Abstracting Models of Strong Normalization for Classical Calculi

Paul Downen, Philip Johnson-Freyd, Zena M. Ariola

University of Oregon
{pdownen,philipjf,ariola}@cs.uoregon.edu

Abstract

Modern programming languages have effects and mix multiple calling conventions, and their core calculi should too. We characterize calling conventions by their "substitution discipline" that says what variables stand for, and design calculi for mixing disciplines in a single program. Building on variations of the reducibility candidates method, including biorthogonality and symmetric candidates which are both specialized for one discipline, we develop a single uniform framework for strong normalization encompassing call-by-name, call-by-value, call-by-need, call-by-push-value, non-deterministic disciplines, and any others satisfying some simple criteria. We explicate commonalities of previous methods and show they are special cases of the uniform framework and they extend to multi-discipline programs.

Keywords: Strong normalization, calling convention, biorthogonality, symmetric candidates, sequent calculus

1. Introduction

Picking a programming language means choosing not just a syntax and feature set, but also a calling convention. As Peyton Jones (2009) says:

> These days, the strict/lazy decision isn't a straight either/or choice. For example, a lazy language has ways of stating "use call-by-value here," and even if you were to say "Oh, the language should be call by value," you would want ways to achieve laziness anyway. Any successor language to Haskell will have support for both strict and lazy functions. So the question then is: "How do you mix them together?"

This question is as important in language theory as it is in practice: different programming languages merit different calculi. For example, just $\beta\eta$ axioms are enough for equality of call-by-name functions, but more axioms are needed to complete the theory of call-by-value (Sabry and Felleisen, 1992; Herbelin and Zimmermann, 2009). More drastically, call-by-need requires some extra rules even for computing answers. If we then want to reflect the reality of programming languages that mix calling conventions, we need a theory that mixes them, too. Again, the question is: "How?"

Polarized logic (Zeilberger, 2009; Munch-Maccagnoni, 2013) and call-by-push-value (Levy, 2001) partially answers the question of how to mix calling conventions by dividing types into two groups: positive and negative. The positive types, like sums, follow the call-by-value discipline whereas the negative types, like functions, follow the call-by-name regime. Here, by contrast, we do connect calling conventions with types, but allow each type constructor to build a type of any convention; for example we can have both a call-by-value or a call-by-need function type. This more closely reflects practice where OCaml has call-by-value functions and Haskell call-by-need sums.

Even though each calculus for each convention is different, they can be all be seen as variations on the same idea. As pioneered by Ronchi Della Rocca and Paolini (2004), calculi for different calling conventions can be summarized as instances of a common calculus parameterized by a *substitution discipline* (Downen and Ariola, 2014) specifying what might be substituted for identifiers. Call-by-name and -value can then share the same $\beta\eta$ axioms, $(\lambda x.M)\ V = M\{V/x\}$ and $\lambda x.V\ x = V$; what changes is the notion of *value* V. Call-by-name says that V can be any term, and call-by-value is more restrictive. Each of the above mentioned three calling conventions can be uniformly represented, as well as more exotic ones like the dual to call-by-need (Ariola et al., 2011) and the non-deterministic evaluation of the symmetric λ-calculus (Barbanera and Berardi, 1994).

Abstracting away the differences across languages enables us to study properties of those languages in a uniform way. In this paper, we focus on strong normalization. In a reduction theory, a term is strongly normalizing if it is the start of no infinite reduction sequence. Of course, for untyped languages, it is unlikely to be the case that all terms are strongly normalizing, but many typed calculi can be shown to have the property that all their well-typed terms are strongly normalizing. For example, the untyped λ-calculus has non-normalizing terms like $(\lambda x.xx)(\lambda x.xx)$ which reduces to itself, causing the infinite reduction $(\lambda x.xx)(\lambda x.xx) \to (\lambda x.xx)(\lambda x.xx) \to \ldots$,

but the simply typed λ-calculus rejects all such terms as ill-typed. Currently, however, there are separate proofs of strong normalization for calculi of different substitution disciplines. Here, we show *one* common proof for all of them by articulating the essential properties of the substitution discipline that guarantees strong normalization. We build on a technique previously used for studying a language of mixed induction and co-induction (Downen et al., 2015), which is based on both biorthogonal (Girard, 1987; Krivine, 2005; Pitts, 2000) and symmetric candidate (Barbanera and Berardi, 1994) models, and extend it to accommodate multi-discipline languages. Further, the more refined version of the technique presented here lets us formally understand the relationship between orthogonality and symmetric candidates: biorthogonality models are subsumed as a special case of our uniform model.

The orthogonality-based family of methods require that we not only think of how to create values of a type, but also how to use them. This inevitably leads to the invention of abstract-machine-like constructs to represent a reified environment or context of a program fragment (Krivine, 2005; Pitts, 2000). Instead of going about an ad hoc reification, we base our proof on a classical sequent calculus which is already an abstract machine language (Section 2) that is well-suited to mixing disciplines (Section 3).

This work is based on (Downen et al., 2018) which uses a sequent calculus with impredicative polymorphism based on (Downen et al. (2015), see Section 1.1 for comparison) and is extended with multiple disciplines—which are given as a parameter to the system and not fixed a priori—in the sense that different calling conventions can be used in the same program (Sections 2 and 3). From that starting point, we:

- (Section 4) Define and explore the parametric, multi-discipline sequent calculus with functions and polymorphism. We isolate the well-disciplined sub-syntax, which is more lax than full typing, for the purpose of defining the untyped reduction theory. We also define a collection of five particular disciplines corresponding to five important calling conventions—call-by-name, -value, -need, -co-need, and non-deterministic—for which we give a type-safe operational semantics.

  As it covers the main properties of our calculus *other* than strong normalization, this, together with the previous chapters may be of interest even to readers who would like to skip the more technical developments to follow.

- (Section 5) Analyze the properties of rewriting theories for the multidiscipline sequent calculus, giving particular focus to the notion of *top reduction*—an important subset of standard reduction—as a way to generically prove a partial standardization result and identify how reduction relates to the priorities between producers and consumers, which we characterize by the reduction's *charge*.

- (Section 6) Introduce the primary setting of our model: *pre-candidates*; objects which represent both the terms and co-terms of a type, but without the full safety properties. Already, pre-candidates support useful properties, including two separate complete lattices: one of straightforward containment and the other corresponding to behavioral subtyping (Liskov, 1987). We also generalize the notion of orthogonality (Krivine, 2005) to allow for different predicates on the two sides of a pre-candidate, and the usual properties of orthogonality generalize to pre-orthogonality.

- (Section 7) Demonstrate a generalization of the traditional biorthogonality technique (Girard, 1987; Pitts, 2000; Mellies and Vouillon, 2005) which goes beyond modeling just rewriting theories with call-by-name or call-by-value calling conventions, but can also handle other calling conventions like call-by-need and its dual as well. The key is to construct reducibility candidates (*i.e.,* well-behaved models of types) with a built-in "value-restriction" property (Munch-Maccagnoni, 2009) and to use *three* layers of orthogonality. As it turns out, both of these additions simplify away for the call-by-name and call-by-value calling conventions, so the more general construction boils down to the usual biorthogonality construction for these special cases.

- (Section 8) Move beyond modeling just confluent reduction to non-confluent reduction by focusing on fixed point constructions. We recast the symmetric candidates technique (Barbanera and Berardi, 1994; Lengrand and Miquel, 2008) to be less syntactic and more semantic, based not on special syntactic forms but instead on the behavior of terms and co-terms under charged reduction. Our use of pre-candidates demonstrates how the fixed-point solution of symmetric candidates is based on the relationship between pre-orthogonality and behavioral subtyping. And the added generality to the model lets us formalize the relationship between reducibility candidates and symmetric candidates:

the former is wholly subsumed by the latter, and the two concepts are identical for confluent reduction.

- (Section 9) Provide a model of strong normalization in the sequent calculus which is defined uniformly for any collection of admissible disciplines. Each of the five important disciplines above are admissible, which proves strong normalization for a general calculus mixing the most common calling conventions.

1.1. Relation to Previous Work

The proof of strong normalization in this paper is a substantial refinement of the proof of strong normalization which appeared in (Downen et al., 2015). That paper already treated discipline as a *parameter* of the theory, but did not allow for mixing disciplines. As such, the present proof accounts for the possibility that by combining two strongly normalizing disciplines we might break normalization. It does so by ensuring sufficient safety in terms of when disciplines will be mixed and ensuring that each discipline is still *admissible* when used in combination (see Property 3 – note that this property is non-trivial for the combination of call-by-need and its dual as their syntactic definitions overlap).

Beyond this difference in application, the model has been notably improved over our previous work. David and Nour (2005) answered the question of "why the usual [orthogonality based] candidates of reducibility do not work" for the non-deterministic discipline (**u**), but, from our perspective, left open the question of why they do work for deterministic disciplines. We believe the proof in this paper finally answers that question.

In our previous work we used symmetric candidates, with its infinite fixed point construction, to build modified orthogonality based candidates. That was sufficient for building candidates and showing normalization, but it provided little understanding of what was going on and we could not say much about what those candidates looked like. Here, our analysis is finer, and we are able to formally relate reducibility candidates with symmetric candidates, showing that the former are the unique manifestation of the later when dealing with deterministic strategies. We now know that "the usual candidates of reducibility" method works for deterministic strategies because it *is* the symmetric candidates method—for deterministic disciplines possible candidates are unique, and when they exist must be given by orthogonals. The fixed point construction of symmetric candidates ensures

existence of candidates for all disciplines, deterministic or not, whereas the direct orthogonal construction gives an immediate definition of candidates. This difference between the two forms of candidates—which we now know are closely related—allows for a more thorough understanding of the impact of disciplines on advanced type structures like intersection and union types (Downen and Ariola, 2019).

Numerous improvements in our model construction, small on their own perhaps but quite substantial in totality, enable this finer analysis. Our notion of "pre-candidate" now considers only strongly normalizing (co-)terms to begin with, simplifying many arguments. We also now use simpler and weaker commutation lemmas, suitable for a wider variety of systems. More fundamentally, we now have separate notions of "reducibility candidate" and "symmetric candidate" and rigorously relate the two. The former is now required to contain its own value orthogonal, a requirement absent in our previous work and necessary to the uniqueness results. The notion of "head" in our prior work has been eliminated, replaced by a more general saturation condition, while the idea of "forward closure" of a pre-candidate is now mostly handled by the orthogonality machinery (it is still used in the statement of Lemma 7 which is a technical device to get to Corollary 1). Most importantly, we have now gained a detailed analysis of deterministic normalization, allowing us to state and prove Theorem 3: symmetric candidates are always reducibility candidates, and the two notions coincide for deterministic disciplines.

