, \Delta$$

This completes the argument for typing caller normal forms.

A callee normal form is either a variable, or is of the form $\widetilde{\mu}x.c$ or $r \bullet e$. The variable case is easy, and the $\widetilde{\mu}$ case is similar to that for μ. When the term is $r \bullet e$ we have by induction

$$\Gamma_1 \ \vdash \ \boxed{r : A}, \ \Delta_1 \quad \text{and} \quad \Gamma_2, \ \boxed{e : B} \ \vdash \ \Delta_2$$

Then

$$\Gamma_1 \sqcap \Gamma_2, \ \boxed{r \bullet e : A \to B} \ \vdash \ \Delta_1 \sqcup \Delta_2$$

A capsule normal form is of one of the forms

$$\langle \lambda x.r \parallel \alpha \rangle \quad \text{or} \quad \langle x \parallel (r_1 \bullet \ldots \bullet r_n \bullet e) \rangle$$

where $n \geq 0$ and e is either α or $\widetilde{\mu}x.c$

Consider the first case. We have, for some Γ, Δ, and T,

$$\Gamma \ \vdash \ \boxed{\lambda x.r : T}, \ \Delta$$

If α does not occur free in $\lambda x.r$ then we may suppose that $\alpha \notin \Delta$. Then

$$\Gamma \ \vdash \ \boxed{\lambda x.r : T}, \ \Delta, \alpha : T \quad \text{and} \quad \Gamma, \ \boxed{\alpha : T} \ \vdash \ \Delta, \alpha : T$$

so that

$$\langle \lambda x.r \parallel \alpha \rangle : (\Gamma \ \vdash \ \Delta, \alpha : T)$$

If α does occur free in $\lambda x.r$ then we will have, for some A,

$$\Gamma \ \vdash \ \boxed{\lambda x.r : T}, \ \Delta', \alpha : A.$$

Then

$$\Gamma \ \vdash \ \boxed{\lambda x.r : T}, \ \Delta', \alpha : T \cup A.$$

and

$$\Gamma, \ \boxed{\alpha : T} \ \vdash \ \Delta', \alpha : T \cup A$$

so that

$$\langle \lambda x.r \parallel \alpha \rangle : (\Gamma \ \vdash \ \Delta', \alpha : T \cup A)$$

For the case $\langle x \parallel (r \bullet e) \rangle$, we have a typing

$$\Gamma, \ \boxed{r \bullet e : T} \ \vdash \ \Delta$$

If x is not free in $(r \bullet e)$ then

$$\Gamma, x : T \ \vdash \ \boxed{x : T}, \ \Delta$$

and

$$\Gamma, x : T, \ \boxed{r \bullet e : T} \ \vdash \ \Delta$$

so that

$$\langle x \parallel (r \bullet e) \rangle : (\Gamma, x : T \ \vdash \ \Delta)$$

Otherwise, if x gets a type A in Γ

$$\Gamma', x : T \cap A \ \vdash \ \boxed{x : T}, \ \Delta$$

and

$$\Gamma', x : T \cap A, \ \boxed{r \bullet e : T} \ \vdash \ \Delta$$

so that

$$\langle x \parallel (r \bullet e) \rangle : (\Gamma', x : T \cap A \ \vdash \ \Delta)$$

□

5.2 Subject expansion

The following is the main lemma supporting the Subject Expansion theorem; it can be viewed as a converse to the Substitution Lemma.

Lemma 5.2 (Subject Expansion)

1. *Suppose* $\Gamma \vdash \boxed{r[x \leftarrow s] : B},\ \Delta$ *and suppose that caller* s *is typable under* Γ, Δ. *Then there is a type* A *such that*

$$\Gamma \vdash \boxed{s : A},\ \Delta \quad \textit{and} \quad \Gamma, x : A \vdash \boxed{r : B},\ \Delta.$$

2. *Suppose* $\Gamma \vdash \boxed{r[\alpha \leftarrow f] : B},\ \Delta$ *and suppose that callee* f *is typable under* Γ, Δ. *Then there is a type* A *such that*

$$\Gamma,\ \boxed{f : A} \vdash \Delta \quad \textit{and} \quad \Gamma \vdash \boxed{r : B},\ \Delta, \alpha : A.$$

3. *Suppose* $\Gamma,\ \boxed{e[x \leftarrow s] : B} \vdash \Delta$ *and suppose that caller* s *is typable under* Γ, Δ. *Then there is a type* A *such that*

$$\Gamma \vdash \boxed{s : A},\ \Delta \quad \textit{and} \quad \Gamma, x : A,\ \boxed{e : B} \vdash \Delta.$$

4. *Suppose* $\Gamma,\ \boxed{e[\alpha \leftarrow f] : B} \vdash \Delta$ *and suppose that callee* f *is typable under* Γ, Δ. *Then there is a type* A *such that*

$$\Gamma,\ \boxed{f : A} \vdash \Delta \quad \textit{and} \quad \Gamma,\ \boxed{e : B} \vdash \Delta, \alpha : A.$$