Concurrently with the previous version of this article being presented, Miquey and Herbelin (2018) gave an orthogonality based proof of strong normalization for a call-by-need calculus with control, complementing the symmetric candidates based proof of strong normalization of call-by-need from our earlier work. While their system is somewhat different from ours in that their call-by-need makes use of an *environment*, it is notable that they used a similar sequence of three applications of the (restricted) orthogonal as appears in this work. As the present paper shows such candidates to be unique (at least for our calculus) we gain the insight that this triple orthogonal construction produces *exactly the same model* as the symmetric candidates technique.

2. A Linguistic Approach to Abstract Machines

One of the most basic ways of evaluating a λ-calculus term is by repeated β reduction. For instance, if we have the term $(\lambda x.\lambda y.x + y)\ 1\ 2$ we can compute a value in three steps:

$$(\lambda x.\lambda y.x + y)\ 1\ 2 \to (\lambda y.1 + y)\ 2 \to 1 + 2 \to 3$$

However, even in this simple example we can observe one frustration with the β-reduction model from the perspective of implementation: reductions might not always occur at the "top" of the term, but can be buried somewhere within it. In the very first reduction step above, the redex $(\lambda x.\lambda y.x + y)\ 1$ subjected to β reduction happens inside of the outermost application context $\square\ 2$, where $\square$ stands for the position of the sub-term within the context. As such, performing evaluation by β reduction requires a search for the next redex within a term, which must be specified as part of an implementation of the evaluator.

An *abstract machine* gives a lower-level description of evaluation by interweaving search and reduction together. To keep track of its position within the term, a machine does not evaluate terms directly but rather larger configurations. Here, the configurations we use are called *commands* (denoted by the metavariable c) which consist of a term (denoted by v) together with a syntactic representation of its context called *co-term* (denoted by e). One abstract machine in this style is the Krivine machine (Krivine, 2007), which requires only two rules:

$$\langle v\ v' \| e \rangle \to \langle v \| v' \bullet e \rangle \qquad\qquad \langle \lambda x.v \| v' \bullet e \rangle \to \langle v\{v'/x\} \| e \rangle$$

The first rule pushes the argument of a function call onto the call-stack. In other words, evaluating an application of the form $v\ v'$ in a surrounding context e consists of pushing the argument v' on top of e and then evaluating v in the larger context. The second rule implements β reduction by popping the top argument off of a call-stack and plugging it into the formal parameter of a λ-abstraction. In the Krivine machine style, our previous example can be computed as follows, where the term is evaluated in a context named α:

$$\begin{aligned}\langle (\lambda x.\lambda y.x + y)\ 1\ 2 \| \alpha \rangle &\to \langle (\lambda x.\lambda y.x + y)\ 1 \| 2 \bullet \alpha \rangle \\ &\to \langle \lambda x.\lambda y.x + y \| 1 \bullet 2 \bullet \alpha \rangle \\ &\to \langle \lambda y.1 + y \| 2 \bullet \alpha \rangle \to \langle 1 + 2 \| \alpha \rangle \to \langle 3 \| \alpha \rangle\end{aligned}$$

So the machine returns the same result, 3, to the surrounding context as was achieved by β reduction. The Krivine machine thus seems to represent a lower level implementation, one closer to actual computation on a physical machine using call-stacks. Moreover, exploring the laws of the Krivine machine suggests additional possibilities. We see in the Krivine machine that there are actually two different syntactic constructs for invoking a function: both configurations $\langle \Box \| v \bullet e \rangle$ and $\langle \Box\ v \| e \rangle$ do exactly the same thing as the second is rewritten into the first. That is, both call-stack formation and λ calculus application are two ways of expressing the same concept. It is thus natural to wonder if the two can be unified.

We are accustomed to having variables stand for an unknown value and then having the possibility to bind these variables to known terms later. The same can be done with respect to contexts, now that they are embodied with a syntactic representation in the form of co-terms. Already in the example above we refer to α (called a co-variable) as a generic placeholder for the surrounding context of evaluation. Abstracting over co-variables like α is the role of the μ-abstraction, written as $\mu\alpha.c$, which is reduced like so:

$$\langle \mu\alpha.c \| e \rangle \rightarrow c\{e/\alpha\}$$

The above says that when the term $\mu\alpha.c$ is evaluated in a context e, then the next step is to execute the command c with α bound to e. μ-abstractions unify the two forms of function calls by representing function application in terms of call-stack formation. For example, the above λ-calculus term $(\lambda x.\lambda y.x+y)\ 1\ 2$ can be rewritten to avoid function application altogether as $\mu\beta.\langle \lambda x.\lambda y.x + y \| 1 \bullet 2 \bullet \beta \rangle$. This term behaves the same as the original one:

$$\langle \mu\beta.\langle \lambda x.\lambda y.x + y \| 1 \bullet 2 \bullet \beta \rangle \| \alpha \rangle \rightarrow \langle \lambda x.\lambda y.x + y \| 1 \bullet 2 \bullet \alpha \rangle$$

As such, the application term $v\ v'$ becomes syntactic sugar for $\mu\alpha.\langle v \| v' \bullet \alpha \rangle$.

However, the presence of μ-abstraction makes the language more expressive than λ-calculus because a μ has the ability to *erase* its context when the abstracted co-variable is never used:

$$\langle \mu\beta.\langle \lambda x.\lambda y.x + y \| \alpha \rangle \| 1 \bullet 2 \bullet \alpha \rangle \rightarrow \langle \lambda x.\lambda y.x + y \| \alpha \rangle$$

A μ-abstraction can also *duplicate* its context by using the abstracted co-variable more than once. Indeed, terms such as $\mu\alpha.c$ create a *control* effect much like those found in many programming languages. In particular, μ-abstractions are similar to the `callcc` operator from Scheme.

So far, this analysis gives rise to a language for representing abstract machines implementing call-by-name evaluation. But what about call-by-value evaluation, where arguments are evaluated before resolving a function application, giving rise to evaluation contexts of the form $V\ \Box$ (where V denotes a value: a variable or a λ-abstraction) in addition to $\Box\ v$? The call-by-value version of the above Krivine machine would use an extra co-term $V \circ e$ corresponding to the additional form of evaluation context (first apply V to the input and return the result to e), as well as the reduction rules:

$$\begin{aligned}\langle v\ v' \| e\rangle &\to \langle v \| v' \bullet e\rangle \\ \langle V \| v' \bullet e\rangle &\to \langle v' \| V \circ e\rangle \\ \langle V' \| (\lambda x.v) \circ e\rangle &\to \langle v\{V'/x\} \| e\rangle\end{aligned}$$

The first rule pushes an argument onto the call-stack as before. The second rule switches the attention of the machine from the function, represented by V, to the argument v' beginning evaluation of the argument by placing it on the left-hand side of the command. The third rule implements β reduction slightly differently from before, since the function is now found in the co-term after evaluation due to the second rule. The call-by-value evaluation of our example above becomes:

$$\begin{aligned}\langle (\lambda x.\lambda y.x + y)\ 1\ 2 \| \alpha\rangle &\twoheadrightarrow \langle \lambda x.\lambda y.x + y \| 1 \bullet 2 \bullet \alpha\rangle \\ &\to \langle 1 \| (\lambda x.\lambda y.x + y) \circ (2 \bullet \alpha)\rangle \\ &\to \langle \lambda y.1 + y \| 2 \bullet \alpha\rangle \\ &\to \langle 2 \| (\lambda y.1 + y) \circ \alpha\rangle \to \langle 1 + 2 \| \alpha\rangle \to \langle 3 \| \alpha\rangle\end{aligned}$$

Besides changing the language of co-terms to account for a different evaluation strategy, this presentation of call-by-value machines suffers even worse redundancy: there are *three* different syntactic representations of function invocation—$\langle (\lambda x.v)\ v' \| e\rangle$, $\langle \lambda x.v \| v' \bullet e\rangle$, and $\langle v' \| \lambda x.v \circ e\rangle$—all of which are equivalent to one another. In the interest of eliminating redundancy, we should again wonder if all notions of function invocation can be distilled down to a single primitive operation with the help of some other generic binding constructs, like μ. Indeed, call-by-value can employ the *dual* of μ-abstractions, known as $\tilde{\mu}$-abstractions (Curien and Herbelin, 2000), to write everything with call-stacks. Symmetric to a μ, the $\tilde{\mu}$-abstraction $\tilde{\mu}x.c$ is a co-term that binds its input to the variable x and then runs the command c:

$$\langle v \| \tilde{\mu}x.c\rangle \to c\{v/x\}$$

Just like μ-abstractions can be used to write a λ-calculus application with a call-stack, so too can $\tilde{\mu}$-abstractions be used to write the extra call-by-value evaluation context with the primitive form of call-stack: $v \circ e$ becomes syntactic sugar for $\tilde{\mu}x.\langle v \| x \bullet e \rangle$. Expanding this notation, the second rule is:

$$\langle V \| v' \bullet e \rangle \to \langle v' \| \tilde{\mu}x.\langle V \| x \circ e \rangle \rangle$$

which names the argument for evaluation, and the call-by-value implementation of β reduction simplifies to the call-by-name one:

$$\langle V' \| (\lambda x.v) \circ e \rangle = \langle V' \| \tilde{\mu}y.\langle \lambda x.v \| y \bullet e \rangle \rangle \to \langle \lambda x.v \| V' \bullet e \rangle \to \langle v\{V'/x\} \| e \rangle$$

A calculus for abstract machines These basic constructs—functions and call-stacks, variables and co-variables, μ- and $\tilde{\mu}$-abstractions—define a general calculus for reasoning about abstract machines (both call-by-value and call-by-name) known as system L (Munch-Maccagnoni, 2009). System L is a lower-level machine-like calculus, in that no search is needed for evaluation: reduction can always take place at the "top" of a command. But system L also supports high-level reasoning like the λ-calculus, in that it is still *sound* to perform reductions anywhere within a command, which correspond to out-of-order simplifications and optimizations. Also like the λ-calculus, system L can be seen as either an untyped or typed language. Since there are two different forms of variables—both ordinary variables and co-variables—there are two typing environments: $\Gamma = x_1 : A_1, x_2 : A_2, \ldots, x_n : A_n$ for tracking the types of free variables and $\Delta = \alpha_1 : A_1, \alpha_2 : A_2, \ldots, \alpha_n : A_n$ for tracking the types of free co-variables. Since there are three different forms of expressions—commands, terms, and co-terms—there are three different typing judgements. Terms returning a result of type A in environments Γ and Δ are typed as $\Gamma \vdash v : A \mid \Delta$. Co-terms expecting an input of type A in environments Γ and Δ are typed as $\Gamma \mid e : A \vdash \Delta$. And commands that are capable of running in environments Γ and Δ are typed as $c : (\Gamma \vdash \Delta)$. With this notation in mind, the typing rules for the L-style language of abstract machines are:

$$\frac{\Gamma, x{:}A \vdash v : B \mid \Delta}{\Gamma \vdash \lambda x.v : A \to B \mid \Delta} \qquad \frac{\Gamma \vdash v : A \mid \Delta \quad \Gamma \mid e : B \vdash \Delta}{\Gamma \mid v \bullet e : A \to B \vdash \Delta}$$

$$\frac{}{\Gamma, x{:}A \vdash x : A \mid \Delta} \quad \frac{c : (\Gamma \vdash \alpha{:}A, \Delta)}{\Gamma \vdash \mu\alpha.c : A \mid \Delta} \quad \frac{c : (\Gamma, x{:}A \vdash \Delta)}{\Gamma \mid \tilde{\mu}x.c : A \vdash \Delta} \quad \frac{}{\Gamma \mid \alpha{:}A \vdash \alpha : A, \Delta}$$

$$\frac{\Gamma \vdash v : A \mid \Delta \quad \Gamma \mid e : A \vdash \Delta}{\langle v \| e \rangle : (\Gamma \vdash \Delta)}$$

Amazingly, in the same way that the typing rules for λ-calculus correspond to the rules of natural deduction, the above typing rules correspond to the sequent calculus (Curien and Herbelin, 2000)! The typing rules for call-stacks and commands correspond to the logical rules for implication (on the left) and cut. λ-abstractions are typed as usual, the two axioms correspond to (co-)variables, and the $\mu\tilde{\mu}$ abstractions focus on an assumption or conclusion.