5. *Suppose* $c[x \leftarrow s] : (\Gamma \vdash \Delta)$ *and suppose that caller* s *is typable under* Γ, Δ. *Then there is a type* B *such that*

$$\Gamma \vdash \boxed{s : B},\ \Delta \quad \textit{and} \quad c : (\Gamma, x : B \vdash \Delta).$$

6. *Suppose* $c[\alpha \leftarrow f] : (\Gamma \vdash \Delta)$ *and suppose that callee* f *is typable under* Γ, Δ. *Then there is a type* B *such that*

$$\Gamma,\ \boxed{f : B} \vdash \Delta \quad \textit{and} \quad c : (\Gamma \vdash \Delta, \alpha : B).$$

Proof. By induction on r, e and c.

1. – If $r \equiv x$, then $r[x \leftarrow s] \equiv s$ and

$$\Gamma \vdash \boxed{s : B},\ \Delta$$

according to the assumption. On the other hand

$$\Gamma, x : B \vdash \boxed{x : B},\ \Delta$$

by $r^+ - ax$.
If $r \equiv y$, then $r[x \leftarrow s] \equiv y$,

$$\Gamma \vdash \boxed{y : B},\ \Delta$$

and for some type A,

$$\Gamma \ \vdash \ \boxed{s : A}, \ \Delta$$

according to the assumption. Then

$$\Gamma, x : A \ \vdash \ \boxed{y : B}, \ \Delta$$

by Context expansion lemma 3.7.

– Let $r \equiv \lambda y.r'$. Then $(\lambda y.r')[x \leftarrow s] \equiv \lambda y.r'[x \leftarrow s]$, where $y \notin \mathtt{Fv_r}(s)$ by Barendregt convention. By assumption

$$\Gamma \ \vdash \ \boxed{\lambda y.r'[x \leftarrow s] : B}, \ \Delta$$

We perform a sub-induction on the size of type B. The type B is one of the forms: $T_1 \to T_2$, $T_1 \cap T_2$, or $T_1 \cup T_2$.
case B is $T_1 \to T_2$. By Generation lemma 3.8

$$\Gamma, y : T_1 \ \vdash \ \boxed{r'[x \leftarrow s] : T_2}, \ \Delta$$

and by the induction hypothesis

$$\Gamma, y : T_1 \ \vdash \ \boxed{s : T_2}, \ \Delta$$

(which yields $\Gamma \ \vdash \ \boxed{s : B}, \ \Delta$ since $y \notin \mathtt{Fv_r}(s)$) and

$$\Gamma, y : T_1, x : B \ \vdash \ \boxed{r' : T_2}, \ \Delta$$

(which yields $\Gamma, x : B \ \vdash \ \boxed{\lambda y.r' : T_1 \to T_2}, \ \Delta$).
case B is $T_1 \cap T_2$. Then by Elimination lemma 3.4

$$\Gamma \ \vdash \ \boxed{\lambda y.r'[x \leftarrow s] : T_i}, \ \Delta$$

and by the induction hypothesis there is a type A_i such the

$$\Gamma \ \vdash \ \boxed{s : A_i}, \ \Delta \quad \text{and} \quad \Gamma, x : A_i \ \vdash \ \boxed{\lambda y.r' : T_i}, \ \Delta$$

for $i = 1, 2$. The result follows by Lemma 3.6 and $\cap_R$.
case B is $T_1 \cup T_2$. The last rule applied must have been $\cup_R$ hence

$$\Gamma \ \vdash \ \boxed{\lambda y.r'[x \leftarrow s] : T_i}, \ \Delta$$

for either $i = 1$ or $i = 2$. Take $i = 1$, then by the induction hypothesis

$$\Gamma \ \vdash \ \boxed{s : A_1}, \ \Delta \quad \text{and} \quad \Gamma, x : A_1 \ \vdash \ \boxed{\lambda y.r' : T_1}, \ \Delta$$

The result follows by $\cup_R$.

– Let $r \equiv \mu\alpha.c$. Then $(\mu\alpha.c)[x \leftarrow s] \equiv \mu\alpha.c[x \leftarrow s]$, where $\alpha \notin \mathtt{Fv_e}(s)$. Let

$$\Gamma \ \vdash \ \boxed{\mu\alpha.(c[x \leftarrow s]) : B}, \ \Delta$$

According to Generation lemma 3.8

$$c[x \leftarrow s] : (\Gamma \vdash \alpha : B, \Delta)$$

hence by the induction hypothesis

$$\Gamma \ \vdash \ \boxed{s : A}, \ \Delta \quad \text{and} \quad c : (\Gamma, x : A \vdash \alpha : B, \Delta)$$

Thus

$$\Gamma, x : A \ \vdash \ \boxed{\mu\alpha.c : B}, \ \Delta$$

2. 3. and 4. Similar.