3. Substitution Disciplines: Resolving the Dillema

But there is a problem that rears its head when we try to compute; the fundamental critical pair of classical logic between the μ- and $\tilde{\mu}$-abstractions (Curien and Herbelin, 2000):

$$c_1\{\tilde{\mu}x.c_2/\alpha\} \leftarrow \langle \mu\alpha.c_1 \| \tilde{\mu}x.c_2 \rangle \rightarrow c_2\{\mu\alpha.c_1/x\}.$$

The choice between these two reductions takes us down two separate paths. In the worst case, x and α are never used and c_1 and c_2 are unrelated to one another, which means that a single command can reduce to two completely unrelated results. This critical pair can be resolved by always preferring one reduction or the other, giving two different calculi. Favoring μ by always taking the left path gives the call-by-value calculus, whereas favoring $\tilde{\mu}$ by always taking the right path gives the call-by-name calculus.

As observed by Plotkin (1975), different calling conventions require different calculi: the traditional λ-calculus is suitable for reasoning about Haskell programs, as the call-by-value λ-calculus is for OCaml programs. But denotational semantics seems to capture the essential difference between call-by-name and call-by-value more generally: the difference is reflected in the *Denotable* domain (Turbak et al., 2008). A call-by-name variable can denote any expressible value, including errors or divergence, whereas a call-by-value variable can only denotes "regular" values.

This idea can be represented syntactically by characterizing the calculus in two parts (Ronchi Della Rocca and Paolini, 2004; Downen and Ariola, 2014); one part is common to different parameter passing techniques and the other only differs in one aspect: what can be substituted for a variable and co-variable. We refer to what variables and co-variables stand for as a *substitution discipline*. We call a term that can be substituted for a variable a *value*, and call a co-term that can be substituted for a co-variable a *co-value*. Thus, the call-by-name calculus is defined by saying that *every*

term is a substitutable value, while the set of co-values is restricted to the bare minimum necessary to not get stuck. Symmetrically, the call-by-value calculus is formed by saying that *every* co-term is a co-value, and restricting values down to the bare minimum to avoid getting stuck. Moreover, call-by-name and -value are not the only disciplines expressible in this framework. For instance, call-by-need can be characterized by the notion of substitution discipline as well (Ariola et al., 2011).

Mixing Disciplines This framework allows for a characterization of the differences between calling conventions as a resolution to the above fundamental critical pair, which can be further distilled into a discipline on substitution. Why, then, should only choose one discipline globally for the entire program? Often times such a restriction can be quite limiting. As observed in (Peyton Jones and Launchbury, 1991), some functions like $\lambda x.x + x$ will always evaluate their argument eagerly even in a lazy language, and as such the extra costs associated with lazy evaluation should be avoided when laziness is irrelevant. Thus, it would be more practical to let the programmer, or at least the compiler during code generation and optimization, choose which discipline is appropriate for each juncture. In other words, we want a *multi-discipline* language that incorporates many calling conventions.

The obvious way to signal the intended discipline is to just annotate each command with symbols such as $\mathbf{v}$ (for call-by-value) and $\mathbf{n}$ (for call-by-name), which resolves the fundamental critical pair on a per-command basis. So in the above example, we could write the call-by-value choice as

$$\langle \mu\alpha.c_1 | \mathbf{v} | \tilde{\mu}x.c_2 \rangle \to c_1\{\tilde{\mu}x.c_2/\alpha\}$$

and the call-by-name choice as

$$\langle \mu\alpha.c_1 | \mathbf{n} | \tilde{\mu}x.c_2 \rangle \to c_2\{\mu\alpha.c_1/x\}.$$

Unfortunately, just marking commands is not enough, as it only pushes the issue of the critical pair one step away. The problem is that we could *lie* about what a variable or co-variable denotes by using it in a context that violates the contract of its binding. For example, the same critical pair is simulated as follows:

$$\langle \mu\alpha.c_1 | \mathbf{v} | \tilde{\mu}y.c_2 \rangle \leftarrow \langle \mu\alpha.c_1 | \mathbf{n} | \tilde{\mu}x.\, \langle x | \mathbf{v} | \tilde{\mu}y.c_2 \rangle \rangle \to \langle \mu\alpha.c_1 | \mathbf{n} | \tilde{\mu}x.c_2\{x/y\} \rangle\,.$$

By reducing the top redex and plugging in the computation $\mu\alpha.c_1$ for the $\mathbf{n}$ variable x, on the left we end up with a $\mathbf{v}$ command that will prioritize

the term. But by instead performing the inner redex, we end up with the equivalent **n** command that will prioritize the co-term.

So a multi-discipline sequent calculus cannot just annotate commands, but must ensure that the chosen discipline of variables and co-variables remains consistent throughout their lifetime. To make this choice apparent in the syntax, variables and co-variables must have a statically-inferable discipline which we accomplish with annotations, *e.g.,* $x^{\mathbf{v}}$ and $\alpha^{\mathbf{n}}$. Furthermore, terms and co-terms in general also much have a statically-inferable discipline, since it is sometimes necessary to introduce a new binding during reduction. For example, recall the second rule of the call-by-value abstract machine in Section 2, which corresponds to naming the argument of a function with a $\tilde{\mu}$-abstraction. This naming step is necessary to avoid getting stuck during a call-by-value function call: call-by-value β reduction does not apply to $\langle \lambda x.v \| v' \bullet e \rangle$ when v' is not a value. This is done by *lifting* v' out of the call-stack (Downen and Ariola, 2014):

$$\langle \lambda x^{\mathbf{v}}.v \| v' \bullet e \rangle \to \langle \lambda x^{\mathbf{v}}.v \| \tilde{\mu} x.\langle v' \| \tilde{\mu} y.\langle x \| y \bullet e \rangle \rangle \rangle.$$

However, to annotate α and y above, we would need to know what the intended disciplines of $\lambda x^{\mathbf{v}}.v$ and e are.

4. A Parametric, Multi-Discipline Sequent Calculus

We now formalize the core calculus for studying multi-discipline reduction in the presence of control. For simplicity we limit to a few key type formers: functions and parametric polymorphism. These features are found in most real functional programming languages, are enough both to write a variety of interesting programs, and expose the main challenges faced in strong normalization proofs.

4.1. Syntax

As in the abstract machine language of Section 2, we will build on top of a calculus comprised of terms ("producers" v), co-terms ("consumers" e), and commands ("executables" c) as shown in Fig. 1. The first thing to notice is a change of syntax for functions. Instead of λ-abstractions, functions are written by *pattern-matching* on their context: a call-stack of the form $x \bullet \alpha$. This change of notation is syntactic in nature—note that $\lambda x.v$ is equivalent to $\mu[x \,.\, \alpha].\langle v \| \alpha \rangle$—which helps to emphasize the role of functions as responders

$$
\begin{aligned}
A, B \in \mathit{Type} &::= a \mid A \to B \mid \forall a.A \\
v \in \mathit{Term} &::= x \mid \mu\alpha.c \mid \mu[x \bullet \alpha].c \mid \mu[a \bullet \alpha].c \\
e \in \mathit{CoTerm} &::= \alpha \mid \tilde{\mu}x.c \mid v \bullet e \mid A \bullet e \\
c \in \mathit{Command} &::= \langle v \| e \rangle
\end{aligned}
$$

Figure 1: Syntax of an undisciplined, polymorphic sequent calculus.

$$
\begin{aligned}
\mathbf{r}, \mathbf{s}, \mathbf{t} \in \mathit{Kind} &::= \mathbf{n} \mid \mathbf{v} \mid \ldots \qquad \boldsymbol{x} ::= x^{\mathbf{s}} \qquad \boldsymbol{\alpha} ::= \alpha^{\mathbf{s}} \qquad \boldsymbol{a} ::= a^{\mathbf{s}} \\
A_{\mathbf{s}}, B_{\mathbf{s}} \in \mathit{Type}_{\mathbf{s}} &::= a^{\mathbf{s}} \mid A \xrightarrow{\mathbf{s}} B \mid \forall^{\mathbf{s}}\boldsymbol{a}.A \qquad\qquad A, B \in \mathit{Type} ::= A_{\mathbf{s}} \\
v_{\mathbf{s}} \in \mathit{Term}_{\mathbf{s}} &::= x^{\mathbf{s}} \mid \mu\alpha^{\mathbf{s}}{:}c \mid \mu[\boldsymbol{x} \mathbin{⓼} \boldsymbol{\alpha}].c \mid \mu[\boldsymbol{a} \mathbin{⓼} \boldsymbol{\alpha}].c \\
e_{\mathbf{s}} \in \mathit{CoTerm}_{\mathbf{s}} &::= \alpha^{\mathbf{s}} \mid \tilde{\mu}x^{\mathbf{s}}{:}c \mid v \mathbin{⓼} e \mid A \mathbin{⓼} e \\
c \in \mathit{Command} &::= \langle v_{\mathbf{s}} \| e_{\mathbf{s}} \rangle \qquad v \in \mathit{Term} ::= v_{\mathbf{s}} \qquad e \in \mathit{CoTerm} ::= e_{\mathbf{s}}
\end{aligned}
$$

Figure 2: Syntax of a multi-discipline, polymorphic sequent calculus.

to call-stacks. As in system F, polymorphism is expressed in terms of type abstraction and specialization. Note that these constructs are analogous to functions, except that the parameter is a type, not a value.