5. In the case of a capsule $\langle r \parallel e \rangle[x \leftarrow s] \equiv \langle r[x \leftarrow s] \parallel e[x \leftarrow s] \rangle$, hence the last typing rule applied is (cut), therefore

$$\Gamma \ \vdash \ \boxed{r[x \leftarrow s] : B}, \ \Delta \quad \text{and} \quad \Gamma, \ \boxed{e[x \leftarrow s] : B} \ \vdash \ \Delta$$

By the induction hypothesis there exists A_1 such that

$$\Gamma \ \vdash \ \boxed{s : A_1}, \ \Delta \quad \text{and} \quad \Gamma, x : A_1 \ \vdash \ \boxed{r : B}, \ \Delta$$

and there exists A_2 such that

$$\Gamma \ \vdash \ \boxed{s : A_2}, \ \Delta \quad \text{and} \quad \Gamma, x : A_2, \ \boxed{e : B} \ \vdash \ \Delta$$

By $(\cap R)$ we get

$$\Gamma \ \vdash \ \boxed{s : A_1 \cap A_2}, \ \Delta$$

By Lemma 3.6 we get

$$\Gamma, x : A_1 \cap A_2 \ \vdash \ \boxed{r : B}, \ \Delta \quad \text{and} \quad \Gamma, x : A_1 \cap A_2, \ \boxed{e : B} \ \vdash \ \Delta$$

Using (cut) we get

$$\langle r \parallel e \rangle : (\Gamma, x : A_1 \cap A_2 \vdash \Delta)$$

Part 6. is similar.

□

Theorem 5.3 *All strongly normalizing terms are typable.*

Proof. Let c be strongly normalizing. We proceed by induction on the length of the longest reduction of c. So suppose c and c' are capsules with $c \longrightarrow c'$. From the assumption that c' is typeable we derive a typing for c; there are three cases according to the rule used to reduce c. Here we present only one, the case of the λ rule: $\langle \lambda x.r \parallel r' \bullet e \rangle \longrightarrow \langle r' \parallel \widetilde{\mu} x.\langle r \parallel e \rangle \rangle$. By the induction hypothesis $\langle r' \parallel \widetilde{\mu} x.\langle r \parallel e \rangle \rangle : (\Gamma \vdash \Delta)$. In this typing (cut) must have been the last rule applied.

Therefore for some type A

$$\Gamma \vdash \boxed{r' : A}, \Delta \quad \text{and} \, \Gamma, \boxed{\widetilde{\mu} x.\langle r \parallel e \rangle : A} \vdash \Delta.$$

The second implies $\langle r \parallel e \rangle : (\Gamma, x : A \vdash \Delta)$. So, there is a type B such that

$$\Gamma, x : A, \boxed{e : B} \vdash \Delta \quad \text{and} \quad \Gamma, x : A \vdash \boxed{r : B}, \Delta.$$

By the variable convention x is not free in e so $\Gamma, \boxed{e : A} \vdash \Delta$. Putting this together we get to $\langle \lambda x.r \parallel r' \bullet e \rangle : (\Gamma \vdash \Delta)$.

□

6 Typable terms are strongly normalizing

Curien and Herbelin [11], and Lengrand [18], have shown how to deduce strong normalization for simply-typed $\overline{\lambda}\mu\widetilde{\mu}$ in both the call-by-name and call-by-value disciplines; the former by embedding $\overline{\lambda}\mu\widetilde{\mu}$ into the λ-calculus with continuation-passing style translations and the latter via interpretation into the calculus of Urban and Bierman [28]. But it is not clear how to adapt these arguments to obtain strong normalization for an intersection-types system. In this section we present a self-contained SN proof for $\mathcal{M}^{\cap\cup}$-typable $\overline{\lambda}\mu\widetilde{\mu}$ terms by a variant of the standard reducibility method for the λ-calculus. In fact we give a uniform proof that applies to both the call-by-name and call-by-value regimes, and indeed generalizes to even more flexible systems.

The call-by-name subsystem of $\overline{\lambda}\mu\widetilde{\mu}$ is the system defined by the constraint that says that $\widetilde{\mu}$-redexes have "higher priority" in the critical pair, the call-by-value subsystem gives priority to μ-redexes. Let us generalize these conventions.

Definition 6.1 *A* priority π *is a well-founded partial ordering defined on terms of the form* $\mu\alpha.c$ *and* $\widetilde{\mu}x.c$

□

For example, the *call-by-name priority* is defined by $\mu\alpha.c \prec \widetilde{\mu}x.c'$ for all c and c', with no other relations, and the *call-by-value priority* is defined by $\widetilde{\mu}x.c' \prec \mu\alpha.c$ for all c and c', with no other relations. The *longest capsule first priority* defined by $\widetilde{\mu}x.c' \prec \mu\alpha.c$ if $\|c'\| < \|c\|$ (where is $\|c\|$ is the size of c in term of number of symbols) and $\mu\alpha.c \prec \widetilde{\mu}x.c'$ otherwise is another priority.

In order to make precise the sense in which a priority defines a notion of reduction, we require a few definitions.

Definition 6.2 *Let* π *be a priority.*

- *If r is a caller, let us say that a callee e is* available *to r if either e is not of the form $\tilde{\mu}x.c'$ or e precedes r in the priority π. Similarly, for a callee e we say that a caller r is* available *to e if either r is not of the form $\mu\alpha.c'$ or r precedes e in the priority π.*
- *A reduction step* respects π *if either its redex is not an instance of a critical pair, or, letting the redex be $\langle \mu\alpha.c \parallel \tilde{\mu}x.c' \rangle$, the $\tilde{\mu}$-reduction is done if $\mu\alpha.c$ precedes $\tilde{\mu}x.c'$, but the μ-reduction is done if $\tilde{\mu}x.c'$ precedes $\mu\alpha.c$.*
- *A reduction sequence is a π-reduction if each step respects π.*
- *A term t is* strongly normalizing under π, *or is a SN^π-*term, *if every π-reduction sequence out of t terminates.*

□

So in the terminology above, the call-by-name subsystem of $\overline{\lambda}\mu\tilde{\mu}$ is the system in which each reduction respects the call-by-name priority, while the call-by-value subsystem respects the call-by-value priority.

But note that there are many priorities other than these two. For example, one can define a priority that ensures that certain capsules obey the call-by-name discipline, while others obey call-by-value. On the other hand, there are reduction sequences that are not π-reductions for any π: this will be the case whenever a redex $\langle \mu\alpha.c \parallel \tilde{\mu}x.c' \rangle$ is contracted in the reduction sometimes as a μ-reduction and sometimes as a $\tilde{\mu}$-reduction.

Our goal is to prove that if t is a $\mathcal{M}^{\cap\cup}$-typable term then any reduction out of t that respects *any* priority must be finite.

Definition 6.3 *Fix a priority π. For each type T we define a set $(\mathcal{S}^\pi)^T$ of terms; we let $(\mathcal{S}^\pi)^T_r$ and $(\mathcal{S}^\pi)^T_e$ denote the corresponding sets of callers and callees at each type T.*

To keep the notation manageable, we will simply write $\mathcal{S}^T$, $\mathcal{S}^T_r$ and $\mathcal{S}^T_e$ whenever a fixed π is under consideration.

This definition is by induction on types, with a sub-induction over the priority π.

- *α is in $\mathcal{S}^T_e$ for all T, and x is in $\mathcal{S}^T_r$ for all T.*
- *$r \bullet e$ is in $\mathcal{S}^{A\to B}_e$ if r is in $\mathcal{S}^A_r$ and e is in $\mathcal{S}^B_e$*
- *$\lambda x.r$ is in $\mathcal{S}^{A\to B}_r$ if for all $e \in \mathcal{S}^B_e$, $\tilde{\mu}x.\langle r \parallel e \rangle$ is in $\mathcal{S}^A_e$.*
- *$\tilde{\mu}x.c$ is in $\mathcal{S}^T_e$ if for all r in $\mathcal{S}^T_r$ which is available to $\tilde{\mu}x.c$ under π, $c[x \leftarrow r]$ is SN.*
- *$\mu\alpha.c$ is in $\mathcal{S}^T_r$ if for all e in $\mathcal{S}^T_e$ which is available to $\mu\alpha.c$ under π, $c[\alpha \leftarrow e]$ is SN.*
- *$t \in \mathcal{S}^{T_1\cap T_2}_r$ if $t \in \mathcal{S}^{T_1}_r \cap \mathcal{S}^{T_2}_r$, and $t \in \mathcal{S}^{T_1\cap T_2}_e$ if $t \in \mathcal{S}^{T_1}_e \cup \mathcal{S}^{T_2}_e$*
- *$t \in \mathcal{S}^{T_1\cup T_2}_r$ if $t \in \mathcal{S}^{T_1}_r \cup \mathcal{S}^{T_2}_r$, and $t \in \mathcal{S}^{T_1\cup T_2}_e$ if $t \in \mathcal{S}^{T_1}_e \cap \mathcal{S}^{T_2}_e$*

□

This definition is sensible because we have assumed that π is a well-ordering. Note the duality between $\cap$ and $\cup$ in the last four clauses of the definition.

Two easy consequences of the definitions are:

- For each priority π and each type T, all caller variables are in $(\mathcal{S}^\pi)^T_r$ and all callee variables are in $(\mathcal{S}^\pi)^T_e$.
- For each priority π and each type T, $(\mathcal{S}^\pi)^T_r \subseteq SN^\pi$ and $(\mathcal{S}^\pi)^T_e \subseteq SN^\pi$.

Lemma 6.4 *For each priority π and each type T, if $r \in (\mathcal{S}^\pi)^T_r$ and $e \in (\mathcal{S}^\pi)^T_e$ then $\langle r \parallel e\rangle$ is in SN^π.*

Proof. We know that r and e are SN^π, so we may reason by multiset induction over the π-reduction relation out of r and e. Then it suffices to argue that the result of immediately contracting the top-level redex of $\langle r \parallel e\rangle$, if any, is SN^π.