However, we are not just interested in representing a single, language-wide evaluation mechanism, but instead a program-specific choice of discipline. And as discussed previously in Section 3, it is too easy in the raw syntax to mislead about this choice. Therefore, the full syntax of our calculus has terms and co-terms that are explicitly divided by their discipline, notated by a finite collection of symbols denoted by the metavariable $\mathbf{s}$, so that $v_{\mathbf{s}}$ produces an $\mathbf{s}$ value and $e_{\mathbf{s}}$ consumes an $\mathbf{s}$ value. This distinction shows up explicitly in annotations on variables and co-variables, where $x^{\mathbf{s}}$ is a member of (only) $\mathit{Term}_{\mathbf{s}}$ and similarly $\alpha^{\mathbf{s}}$ is in $\mathit{CoTerm}_{\mathbf{s}}$. A bold (co-)variable denotes an annotated (co-)variable, respectively, where the annotation could be any discipline. Commands, in contrast, do not have an outwardly-visible discipline because they do not produce or consume anything, but instead are only well-formed if they have an internally-consistent discipline shared by a producer and consumer cooperating together. To ensure that *every* term and co-term belong to exactly one syntactic category $\mathit{Term}_{\mathbf{s}}$ and $\mathit{CoTerm}_{\mathbf{s}}$, the formation of and pattern matching a call stack dot is also annotated with a

$$\langle \mu\alpha^{\mathbf{s}}_{:}c \| E_{\mathbf{s}} \rangle \succ_{\mu} c\{E_{\mathbf{s}}/\alpha^{\mathbf{s}}\} \qquad \mu\alpha^{\mathbf{s}}_{:}\langle v_{\mathbf{s}} \| \alpha^{\mathbf{s}} \rangle \succ_{\eta_\mu} v_{\mathbf{s}}$$

$$\langle V_{\mathbf{s}} \| \tilde{\mu}x^{\mathbf{s}}_{:}c \rangle \succ_{\tilde{\mu}} c\{V_{\mathbf{s}}/x^{\mathbf{s}}\} \qquad \tilde{\mu}x^{\mathbf{s}}_{:}\langle x^{\mathbf{s}} \| e_{\mathbf{s}} \rangle \succ_{\eta_{\tilde{\mu}}} e_{\mathbf{s}}$$

$$\langle \mu[x^{\mathbf{t}} \bullet_{\mathbf{s}} \alpha^{\mathbf{r}}].c \| V_{\mathbf{t}} \bullet_{\mathbf{s}} E_{\mathbf{r}} \rangle \succ_{\beta\to} c\{V_{\mathbf{t}}/x^{\mathbf{t}}, E_{\mathbf{r}}/\alpha^{\mathbf{r}}\}$$

$$\langle \mu[a^{\mathbf{t}} \bullet_{\mathbf{s}} \alpha^{\mathbf{r}}].c \| A_{\mathbf{t}} \bullet_{\mathbf{s}} E_{\mathbf{r}} \rangle \succ_{\beta\forall} c\{A_{\mathbf{t}}/a^{\mathbf{t}}, E_{\mathbf{r}}/\alpha^{\mathbf{r}}\}$$

$$v_{\mathbf{t}} \bullet_{\mathbf{s}} e \succ_{\varsigma\to} \tilde{\mu}x^{\mathbf{s}}_{:}\langle v_{\mathbf{t}} \| \tilde{\mu}y^{\mathbf{t}}_{:}\langle x^{\mathbf{s}} \| y^{\mathbf{t}} \bullet_{\mathbf{s}} e \rangle \rangle \quad (\nexists V_{\mathbf{t}}, v_{\mathbf{t}} = V_{\mathbf{t}})$$

$$V \bullet_{\mathbf{s}} e_{\mathbf{r}} \succ_{\varsigma\to} \tilde{\mu}x^{\mathbf{s}}_{:}\langle \mu\beta^{\mathbf{r}}_{:}\langle x^{\mathbf{s}} \| V \bullet_{\mathbf{s}} \beta^{\mathbf{r}} \rangle \| e_{\mathbf{r}} \rangle \quad (\nexists E_{\mathbf{r}}, e_{\mathbf{r}} = E_{\mathbf{r}})$$

$$A \bullet_{\mathbf{s}} e_{\mathbf{r}} \succ_{\varsigma\forall} \tilde{\mu}x^{\mathbf{s}}_{:}\langle \mu\beta^{\mathbf{r}}_{:}\langle x^{\mathbf{s}} \| A \bullet_{\mathbf{s}} \beta^{\mathbf{r}} \rangle \| e_{\mathbf{r}} \rangle \quad (\nexists E_{\mathbf{r}}, e_{\mathbf{r}} = E_{\mathbf{r}})$$

$$\frac{c \succ c'}{C[c] \to C[c']} \qquad \frac{e \succ e'}{C[e] \to C[e']} \qquad \frac{v \succ v'}{C[v] \to C[v']}$$

Figure 3: Rewriting theory for multi-discipline, polymorphic sequent calculus.

discipline symbol. That way, it is immediately apparent that $v \bullet_{\mathbf{s}} e$ is an $\mathbf{s}$ co-term and $\mu[\boldsymbol{x} \bullet_{\mathbf{s}} \boldsymbol{\alpha}].c$ is an $\mathbf{s}$ term.

For example, a wholly call-by-value function can be written as $\mu[x^{\mathbf{v}} \bullet_{\mathbf{v}} \alpha^{\mathbf{v}}].c$ that matches a call-stack of the form $v_{\mathbf{v}} \bullet_{\mathbf{v}} e_{\mathbf{v}}$. The $\mathbf{v}$ annotating the brackets tells us the discipline used for computing the function itself, whereas the annotations on the abstracted (co-)variables tell us the discipline of the argument and result. Replacing $\mathbf{v}$ with $\mathbf{n}$ gives instead wholly call-by-name functions, but other more interesting combinations are also possible. The functions found in call-by-push-value (Levy, 2001) and polarized languages (Zeilberger, 2009) would have the form $\mu[x^{\mathbf{v}} \bullet_{\mathbf{n}} \alpha^{\mathbf{n}}].c$ and $v_{\mathbf{v}} \bullet_{\mathbf{n}} e_{\mathbf{n}}$, with a call-by-value argument and call-by-name function and result.

4.2. Parameterized Reduction Theory

The reduction theory, denoted by $\to$ shown in Fig. 3, is the *compatible closure* of the top-level reduction relation $\succ$. Here the metavariable C ranges over any context such that filling the whole with an object of the appropriate sort is well formed. Whereas $\succ$ only applies to the top of some expression, $\to$ can apply *anywhere* inside of it. Further, we use $\twoheadrightarrow$ for the *reflexive, transitive* closure of $\to$. The reduction rules in $\succ$ are given names which we write in subscript. We also use subscripts on the $\to$ rule to denote the restriction to the rules of the same name, for instance $\to_{\beta\to}$ refers to the compatible closure of the relation $\succ_{\beta\to}$. At times we will use multiple subscripts to denote collections of reductions, as in $\succ_{\beta\to,\beta\forall}$ for the union of $\succ_{\beta\to}$ and

$\succ_{\beta^\forall}$. When a relation such as $\succ$ or $\to$ is used without a subscript it refers to the union over all of the rules in Fig. 3.

The reduction theory is parameterized by a set of specific discipline symbols equipped with an associated subset of terms called *values* and co-terms called *co-values* (denoted by $V_{\mathbf{s}}$ and $E_{\mathbf{s}}$, respectively, for each discipline symbol $\mathbf{s}$). As with (co-)terms, we use the plain metavariables V and E to refer to the union of values and co-values for every $\mathbf{s}$. Disciplines are not just restrictive but also enabling in the case of the ς rules (originally due to Wadler (2003)) that lift unevaluated components out of call-stacks to be computed, so there is no "largest" reduction theory that subsumes all others.

4.3. The Values and Co-Values of Five Important Disciplines

We can illustrate the use of disciplines by giving interpretations of some specific discipline symbols: the call-by-value ($\mathbf{v}$), -name ($\mathbf{n}$), -need ($\mathbf{lv}$ for "lazy value" (Ariola et al., 2011)), -co-need ($\mathbf{ln}$ for "lazy name") and non-deterministic ($\mathbf{u}$, implemented for instance by Barbanera and Berardi (1994)) disciplines are defined by the values and co-values in Fig. 4. These five particular disciplines are chosen for illustration for different reasons. Call-by-value and call-by-name reduction are touchstones in the theory of computer science, and appear pervasively in the study of programming language semantics. Call-by-need is a very pragmatic evaluation strategy that expresses on-demand computation while preserving the correct performance characteristics of typical algorithms. In contrast, call-by-co-need and non-deterministic evaluation are more curiosities that arise from the study of logic, where co-need is the logical dual of need, and non-deterministic evaluation is the extreme edge-case that corresponds to the original cut elimination procedure for the sequent calculus due to Gentzen (1935).

4.4. A Multi-Discipline Operational Semantics

Due to the machine-like nature of the sequent calculus, the operational semantics is straightforward to define; at least for a specified choice of disciplines. The operational semantics for the above combined $\mathbf{v}$, $\mathbf{n}$, $\mathbf{lv}$, $\mathbf{ln}$, and $\mathbf{u}$ instance of the parametric, multi-discipline sequent calculus is shown in Fig. 5, and allows for applications of the reduction rules on commands inside $D_{\mathbf{l}}$ contexts from Fig. 4 and on the co-term of a command to perform ς-reduction. In other words, standard reduction can usually proceed at the top of a command except in the case where there are delayed lazy $\mathbf{lv}$ or $\mathbf{ln}$ bindings, in which case standard reduction computes under the binders.