This leads to consideration of the following cases

1. $\langle \mu\alpha.c \parallel e\rangle \longrightarrow c[\alpha \leftarrow e]$
2. $\langle r \parallel \tilde{\mu}x.c'\rangle \longrightarrow c'[x \leftarrow r]$
3. $\langle \lambda x.r \parallel a \bullet e'\rangle \longrightarrow \langle a \parallel \tilde{\mu}x.\langle r \parallel e'\rangle\rangle$

For cases 1 and 2 the result is immediate by definition of $(\mathcal{S}^\pi)^T$ and the assumption that the reduction respects π.

For case 3, we know that a is in $(\mathcal{S}^\pi)^A_r$ and e' is in $(\mathcal{S}^\pi)^B_e$. The latter fact together with $(\lambda x.r) \in (\mathcal{S}^\pi)^{A\to B}_r$ implies that $\tilde{\mu}x.\langle r \parallel e'\rangle$ is in $(\mathcal{S}^\pi)^A_e$. So the capsule $\langle a \parallel \tilde{\mu}x.\langle r \parallel e'\rangle\rangle$ is an instance of case 2, already treated. □

Theorem 6.5 (Soundness) *Let π be a priority. If term t is typable with type A then t is in $(\mathcal{S}^\pi)^A$.*

Proof. Let us say that a substitution θ *satisfies* Γ under π if whenever $(x : A)$ is in Γ, $\theta x \in (\mathcal{S}^\pi)^A_r$, and that θ *satisfies* Δ under π if whenever $(\alpha : A)$ is in Δ, $\theta\alpha \in (\mathcal{S}^\pi)^A_e$. Then to prove the theorem it is convenient to prove the following stronger statement: *Suppose θ satisfies Γ and Δ under π. If $\Gamma \vdash \boxed{r : A}, \Delta$, then $\theta r \in (\mathcal{S}^\pi)^A_r$, and if $\Gamma, \boxed{e : B} \vdash \Delta$, then $\theta e \in (\mathcal{S}^\pi)^A_e$.* This implies the theorem since the identity substitution satisfies every Γ and Δ. We prove the statement above by induction on typing derivations.

For the rest of the proof we suppress writing π and write $\mathcal{S}$ instead of $(\mathcal{S}^\pi)$, and SN instead of SN^π. Choose a substitution θ that satisfies Γ and Δ, and a typable term r or e; we wish to show that $\theta r \in \mathcal{S}^A_r$ or $\theta e \in \mathcal{S}^A_e$, as appropriate. We consider the possible forms of the given typing.

Case:

$$\frac{}{\Gamma,\ \boxed{\alpha : A_i}\ \vdash\ \alpha : A_1 \cup \cdots \cup A_n, \Delta}\ (e^+\text{-}ax)$$

When the term is a variable α typed as above, we need to show that $\theta\alpha \in \mathcal{S}^{A_i}_e$, under the assumption that $\theta\alpha \in \mathcal{S}^{A_1\cup\cdots\cup A_n}_e$. Since $\mathcal{S}^{A_1\cup\cdots\cup A_n}_e = \mathcal{S}^{A_1}_e \cap \cdots \cap \mathcal{S}^{A_n}_e$ this is clear.

Case:

$$\frac{}{\Gamma, x : A_1 \cap \cdots \cap A_n\ \vdash\ \boxed{x : A_i},\ \Delta}\ (r^+\text{-}ax)$$

When the term is a variable x typed as above, we need to show that $\theta x \in \mathcal{S}^{A_i}_r$, under the assumption that $\theta x \in \mathcal{S}^{A_1\cap\cdots\cap A_n}_r$. Since $\mathcal{S}^{A_1\cap\cdots\cap A_n}_r = \mathcal{S}^{A_1}_r \cap \cdots \cap \mathcal{S}^{A_n}_r$ this is clear.

Case:

$$\frac{\Gamma \vdash \boxed{r : A}, \Delta \quad \Gamma, \boxed{e : B} \vdash \Delta}{\Gamma, \boxed{r \bullet e : A \to B} \vdash \Delta} \; (\to L)$$

We wish to show that $(\theta r \bullet \theta e) \in \mathcal{S}_e^{A \to B}$. It suffices to show that $(\theta r \bullet \theta e) \in \mathcal{S}_e^{A \to B}$, which is immediate since by the induction hypothesis $\theta r \in \mathcal{S}_r^A$ and $\theta e \in \mathcal{S}_e^B$.

Case:

$$\frac{\Gamma, x : A \vdash \boxed{r : B}, \Delta}{\Gamma \vdash \boxed{\lambda x.r : A \to B}, \Delta} \; (\to R)$$

We wish to show that $\theta(\lambda x.\theta r) \in \mathcal{S}_r^{A \to B}$. It suffices to show that for every e in $\mathcal{S}_e^A$, $\widetilde{\mu} x.\langle \theta r \parallel e \rangle \in \mathcal{S}_e^A$. For the latter it suffices to consider an arbitrary a that is available to $\widetilde{\mu} x.\langle \theta r \parallel e \rangle$ under π and argue that $\langle \theta r \parallel e \rangle [x \leftarrow a]$ is SN^π. Since we may assume that x is not free in e we must argue that $\langle (\theta r)[x \leftarrow a] \parallel e \rangle$ is SN^π. But $(\theta r)[x \leftarrow a] \in \mathcal{S}_r^B$ by induction, and we have assumed $e \in \mathcal{S}_e^B \subseteq SN$.

Case:

$$\frac{\dfrac{\Gamma, x : A \;\vdash\; \boxed{r : T}, \;\Delta \quad \Gamma, x : A, \;\boxed{e : T} \;\vdash\; \Delta}{\langle r \parallel e \rangle \;:\; (\Gamma, x : A \;\vdash\; \Delta)} \; (cut)}{\Gamma, \;\boxed{\widetilde{\mu} x.\langle r \parallel e \rangle : A} \;\vdash\; \Delta} \; (\mu)$$

Note that any application of typing rule $(\widetilde{\mu})$ must indeed immediately follow a cut. We wish to show that $\theta(\widetilde{\mu} x.\langle r \parallel e \rangle) \equiv (\widetilde{\mu} x.\langle \theta r \parallel \theta e \rangle) \in \mathcal{S}_e^A$. For this we need to consider an r_1 in $\mathcal{S}_r^A$ that is available to $(\widetilde{\mu} x.\langle \theta r \parallel \theta e \rangle)$ under π and argue that

$$\langle \theta r \parallel \theta e \rangle [x \leftarrow r_1] \quad \text{is } SN^\pi.$$

Letting θ' denote the substitution obtained by augmenting θ with the binding $x \mapsto r_1$, what we want to show is that $\langle \theta' r \parallel \theta' e \rangle$ is SN^π. The substitution θ' satisfies $\Gamma, x : A$ and Δ by hypothesis and the fact that $r_1 \in \mathcal{S}_r^A$. So $\theta' r \in \mathcal{S}_r^T$ and $\theta' e \in \mathcal{S}_e^T$ by induction hypothesis, and so $\langle \theta' r \parallel \theta' e \rangle$ is SN^π.

Case:

$$\frac{\dfrac{\Gamma \;\vdash\; \boxed{r : T}, \;\alpha : A, \Delta \quad \Gamma, \;\boxed{e : T} \;\vdash\; \alpha : A, \Delta}{\langle r \parallel e \rangle \;:\; (\Gamma \;\vdash\; \alpha : A, \Delta)} \; (cut)}{\Gamma \;\vdash\; \boxed{\mu\alpha.\langle r \parallel e \rangle : A}, \;\Delta} \; (\mu)$$

Note that any application of typing rule (μ) must indeed immediately follow a cut. We wish to show that $(\mu\alpha.\langle \theta r \parallel \theta e \rangle) \in \mathcal{S}_r^A$. For this we need to consider an e_1 in $\mathcal{S}_e^A$ that is available to $(\mu\alpha.\langle \theta r \parallel \theta e \rangle)$ under π and argue that

$$\langle \theta r \parallel \theta e \rangle [\alpha \leftarrow e_1] \quad \text{is } SN^\pi.$$

Letting θ' denote the substitution obtained by augmenting θ with the binding $\alpha \mapsto e_1$, what we want to show is that $\langle \theta' r \parallel \theta' e \rangle$ is SN^π.

The substitution θ' satisfies Γ and $\Delta, \alpha : A$ by hypothesis and the fact that $e_1 \in \mathcal{S}_e^A$. So $\theta' r \in \mathcal{S}_r^T$ and $\theta' e \in \mathcal{S}_e^T$ by induction hypothesis, and so $\langle \theta' r \parallel \theta' e \rangle$ is SN^π.

Case:

$$\frac{\Gamma,\ \boxed{e:A}\ \vdash\ \Delta}{\Gamma,\ \boxed{e:A\cap B}\ \vdash\ \Delta}\ (\cap_L)$$

We wish to show that $\theta e \in \mathcal{S}_e^{A\cap B}$. Since $\mathcal{S}_e^{A\cap B} = \mathcal{S}_e^A \cup \mathcal{S}_e^B$ this follows from the fact that (by induction hypothesis) $\theta e \in \mathcal{S}_e^A$.

Case:

$$\frac{\Gamma\ \vdash\ \boxed{r:A},\ \Delta \qquad \Gamma\ \vdash\ \boxed{r:B},\ \Delta}{\Gamma\ \vdash\ \boxed{r:A\cap B},\ \Delta}\ (\cap_R)$$

We wish to show that $\theta r \in \mathcal{S}_r^{A\cap B}$. Since $\mathcal{S}_r^{A\cap B} = \mathcal{S}_r^A \cap \mathcal{S}_r^B$ this follows from the fact that (by induction hypothesis) $\theta r \in \mathcal{S}_r^A$ and $\theta r \in \mathcal{S}_r^B$.