$$\begin{array}{rl}
\mathbf{r},\mathbf{s},\mathbf{t} ::= & \mathbf{n} \mid \mathbf{v} \mid \mathbf{lv} \mid \mathbf{ln} \mid \mathbf{u} \\
V ::= & V_{\mathbf{s}} \\
V_{\mathbf{u}} ::= & v_{\mathbf{u}} \\
V_{\mathbf{v}} ::= & x^{\mathbf{v}} \mid \mu[\boldsymbol{x} \textcircled{\mathbf{v}} \boldsymbol{\alpha}].c \mid \mu[\boldsymbol{a} \textcircled{\mathbf{v}} \boldsymbol{\alpha}].c \\
V_{\mathbf{n}} ::= & v_{\mathbf{n}} \\
V_{\mathbf{lv}} ::= & x^{\mathbf{lv}} \mid \mu[\boldsymbol{x} \textcircled{\mathbf{lv}} \boldsymbol{\alpha}].c \mid \mu[\boldsymbol{a} \textcircled{\mathbf{lv}} \boldsymbol{\alpha}].c \\
E_{\mathbf{lv}} ::= & \alpha^{\mathbf{lv}} \mid \tilde{\mu}x^{\mathbf{lv}}\dot{.}D_{\mathbf{l}}[\langle x^{\mathbf{lv}} \| E_{\mathbf{lv}} \rangle] \mid V \textcircled{\mathbf{lv}} E \mid A \textcircled{\mathbf{lv}} E \\
V_{\mathbf{ln}} ::= & x^{\mathbf{ln}} \mid \mu[\boldsymbol{x} \textcircled{\mathbf{ln}} \boldsymbol{\alpha}].c \mid \mu[\boldsymbol{a} \textcircled{\mathbf{ln}} \boldsymbol{\alpha}].c \mid \mu\alpha^{\mathbf{ln}}\dot{.}D_{\mathbf{l}}[\langle V_{\mathbf{ln}} \| \alpha^{\mathbf{ln}} \rangle] \\
E_{\mathbf{ln}} ::= & \alpha^{\mathbf{ln}} \mid V \textcircled{\mathbf{ln}} E \mid A \textcircled{\mathbf{ln}} E \\
D_{\mathbf{l}} ::= & \Box \mid \langle v_{\mathbf{lv}} \| \tilde{\mu}y^{\mathbf{lv}}\dot{.}D_{\mathbf{l}} \rangle \mid \langle \mu\alpha^{\mathbf{ln}}\dot{.}D_{\mathbf{l}} \| e_{\mathbf{ln}} \rangle
\end{array}$$

$$E ::= E_{\mathbf{s}}$$
$$E_{\mathbf{u}} ::= e_{\mathbf{u}}$$
$$E_{\mathbf{v}} ::= e_{\mathbf{v}}$$
$$E_{\mathbf{n}} ::= \alpha^{\mathbf{n}} \mid V \textcircled{\mathbf{n}} E \mid A \textcircled{\mathbf{n}} E$$

Figure 4: (Co-)values for call-by-name (**n**), call-by-value (**v**), call-by-need (**lv**), call-by-co-need (**ln**) and nondeterministic (**u**) disciplines.

$$\frac{c \succ_{\mu,\tilde{\mu},\beta} c'}{c \mapsto_{\mu,\tilde{\mu},\beta} c'} \qquad \frac{e \succ_{\varsigma} e'}{\langle V \| e \rangle \mapsto_{\varsigma} \langle V \| e' \rangle} \qquad \frac{e_{\mathbf{lv}} \succ_{\varsigma^{\mathbf{lv}}} e'_{\mathbf{lv}}}{\langle v_{\mathbf{lv}} \| e_{\mathbf{lv}} \rangle \mapsto_{\varsigma_{\mathbf{lv}}} \langle v_{\mathbf{lv}} \| e'_{\mathbf{lv}} \rangle} \qquad \frac{c \mapsto c'}{D_{\mathbf{l}}[c] \mapsto D_{\mathbf{l}}[c']}$$

Figure 5: A multi-discipline operational semantics for call-by-name, call-by-value, call-by-need, call-by-co-need, and nondeterministic evaluation.

This helps explain how **lv** and **ln** implement laziness. For example, when we have a command of the form $\langle \mu\alpha^{\mathbf{lv}}.c_1 \| \tilde{\mu}x^{\mathbf{lv}}.c_2 \rangle$, the standard thing to do is to delay $\mu\alpha^{\mathbf{lv}}.c_1$ and work on c_2. Then only once c_2 has been reduced down to $D_{\mathbf{l}}[\langle x^{\mathbf{lv}} \| E_{\mathbf{lv}} \rangle]$ is x demanded, since only then is $\tilde{\mu}x^{\mathbf{lv}}.D_{\mathbf{l}}[\langle x^{\mathbf{lv}} \| E_{\mathbf{lv}} \rangle]$ a co-value, and only a co-value can be substituted for $\alpha^{\mathbf{lv}}$ in order to run c_1 to get the result of the μ-abstraction. Thus, we see that the standard reduction in **lv** reduces a co-term until it corresponds to a forcing context, before switching to reduce the term which is now demanded.

4.5. Typing

As a generalization of polarity, types belong to one of several *kinds*, each associated with a discipline. The kind of a type is specified by its top constructor, for example $A \xrightarrow{\mathbf{v}} B$ and $A \xrightarrow{\mathbf{lv}} B$ are types of call-by-value and call-by-need, respectively. Type variables range over a specific kind denoting

$$\frac{}{\Gamma, \boldsymbol{x} : A \vdash_\Theta \boldsymbol{x} : A \mid \Delta}\ \mathit{VarR} \qquad \frac{}{\Gamma \mid \boldsymbol{\alpha} : A \vdash_\Theta \boldsymbol{\alpha} : A, \Delta}\ \mathit{VarL}$$

$$\frac{c : (\Gamma \vdash_\Theta \boldsymbol{\alpha} : A, \Delta)}{\Gamma \vdash_\Theta \mu\boldsymbol{\alpha}.c : A \mid \Delta}\ \mathit{ActR} \qquad \frac{c : (\Gamma, \boldsymbol{x} : A \vdash_\Theta \Delta)}{\Gamma \mid \tilde{\mu}\boldsymbol{x}.c : A \vdash_\Theta \Delta}\ \mathit{ActL}$$

$$\frac{\Gamma \vdash_\Theta v : A \mid \Delta \quad \Gamma \mid e : A \vdash_\Theta \Delta}{\langle v \| e \rangle : (\Gamma \vdash_\Theta \Delta)}\ \mathit{Cut}$$

$$\frac{\Gamma \vdash_\Theta v : A \mid \Delta \quad \Gamma \mid e : B \vdash_\Theta \Delta}{\Gamma \mid v \mathbin{⧟} e : A \xrightarrow{\mathbf{s}} B \vdash_\Theta \Delta} \to L \qquad \frac{c : (\Gamma, \boldsymbol{x} : A \vdash_\Theta \boldsymbol{\alpha} : B, \Delta)}{\Gamma \vdash_\Theta \mu[\boldsymbol{x} \mathbin{⧟} \boldsymbol{\alpha}].c : A \xrightarrow{\mathbf{s}} B \mid \Delta} \to R$$

$$\frac{\Gamma \mid e : B\{A_{\mathbf{t}}/a^{\mathbf{t}}\} \vdash_\Theta \Delta}{\Gamma \mid A_{\mathbf{t}} \mathbin{⧟} e : \forall^{\mathbf{s}} a^{\mathbf{t}}.B \vdash_\Theta \Delta} \forall L \qquad \frac{c : (\Gamma \vdash_{\Theta, \boldsymbol{a}} \boldsymbol{\alpha} : B, \Delta)}{\Gamma \vdash_\Theta \mu[\boldsymbol{a} \mathbin{⧟} \boldsymbol{\alpha}].c : \forall^{\mathbf{s}} \boldsymbol{a}.B \mid \Delta} \forall R$$

Figure 6: Type system for the multi-discipline, polymorphic sequent calculus.

the discipline of (co-)terms they specify, and the polymorphic quantifier $\forall^{\mathbf{s}}$ must choose a specific kind of type to abstract over.

The typing rules for the calculus are given in Fig. 6. There are some criteria for when sequents are well formed: (1) identifiers (a, x, α) in Θ, Γ, and Δ are all unique, (2) the disciplines of (co-)variables must match that of their type, as in $x^{\mathbf{s}} : A_{\mathbf{s}}$ and $\alpha^{\mathbf{s}} : A_{\mathbf{s}}$, and (3) in the sequent $\Gamma \vdash_\Theta v : A \mid \Delta$, all the free type variables of Γ, Δ, v, and A are included in Θ, and similarly for $\Gamma \mid e : A \vdash_\Theta \Delta$ and $c : (\Gamma \vdash_\Theta \Delta)$. Only derivations where all sequents are well formed are considered proofs. Note that this imposes the standard side-condition on the right $\forall$ rule that the abstracted type variable in the premise is not free in the conclusion (*i.e.,* that $a^{\mathbf{t}} \notin FV(\Gamma \vdash_\Theta \Delta)$). Well-formedness also ensures that in the cut and the left rule for $\forall$, the free variables of the cut and instantiated type are contained in Θ.

4.6. Type Safety

In order to properly state type safety, we also need to say when a command is done computing. However, in the calculus we've considered here, there is no basic type, like booleans or numbers for example, for which we can observe the result. In their stead, we can use type variables as our observables since they *might* stand in for a basic type. Note that type variables can only possibly classify generic (co-)terms like (co-)variables or μ- and $\tilde{\mu}$-abstractions, which means that we only get to observe free (co-)variables

that stand in for results. While in the λ-calculus we usually only run closed terms, this change of attitude goes along with the basic fact of logical consistency in the sequent calculus: there is no such thing as a well-typed, closed command. Therefore, we can say that a combined **n-v-lv-ln-u**-disciplined sequent calculus command c is *done* exactly when $c = D_{\mathbf{l}}[\langle \boldsymbol{x} \| \boldsymbol{\alpha} \rangle]$ and neither $\boldsymbol{x}$ nor $\boldsymbol{\alpha}$ are bound by the context $D_{\mathbf{l}}$. We can now sketch out type safety in terms of progress and preservation.

Theorem 1 (Progress)**.** *Given a multi-discipline sequent calculus command* $c : (\boldsymbol{x} : \boldsymbol{a}, \dots \vdash_\Theta \boldsymbol{\alpha} : \boldsymbol{b}, \dots)$*, either* c *is done or* $c \mapsto c'$ *for some* c'*.*

Proof. To begin, we generalize the set of done commands to include all *paused* commands of the form $D_{\mathbf{l}}[\langle V \| E \rangle]$ such at least one of V or E is a (co-)variable, none of which are bound by $D_{\mathbf{l}}$. Progress is then a special case of the three following lemmas

1. *If* $c : (\Gamma \vdash_\Theta \Delta)$ *then either* c *is paused or* $c \mapsto$*.*
2. *If* $\Gamma \mid e : A_{\mathbf{lv}} \vdash_\Theta \Delta$ *then either* e *is a co-value of* $\mathbf{lv}$*,* $e \succ_\varsigma$*, or* $e = \tilde{\mu}\boldsymbol{x}^{\mathbf{lv}}.c$ *such that* c *is paused or* $c \mapsto$*.*
3. *If* $\Gamma \vdash_\Theta v : A_{\mathbf{ln}} \mid \Delta$ *then either* v *is a value of* $\mathbf{ln}$*, or* $v = \mu\alpha^{\mathbf{ln}}.c$ *such that* c *is paused or* $c \mapsto$*.*

because every paused command of type $c : (\boldsymbol{x} : \boldsymbol{a}, \dots \vdash_\Theta \boldsymbol{\alpha} : \boldsymbol{b}, \dots)$ must be done. These lemmas can be fully shown by mutual induction the given typing derivation. Briefly here, consider the possible counter-examples where we might have a typed command c that is not paused and yet has no standard reduction, $c \not\mapsto$. The three main potential causes for getting stuck are:

- Ill-typed pairs: where we have one of the many configurations that are stuck on a type error. These consist of confusing functions with polymorphism, as in $\langle \mu[\boldsymbol{x} \, \textcircled{\scriptsize s} \, \boldsymbol{\beta}].c \| A \, \textcircled{\scriptsize s} \, E \rangle$ and also where there is a discipline miss-matched β-reduction in a nonetheless well-disciplined command, as in $\langle \mu[\boldsymbol{x} \, \textcircled{\scriptsize s} \, \boldsymbol{\beta}].c \| V \, \textcircled{\scriptsize r} \, E \rangle$ when $\mathbf{s} \neq \mathbf{r}$. Both of these cases are ruled out by type checking—either because the type $A_1 \xrightarrow{\mathbf{s}} A_2$ can never match with $\forall^{\mathbf{s}}\boldsymbol{a}.B$, or because $A \xrightarrow{\mathbf{s}} B$ can only match with $A \xrightarrow{\mathbf{r}} B$ when $\mathbf{s} = \mathbf{r}$—so they cannot happen.