Case:

$$\frac{\Gamma,\ \boxed{e:A}\ \vdash\ \Delta \qquad \Gamma,\ \boxed{e:B}\ \vdash\ \Delta}{\Gamma,\ \boxed{e:A\cup B}\ \vdash\ \Delta}\ (\cup_L)$$

We wish to show that $\theta e \in \mathcal{S}_e^{A\cup B}$. Since $\mathcal{S}_e^{A\cup B} = \mathcal{S}_e^A \cap \mathcal{S}_e^B$ this follows from the fact that (by induction hypothesis) $\theta e \in \mathcal{S}_e^A$ and $\theta e \in \mathcal{S}_e^B$.

Case:

$$\frac{\Gamma\ \vdash\ \boxed{r:A},\ \Delta}{\Gamma\ \vdash\ \boxed{r:A\cup B},\ \Delta}\ (\cup_R)$$

We wish to show that $\theta r \in \mathcal{S}_r^{A\cup B}$. Since $\mathcal{S}_r^{A\cup B} = \mathcal{S}_r^A \cup \mathcal{S}_r^B$ this follows from the fact that (by induction hypothesis) $\theta r \in \mathcal{S}_r^A$.

Theorem 6.6 *For every priority π, every typable term is SN^π.*

Proof. By Theorem 6.5 and the fact that every term in $\mathcal{S}$ is SN^π.

The above theorem has as immediate corollaries the fact that every typable term is strongly normalizing under both the the call-by-name and the call-by-value variants of $\overline{\lambda}\mu\widetilde{\mu}$.

Corollary 6.7 *The following are equivalent, for a term t:*

- *t is strongly normalizing under some priority.*
- *t is strongly normalizing under every priority.*

In particular, the set of terms that are strongly normalizing in $\overline{\lambda}\mu\widetilde{\mu}$ under the call-by-name and the call-by-value disciplines coincide.

Proof. If t is strongly normalizing under some priority then by Theorem 5.3 t is typable. So by Theorem 6.6 t is strongly normalizing under every priority.

7 Conclusion

We conclude by noting some directions for future research.

Intersection types have proven to be an invaluable tool for studying reduction properties in the traditional λ-calculus, and in future work we expect to use suitable variants on the system presented here to characterize weak normalization and head-normalization in $\overline{\lambda}\mu\widetilde{\mu}$.

It will be interesting to investigate the relationship between the system presented here and that of Laurent [17].

As mentioned in Section 6 there are technical difficulties in lifting the reducibility technique in the presence of intersections and unions to the unrestricted $\overline{\lambda}\mu\widetilde{\mu}$ calculus with the $(\mu, \widetilde{\mu})$ critical pair. This would seem to be an important step in applying intersection-types techniques to the denotational semantics of non-confluent classical calculi. Solving this problem seems to require a deeper understanding of the combinatorial interaction between μ- and $\widetilde{\mu}$-reductions.

It is important to better understand the role of union types in a classical calculus, in light of the work in [23] and [7] showing how union types can play a role in the design of programming languages and query languages for semistructured data types, and especially the results in [20] and [30] relating unions to flow analysis. An obvious question is whether the price we pay for having Subject Reduction in our system is a decrease in expressive power relative to those systems, and if so, we should try to understand the trade-offs.

Acknowledgments

We are grateful to Pierre-Louis Curien, Norman Danner, Hugo Herbelin, St´ephane Lengrand, and Michel Parigot for several conversations about this work.

References

[1] Z. M. Ariola and H. Herbelin. Minimal classical logic and control operators. In *ICALP: Annual International Colloquium on Automata, Languages and Programming*, volume 2719 of *Lecture Notes in Computer Science*, pages 871–885. sv, 2003.

[2] F. Barbanera and S. Berardi. A symmetric lambda calculus for classical program extraction. *Information and Computation*, 125(2):103–117, 1996.

[3] F. Barbanera, M. Dezani-Ciancaglini, and U. de' Liguoro. Intersection and union types: syntax and semantics. *Information and Computation*, 119(2):202–230, 1995.

[4] H. P. Barendregt, M. Coppo, and M. Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *Journal of Symbolic Logic*, 48(4):931–940 (1984), 1983.

[5] H. P. Barendregt and S. Ghilezan. Lambda terms for natural deduction, sequent calculus and cut-elimination. *Journal of Functional Programming*, 10(1):121–134, 2000.

[6] G. M. Bierman. A computational interpretation of the $\lambda\mu$-calculus. In *Proceedings of Symposium on Mathematical Foundations of Computer Science.*, volume 1450 of *Lecture Notes in Computer Science*, pages 336–345. Springer-Verlag, 1998.