- Insufficient substitution: where we have a well-typed command that almost has a standard μ- or $\tilde{\mu}$-redex, but the bound (co-)term is not yet a (co-)value. For example, we could have a command of the form $c = \langle v_{\mathbf{lv}} \| \tilde{\mu}\boldsymbol{x}^{\mathbf{lv}}.c' \rangle$ where $v_{\mathbf{lv}}$ is not a value. In this case, by the second lemma above, it must be that $\tilde{\mu}\boldsymbol{x}^{\mathbf{lv}}.c'$ is a value, in which case $c \mapsto_{\mu}$ since the non-value $v_{\mathbf{lv}}$ must be a μ-abstraction, or $c' \mapsto c''$, in which case $\langle v_{\mathbf{lv}} \| \tilde{\mu}\boldsymbol{x}^{\mathbf{lv}}.c' \rangle \mapsto \langle v_{\mathbf{lv}} \| \tilde{\mu}\boldsymbol{x}^{\mathbf{lv}}.c'' \rangle$.

- Insufficient lifting: where we have a well-typed command that almost has a standard β-redex, except the components of the call stack is not a co-value. For example, we could have a command of the form $c = \langle v \| v' \mathbin{\textcircled{\scriptsize s}} e' \rangle$ where one of v' is not a value or e' is not a co-value. If v is a value, then $c \mapsto_{\varsigma\rightarrow}$ for any $\mathbf{s}$. If v is not, the step depends on $\mathbf{s}$:

  - $\mathbf{s} = \mathbf{n}$: is impossible, since every term of $\mathbf{n}$ is a value of $\mathbf{n}$.
  - $\mathbf{s} = \mathbf{v}$: v must be a μ-abstraction, so that $c \mapsto_{\mu}$.
  - $\mathbf{s} = \mathbf{lv}$: $c \mapsto_{\varsigma\rightarrow}$ anyway even though v is not a value of $\mathbf{lv}$.
  - $\mathbf{s} = \mathbf{ln}$: by the third lemma above, v must be a μ-abstraction $\mu\boldsymbol{\alpha}^{\mathbf{ln}}.c'$ such that $c' \mapsto c''$ or c' is paused. If $c' \mapsto c''$, then the whole command $\langle \mu\boldsymbol{\alpha}^{\mathbf{ln}}.c' \| v' \mathbin{\textcircled{\scriptsize s}} e' \rangle \mapsto \langle \mu\boldsymbol{\alpha}^{\mathbf{ln}}.c'' \| v' \mathbin{\textcircled{\scriptsize s}} e' \rangle$. If c' is paused, the only non-value form of v is when $v = \mu\boldsymbol{\alpha}^{\mathbf{ln}}.D_{\mathbf{l}}[\langle V \| E \rangle]$ where $E \neq \boldsymbol{\alpha}$. In this case, the whole command c would be paused.

Therefore, there is no way to find a well-typed, unpaused command that cannot take a step. □

Theorem 2 (Preservation)**.** *In the multi-discipline sequent calculus*

1) if $c : (\Gamma \vdash_{\Theta} \Delta)$ *and* $c \to c'$ *then* $c' : (\Gamma \vdash_{\Theta} \Delta)$,

2) if $\Gamma \vdash_{\Theta} v : A \mid \Delta$ *and* $v \to v'$ *then* $\Gamma \vdash_{\Theta} v' : A \mid \Delta$, *and*

3) if $\Gamma \mid e : A \vdash_{\Theta} \Delta$ *and* $e \to e'$ *then* $\Gamma \mid e' : A \vdash_{\Theta} \Delta$.

Proof. By induction on the given typing derivation and cases on the top-level reduction rules where they occur. This relies on three standard substitution lemmas that state that substituting (1) a value for a variable of the same type, (2) a co-value for a co-variable of the same type, and (3) a type for a type variable of the same kind preserves typing. □

$$\frac{c \succ_{\mu,\tilde{\mu},\beta} c'}{c \rightsquigarrow_{\mu,\tilde{\mu},\beta} c'} \qquad\qquad \frac{e \succ_{\varsigma} e'}{\langle V \| e \rangle \rightsquigarrow_{\varsigma} \langle V \| e' \rangle}$$

Figure 7: The top-level reduction relation.

5. Properties of Multi-Discipline Rewriting

In order to build a proof of strong normalization for our classical sequent calculus, we will need to know some facts about disciplines and their associated reduction theories. In summary, we will need to (1) identify and classify the set of "important" reductions (here called "top" reductions) on which to base our model of normalization, (2) show that top reductions are indeed necessary for computation, in that they can always be performed first before any other "internal" reductions, and (3) characterize what general properties of disciplines are needed for them to be "admissible" for use in strongly normalizing reduction.

5.1. Charged Top Reduction

A common aspect of proving normalization is to identify a subset of reductions that are important to check for the purpose of normalization: if those reductions can lead to an infinite reduction sequence then all hope is lost. Usually in the λ-calculus, these important reductions are the standard reductions that make up an operational semantics. But since we are working in the sequent calculus, we already have a notion of "main" reduction that is immediately apparent in the syntax: the reductions that occur at the "top" of a command. We define *top reduction* ($\rightsquigarrow$) on commands as shown in Fig. 7, which applies a primary rewriting rule either to the command itself (in the case of a μ, $\tilde{\mu}$, or β reduction), or to its immediate sub-co-term (in the case of a ς reduction). We use $\rightsquigarrow^?$ to denote the reflexive closure of $\rightsquigarrow$.

Notice that top reduction in the sequent calculus does not necessarily correspond to the standard reductions of an operational semantics. For example, consider again the operational semantics for the five important disciplines that was given in Fig. 5. The standard reduction relation includes all the reductions allowed in top reduction, but it *also* allows for a more general application of a call-by-need $\varsigma_{\mathbf{lv}}$ reduction, as well as reductions inside of a delayed $D_{\mathbf{l}}$ context. So every top reduction is a standard reduction, but not vice versa. As a consequence, top reduction is type safe (*i.e.,* it

doesn't get stuck on well-typed commands) for disciplines like call-by-name (**n**), call-by-value (**v**), and non-deterministic evaluation (**u**), but not for ones like call-by-need (**lv**), and -by-co-need (**ln**). However, the advantage of considering top reduction is that it's definition does not depend deeply on the chosen disciplines: it is just reduction of the top-level command or it's immediate sub-expressions. And as it turns out, while top reduction might not be enough for type-safe computation, it *is* enough to characterize the important steps of strong normalization for any substitution discipline.

Top reduction is useful for analyzing strong normalization because it reduces the number of counter-examples of normalization we need to consider. In particular, given a non-normalizing command made up of a strongly normalizing term and co-term, it must be possible to find some reduction sequence that passes through a top reduction.

Lemma 1. *If v and e are strongly normalizing but $\langle v \| e \rangle$ is not strongly normalizing, then there are some v', e', and c' such that $v \twoheadrightarrow v'$, $e \twoheadrightarrow e'$, $\langle v \| e \rangle \twoheadrightarrow \langle v' \| e' \rangle \mapsto c'$, and c' is not strongly normalizing.*

Proof. This lemma relies on the fact that every reduction on commands is either a top reduction, or internal to one side of the command. That is, if $\langle v \| e \rangle \to c'$, then one of the following must be true:

(i) $c \rightsquigarrow c'$,

(ii) $v \to v'$ for some v' and $c' = \langle v' \| e \rangle$, or

(iii) $e \to e'$ for some e' and $c' = \langle v \| e' \rangle$.

So consider an infinite reduction sequence starting with $\langle v \| e \rangle$ ($\langle v \| e \rangle \to c_1 \to c_2 \to \dots$) where each reduction step must be one of the above three cases. It is impossible that there is an infinite number of reductions of the form (ii) or (iii), because both v and e are strongly normalizing. Therefore, there must be a step of case (i) in the infinite reduction sequence, which can be found by induction on the longest number of reductions starting from v and e:

- If $\langle v \| e \rangle \rightsquigarrow c_1$, then the result holds by reflexivity.
- If $v \to v'$ and $c_1 = \langle v' \| e \rangle$ then the result holds by the inductive hypothesis, where the longest number of reductions starting from v' is strictly less than that of v.

- If $e \to e'$ and $c_1 = \langle v \| e' \rangle$ then the result holds analogously to the previous case. □

In order to analyze the behavior of top reduction, we will assign a *charge*—positive (+), negative (−), or neutral (0)—to individual reduction rules which characterize their important semantic properties.

Definition 1 (Charged Reduction)**.** We assign the following charges to the top-level reduction relation (for each discipline $\mathbf{s}$):

- A *positive* reduction uses the rule $\mu_{\mathbf{s}}$, wherein the term is solely responsible for the result of the reduction. We write this charge as + which denotes $\mu_{\mathbf{s}}$ reduction only.
- A *negative* reduction uses either the rule $\mu_{\mathbf{s}}$, $\varsigma_{\mathbf{s}}^{\to}$, or $\varsigma_{\mathbf{s}}^{\forall}$, wherein the co-term is solely responsible for the result of the reduction. We write this charge as − which denotes the union of $\tilde{\mu}_{\mathbf{s}}\varsigma_{\mathbf{s}}^{\to}\varsigma_{\mathbf{s}}^{\forall}$ reduction.
- A *neutral* reduction uses either the rule $\beta_{\mathbf{s}}^{\to}$ or $\beta_{\mathbf{s}}^{\forall}$, wherein both the term and co-term must cooperate with one another to take a step. We write this charge as 0 which denotes the union of $\beta_{\mathbf{s}}^{\to}\beta_{\mathbf{s}}^{\forall}$ reduction.

A charged reduction step is just a standard reduction limited to a subset of reduction rules. So $\rightsquigarrow_+$ denotes $\rightsquigarrow_\mu$ and $\rightsquigarrow_-$ denotes $\rightsquigarrow_{\tilde{\mu}\varsigma^{\to}\varsigma^{\forall}}$. Combinations of charges are just the union of the underlying rules, so $\rightsquigarrow_{+0}$ denotes $\rightsquigarrow_{\mu\beta^{\to}\beta^{\forall}}$ and $\rightsquigarrow_{+-0}$ (which is also written as just $\rightsquigarrow$) denotes $\rightsquigarrow_{\mu\tilde{\mu}\beta^{\to}\varsigma^{\to}}$.

Intuitively, neutrally charged reductions require cooperation of the term and co-term. Positively charged reductions allow the term to take over control of the command in order to simplify itself. And dually, negatively charged reductions are allow the co-term to take over control of the command.

By classifying the charge of a top reduction, we can make some statements about what behavior is (or is not) possible independently of any specific substitution discipline. In particular, certain combinations of charges lead to a guaranteed deterministic top reduction relation no matter the definition of values and co-values.

Property 1. *When taken on their own, each of the charged head reduction relations, $\rightsquigarrow_0$, $\rightsquigarrow_+$, and $\rightsquigarrow_-$, are deterministic regardless of the discipline (*e.g., *$c_1 \leftsquigarrow_0 c \rightsquigarrow_0 c_2$ implies that $c_1 = c_2$), as are the combined $\rightsquigarrow_{+0}$ and*

$\rightsquigarrow_{-0}$ *top reduction relations. However, the combined* $\rightsquigarrow_{+-}$ *top reduction relation may be non-deterministic depending on the discipline (*i.e., *it may be that* $c_1 \leftsquigarrow_{+} c \rightsquigarrow_{-} c_2$ *and* $c_1 \neq c_2$*).*

Therefore, the question of the possibility of "essential" non-determinism which goes comes down to the interaction between positive and negative reductions.

Definition 2. A discipline $\mathbf{s}$ is *deterministic* if the $\rightsquigarrow_{+-}$ reduction relation is deterministic on all commands of the form $\langle v_{\mathbf{s}} \| e_{\mathbf{s}} \rangle$.

Note that of course the general reduction relation ($c \to c'$) is not deterministic, because any sub-expression may be reduced. However, general reduction is still *confluent* for deterministic disciplines (Downen, 2017). In contrast, reduction may be *non-confluent* for non-deterministic disciplines, which is the essential difficulty posed by the critical pair between positively- and negatively-charged reductions. For example, the specific $\mathbf{u}$ discipline leads to non-confluent reduction, and is thus non-deterministic, but the other four of the five important disciplines are deterministic.

Property 2. *The specific* $\mathbf{v}$, $\mathbf{n}$, $\mathbf{lv}$, *and* $\mathbf{ln}$ *disciplines are all deterministic. The* $\mathbf{u}$ *discipline is non-deterministic.*

5.2. Admissible Disciplines

Our proof of strong normalization is parameterized by a collection of discipline symbols and their interpretation. However, not every set of values and co-values make for a sensible substitution discipline. In particular, there are two important properties—being stable and focalizing—that are required of *admissible* disciplines that ensure strong normalization.

Definition 3. A discipline is

(i) *stable* when (co-)values are closed under reduction and substitution,

(ii) *focalizing* when *at least* all (1) variables, $\mu[\boldsymbol{x} \, \textcircled{\scriptsize s} \, \boldsymbol{\alpha}].c$, and $\mu[\boldsymbol{a} \, \textcircled{\scriptsize s} \, \boldsymbol{\alpha}].c$ are values, and (2) co-variables, $V \, \textcircled{\scriptsize s} \, E$, and $A \, \textcircled{\scriptsize s} \, E$, are co-values, and

(iii) *admissible* when it is stable and focalizing.

The focalizing property of disciplines corresponds to focalization in logic (Curien and Munch-Maccagnoni, 2010)—each criterion comes from a focused inference rule for typing (co-)values. As a special, each of the five specific disciplines from Section 4 are all examples of admissible disciplines.

Property 3. *The specific* **n**, **v**, **lv**, **ln**, *and* **u** *disciplines are collectively admissible.*

Proof. Let us first introduce a notion of a *hygienic decomposition* of a command c into $D_{\mathbf{l}}[c']$ as one where c' is of the form $\langle V_{\mathbf{ln}} \| \alpha^{\mathbf{ln}} \rangle$ with $\alpha^{\mathbf{ln}}$ is not bound in c or $\langle x^{\mathbf{lv}} \| E_{\mathbf{lv}} \rangle$ where $x^{\mathbf{lv}}$ is not bound in c. Because we assume the Barendregt convention, we can verify that hygienic decompositions are, when they exist, unique by induction on the size of c.

For all the disciplines, focalization is immediate and stability follows immediately by induction for all but call-by-need and its dual. To show stability for call-by-need and its dual we use the fact about hygienic decompositions above.

Observe that values and co-values are closed under substitution, which follows from the fact that $D_{\mathbf{l}}$ contexts are also closed under substitution of (co-)values for (co-)values. All that remains is showing that the reduction rules do not turn (co-)values into non-(co-)values. To do so, observe all at once the four facts that:

(i) if $E \to e'$ then e' is a co-value,

(ii) if $V \to v'$ then v' is a value,

(iii) if $D_{\mathbf{l}}[\langle x^{\mathbf{lv}} \| E_{\mathbf{lv}} \rangle]$ is a hygienic decomposition and $D_{\mathbf{l}}[\langle x^{\mathbf{lv}} \| E_{\mathbf{lv}} \rangle] \to c$ then there is a hygienic decomposition $c = D'_{\mathbf{l}}[\langle x^{\mathbf{lv}} \| E'_{\mathbf{lv}} \rangle]$, and

(iv) if $D_{\mathbf{l}}[\langle V_{\mathbf{ln}} \| \alpha^{\mathbf{ln}} \rangle]$ is a hygienic decomposition and $D_{\mathbf{l}}[\langle V_{\mathbf{ln}} \| \alpha^{\mathbf{ln}} \rangle] \to c$ then there is a hygienic decomposition $c = D'_{\mathbf{l}}[\langle V'_{\mathbf{ln}} \| \alpha^{\mathbf{ln}} \rangle]$.

follows by mutual induction on the syntax of commands and (co-)terms. The interesting case of fact 1 are the ς rules, but ς only applies when one of the two components of a call stack are not (co-)values, and so either ς cannot apply, the entire call stack was not originally a (co-)value of **n**, **lv**, or **ln**, or the call stack is trivially a (co-)value of **v** or **u** before and after reduction. The interesting case of facts 3 and 4 is

$$\begin{aligned} \langle V \| \tilde{\mu} y^{\mathbf{lv}}.D_{\mathbf{l}}[\langle x^{\mathbf{lv}} \| E \rangle] \rangle &\to D_{\mathbf{l}}[\langle x^{\mathbf{lv}} \| E \rangle]\{y^{\mathbf{lv}}/V\} \\ &= D_{\mathbf{l}}\{y^{\mathbf{lv}}/V\}[\langle x^{\mathbf{lv}} \| E\{y^{\mathbf{lv}}/V\} \rangle] \end{aligned}$$

(and its dual) where $D_{\mathbf{l}}[\langle x^{\mathbf{lv}} \| E \rangle]$ is a hygienic decomposition, which follows by the fact that $D_{\mathbf{l}}$ contexts are closed under substitution and because $x^{\mathbf{lv}} \neq y^{\mathbf{lv}}$ by the definition of hygienic decomposition. □

These are the only facts we use about the collection of substitution disciplines in order to prove strong normalization: any other properties that are needed for normalization can be derived from stability and focalization above. For example, focalization tells us something about the top reduction relation: every top-reducible command is made up of a value (for positive reduction), a co-value (for negative reduction), or both (for neutral reduction).

Property 4. *For any focalizing discipline* $\mathbf{s}$*,*

(i) if $\langle v_{\mathbf{s}} \| e_{\mathbf{s}} \rangle \rightsquigarrow_+ c$ *then* $e_{\mathbf{s}}$ *is a co-value of* $\mathbf{s}$*,*

(ii) if $\langle v_{\mathbf{s}} \| e_{\mathbf{s}} \rangle \rightsquigarrow_- c$ *then* $v_{\mathbf{s}}$ *is a value of* $\mathbf{s}$*, and*

(iii) if $\langle v_{\mathbf{s}} \| e_{\mathbf{s}} \rangle \rightsquigarrow_0 c$ *then* $v_{\mathbf{s}}$ *is a value and* $e_{\mathbf{s}}$ *is a co-value of* $\mathbf{s}$*.*

Proof. The facts about positive and negative reduction are guaranteed by the respective reduction rules. The fact about neutral reduction follows from the assumption that $\mathbf{s}$ is focalizing, which means that the call stack and abstraction taking part in a $\beta^{\rightarrow}$ or $\beta^{\forall}$ reduction must be a co-value and value, respectively. □

Our proof of strong normalization works uniformly for any collection of admissible disciplines. As we present the proof that follows, we assume some admissible disciplines have been chosen, which could include any combination of the five disciplines presented above, or some other admissible disciplines of interest.

5.3. Commuting Top and Internal Reduction

The top reductions are important in the sense that they can always be performed *first*, before any other internal reductions. This kind of property is quite useful for proving strong normalization. For example, studying strong normalization for the λ-calculus we can use standardization to help when demonstrating one of the primary lemmas: that if v' and $v\{v'/x\}$ are both strongly normalizing, then $(\lambda x.v)\ v'$ is as well. This works because, no matter what reductions might be happening internally inside v and v', we can always

rearrange the reduction sequence so that the standard reduction $(\lambda x.v)\ v' \mapsto v\{v'/x\}$ happens first, meaning all reduction sequences must normalize.

Since we will be focusing on top reduction, a subset of standard reduction, we will need to use a similar, partial standardization property about the reduction theory of the sequent calculus. This property says that top reduction commutes with all other internal reductions. With one step of an internal reduction and one top reduction (of a particular charge p), the cases for commutation of appear as follows.