[7] P. Buneman and B. Pierce. Union types for semistructured data. In *Internet Programming Languages*, volume 1686 of *Lecture Notes in Computer Science*. Springer-Verlag, September 1998. Proceedings of the International Database Programming Languages Workshop.

[8] M. Coppo, M. Dezani-Ciancaglini, and B. Venneri. Principal type schemes and λ-calculus semantics. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 535–560. Academic Press, London, 1980.

[9] P.-L. Curien. Symmetry and interactivity in programming. *Archive for Mathematical Logic*, 2001. to appear.

[10] P.-L. Curien. Abstract machines, control, and sequents. In *Applied Semantics, International Summer School, APPSEM 2000, Caminha, Portugal, September 9-15, 2000, Advanced Lectures*, volume 2395 of *Lecture Notes in Computer Science*, pages 123–136. Springer-Verlag, 2002.

[11] P.-L. Curien and H. Herbelin. The duality of computation. In *Proceedings of the 5th ACM SIGPLAN International Conference on Functional Programming (ICFP'00)*, Montreal, Canada, 2000. ACM Press.

[12] Ph. de Groote. On the relation between the $\lambda\mu$-calculus and the syntactic theory of sequential control. In Springer-Verlag, editor, *Proceedings of the (th International Conference on Logic Programming and Automated Reasoning, (LPAR'94)*, volume 822 of *Lecture Notes in Computer Science*, pages 31–43, 1994.

[13] J.-Y. Girard. A new constrcutive logic: classical logic. *Mathematical Structures in Computer Science*, 1(3):255–296, 1991.

[14] T. Griffin. A formulae-as-types notion of control. In *Proceedings of the 17th Annual ACM Symposium on Principles Of Programming Languages, Orlando (Fla., USA)*, pages 47–58, 1990.

[15] H. Herbelin. *Séquents qu'on calcule : de l'interprétation du calcul des séquents comme calcul de λ-termes et comme calcul de stratégies gagnantes*. Thèse d'universit´e, Universit´e Paris 7, Janvier 1995.

[16] J. R. Hindley. Coppo–Dezani types do not correspond to propositional logic. *Theoretical Computer Science*, 28(1-2):235–236, 1984.

[17] O. Laurent. On the denotational semantics of the pure lambda-mu calculus. *Manuscript*, 2004.

[18] S. Lengrand. Call-by-value, call-by-name, and strong normalization for the classical sequent calculus. In Bernhard Gramlich and Salvador Lucas, editors, *Electronic Notes in Theoretical Computer Science*, volume 86. Elsevier, 2003.

[19] C.-H. L. Ong and C. A. Stewart. A Curry-Howard foundation for functional computation with control. In *Proceedings of the 24th Annual ACM Symposium on Principles Of Programming Languages, Paris (France)*, pages 215–227, 1997.

[20] J. Palsberg and C. Pavlopoulou. From polyvariant flow information to intersection and union types. *J. Funct. Programming*, 11(3):263–317, 2001.

[21] M. Parigot. An algorithmic interpretation of classical natural deduction. In *Proc. of Int. Conf. on Logic Programming and Automated Reasoning, LPAR'92*, volume 624 of *Lecture Notes in Computer Science*, pages 190–201. Springer-Verlag, 1992.

[22] M. Parigot. Proofs of strong normalisation for second order classical natural deduction. *The Journal of Symbolic Logic*, 62(4):1461–1479, December 1997.

[23] B. C. Pierce. Programming with intersection types, union types, and polymorphism. Technical Report CMU-CS-91-106, Carnegie Mellon University, February 1991.

[24] E. Polonovski. Strong normalization of $\lambda\mu\tilde{\mu}$-calculus with explicit substitutions. In Igor Walukiewicz, editor, *Foundations of Software Science and Computation Structures, 7th International Conference, FOSSACS 2004*, volume 2987 of *Lecture Notes in Computer Science*, pages 423–437. Springer, 2004.

[25] G. Pottinger. Normalization as homomorphic image of cut-elimination. *Annals of Mathematical Logic*, 12:323–357, 1977.

[26] G. Pottinger. A type assignment for the strongly normalizable λ-terms. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 561–577. Academic Press, London, 1980.

[27] J. C. Reynolds. Design of the programming language Forsythe. Report CMU–CS–96–146, Carnegie Mellon University, Pittsburgh, Pennsylvania, June 28, 1996.

[28] C. Urban and G. M. Bierman. Strong normalisation of cut-elimination in classical logic. In *Typed Lambda Calculus and Applications*, volume 1581 of *Lecture Notes in Computer Science*, pages 365–380, 1999.

[29] P. Wadler. Call-by-value is dual to call-by-name. In *Proceedings of the 8th International Conference on Functional Programming*, 2003.

[30] J. B. Wells, Allyn Dimock, Robert Muller, and Franklyn Turbak. A calculus with polymorphic and polyvariant flow types. *J. Funct. Programming*, 12(3):183–227, May 2002.