Lemma 2. *For any stable discipline* $\mathbf{s}$ *and charge* $p \in \{0, +, -\}$*:*

(i) If $v_{\mathbf{s}} \to v'_{\mathbf{s}}$ *and* $\langle v_{\mathbf{s}} \| e_{\mathbf{s}} \rangle \rightsquigarrow_p c$ *then* $\langle v'_{\mathbf{s}} \| e_{\mathbf{s}} \rangle \rightsquigarrow_p^? c' \twoheadleftarrow c$ *for some* c'*. Moreover, if* $p \in \{0, -\}$ *then* $\langle v'_{\mathbf{s}} \| e_{\mathbf{s}} \rangle \rightsquigarrow_p c' \twoheadleftarrow c$*.*

(ii) If $e_{\mathbf{s}} \to e'_{\mathbf{s}}$ *and* $\langle v_{\mathbf{s}} \| e_{\mathbf{s}} \rangle \rightsquigarrow_p c$ *then* $\langle v_{\mathbf{s}} \| e'_{\mathbf{s}} \rangle \rightsquigarrow_p^? c' \twoheadleftarrow c$ *for some* c'*. Moreover, if* $p \in \{0, +\}$ *then* $\langle v_{\mathbf{s}} \| e'_{\mathbf{s}} \rangle \rightsquigarrow_p c' \twoheadleftarrow c$*.*

In pictures, any spans of these two forms can be completed the following commuting diagram:

Given $v \to v'$,

$$\begin{array}{ccc} \langle v \| e \rangle & \overset{p}{\rightsquigarrow} & c \\ \downarrow & & \vdots \\ \langle v' \| e \rangle & \overset{p}{\dashrightarrow}^? & c' \end{array}$$

and given $e \to e'$,

$$\begin{array}{ccc} \langle v \| e \rangle & \overset{p}{\rightsquigarrow} & c \\ \downarrow & & \vdots \\ \langle v \| e' \rangle & \overset{p}{\dashrightarrow}^? & c' \end{array}$$

Proof. By cases on the possible reductions. Stability of $\mathbf{s}$ is required to ensure that internal reductions do not accidentally turn a (co-)value into a non-(co-)value, thereby blocking the top reduction. Therefore, internal reductions inside a sub-expression of a top reduction commutes immediately. For example, given $V_{\mathbf{s}} \to V'_{\mathbf{s}}$ and $\langle V_{\mathbf{s}} \| \tilde{\mu} x^{\mathbf{s}} . c \rangle \rightsquigarrow_- c\{V_{\mathbf{s}}/x^{\mathbf{s}}\}$, the pair comes back together as

$$\langle V'_{\mathbf{s}} \| \tilde{\mu} x^{\mathbf{s}} . c \rangle \rightsquigarrow_- c\{V'_{\mathbf{s}}/x^{\mathbf{s}}\} \twoheadleftarrow c\{V_{\mathbf{s}}/x^{\mathbf{s}}\}$$

The remaining interesting cases occur when an internal reduction on either side of a command accidentally ends up simulating the top reduction. This can happen with the η_μ and $\eta_{\tilde{\mu}}$ reductions, which can appear as a μ or $\tilde{\mu}$ top reduction, respectively. For example, given $\mu\alpha^{\mathbf{s}} . \langle v_{\mathbf{s}} \| \alpha^{\mathbf{s}} \rangle \to_{\eta_\mu} v_{\mathbf{s}}$, we could have the following pair of identical reductions (one internal and one top):

$$\langle v_{\mathbf{s}} \| E_{\mathbf{s}} \rangle \leftarrow_{\eta_\mu} \langle \mu\alpha^{\mathbf{s}} . \langle v_{\mathbf{s}} \| \alpha^{\mathbf{s}} \rangle \| E_{\mathbf{s}} \rangle \rightsquigarrow_+ \langle v_{\mathbf{s}} \| E_{\mathbf{s}} \rangle$$

In this case, the two end up at the same command, so that $\langle v_{\mathbf{s}} \| E_{\mathbf{s}} \rangle \rightsquigarrow_{+}^{?} \langle v_{\mathbf{s}} \| E_{\mathbf{s}} \rangle \twoheadleftarrow \langle v_{\mathbf{s}} \| E_{\mathbf{s}} \rangle$ by reflexivity.

Another possibility lies with the ς reductions, which both apply to a co-term directly as well as in a top reduction, and the two possibilities coincide exactly. For example, given $v_{\mathbf{t}} \circledS e \rightarrow_{\varsigma} \tilde{\mu} x^{\mathbf{s}} \langle v_{\mathbf{t}} \| \tilde{\mu} y^{\mathbf{t}} \langle x^{\mathbf{s}} \| y^{\mathbf{s}} \circledS e \rangle \rangle$ we then we would have a similar pair of identical reductions.

The last possibility is when a (co-)term is reduced to a (co-)value, which makes a ς reduction no longer necessary (and indeed, no longer possible). For example, if we have some non-value $v_{\mathbf{t}}$ such that $v_{\mathbf{t}} \rightarrow V_{\mathbf{t}}$, there is the pair of reductions: $\langle V'_{\mathbf{s}} \| V_{\mathbf{t}} \circledS e \rangle \leftarrow \langle V'_{\mathbf{s}} \| v_{\mathbf{t}} \circledS e \rangle \rightsquigarrow_{\varsigma} \langle V'_{\mathbf{s}} \| \tilde{\mu} x^{\mathbf{s}} \langle v_{\mathbf{t}} \| \tilde{\mu} y^{\mathbf{t}} \langle x^{\mathbf{s}} \| y^{\mathbf{s}} \circledS e \rangle \rangle \rangle$. These two commands come back together as follows:

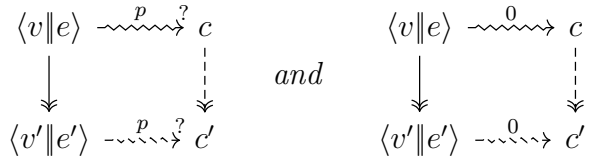
$$\langle V'_{\mathbf{s}} \| \tilde{\mu} x^{\mathbf{s}} \langle v_{\mathbf{t}} \| \tilde{\mu} y^{\mathbf{t}} \langle x^{\mathbf{s}} \| y^{\mathbf{s}} \circledS e \rangle \rangle \rangle \rightarrow \langle V'_{\mathbf{s}} \| \tilde{\mu} x^{\mathbf{s}} \langle V_{\mathbf{t}} \| \tilde{\mu} y^{\mathbf{t}} \langle x^{\mathbf{s}} \| y^{\mathbf{s}} \circledS e \rangle \rangle \rangle \twoheadrightarrow_{\tilde{\mu}} \langle V'_{\mathbf{s}} \| V_{\mathbf{t}} \circledS e \rangle$$

□

In the aggregate, we can commute many internal reductions on both sides of a command with a top-level reduction.

Lemma 3 (Commutation)**.** *Given any $p \in \{0, +, -\}$, if $v \twoheadrightarrow v'$, $e \twoheadrightarrow e'$, and $\langle v \| e \rangle \rightsquigarrow_{p}^{?} c$, then $\langle v' \| e' \rangle \rightsquigarrow_{p}^{?} c' \twoheadleftarrow c$ for some c'. Moreover, if $v \twoheadrightarrow v'$, $e \twoheadrightarrow e'$, and $\langle v \| e \rangle \rightsquigarrow_{0} c$, then $\langle v' \| e' \rangle \rightsquigarrow_{0} c' \twoheadleftarrow c$ for some c'. In other words, any spans of these forms can be completed by the following commuting diagrams:*

Given $v \twoheadrightarrow v'$ and $e \twoheadrightarrow e'$,



and

Proof. By induction on the two reduction paths, $v \twoheadrightarrow v'$ and $e \twoheadrightarrow e'$, applying Lemma 2 at each step. The stronger commutation for neutral top reductions ($\rightsquigarrow_0$) holds due to the fact that no internal reduction on either side of a command can eliminate a neutral top reduction. □

6. Pre-Candidates: A Two-Sided Interpretation of Types

While some properties, like type safety, are straightforward enough to prove directly (Wright and Felleisen, 1994), other properties, like strong normalization, resist a direct approach. The problem with proving strong normalization is that just inducting over syntax or typing derivations is far too

weak. Instead, the standard practice uses a more indirect approach based on the idea behind Tait's method (Tait, 1967) and reducibility candidates (Girard et al., 1989): set up an interpretation for types that serves as a waypoint between syntax and safety. The domain in which types are interpreted should encompass all programs of that type (adequacy) and also fit inside the intended candidate property (safety).

$$\text{Syntax} \overset{\text{Adequacy}}{\Longrightarrow} \text{Interpretation} \overset{\text{Safety}}{\Longrightarrow} \text{Candidate}$$

When interpreting types, the definition is usually designed with safety in mind: interpretations contain only safe programs by construction, but their adequacy needs to be justified. Instead, we will orient ourselves the other way in the style of symmetric candidates (Barbanera and Berardi, 1994), where the interpretations for types are designed with adequacy in mind: interpretations contain all the necessary well-typed programs by construction, but their safety needs to be justified. But that means we need to consider things which are not yet known to be safe, and so are not a candidate interpretation for any type. Therefore, we work in the larger and more lax domain of *pre-candidates* which encompasses all possible candidates but does not impose the necessary safety conditions.

$$\text{Syntax} \overset{\text{Adequacy}}{\Longrightarrow} \text{Interpretation} \overset{\text{Safety}}{\Longrightarrow} \text{Candidate} \overset{\text{Relax}}{\Longrightarrow} \text{Pre-Candidate}$$

Since we will be discussing multiple versions of "candidates," it's useful to begin one step earlier at the domain of "pre-candidates," which outlines the overall shape of candidates but stops short of enforcing the crucial properties that ensure they are well-behaved. Before getting into proper candidates, we can already explore some useful general properties about the domain of pre-candidates as a whole.

6.1. The Two Lattices of Pre-Candidates

In the λ-calculus, types only describe terms. But in the sequent calculus, types describe *both* terms and co-terms. For this reason, pre-candidates are dual objects containing both a term side and a co-term side, which is expressive enough to encompass both roles of types in the sequent calculus.

Definition 4 (Pre-candidate)**.** A *pre-candidate (of* $\mathbf{s}$*)* is a pair $\mathcal{A} = (\mathcal{A}^+, \mathcal{A}^-)$ of a set $(\mathcal{A}^+)$ of strongly normalizing terms (of discipline $\mathbf{s}$) and a set $(\mathcal{A}^-)$ of strongly normalizing co-terms (also of $\mathbf{s}$). As notation on a pre-candidate

$\mathcal{A}$, we write $\mathcal{A}^+$ for the first component of $\mathcal{A}$ (the set of terms), $\mathcal{A}^-$ for the second component of $\mathcal{A}$ (the set of co-terms), $v \in \mathcal{A}$ as shorthand for $v \in \mathcal{A}^+$ and $e \in \mathcal{A}$ as shorthand for $e \in \mathcal{A}^-$.

Two important pre-candidates the "biggest" pre-candidate $\mathcal{W}_{\mathbf{s}}$ of all strongly normalizing terms and co-terms of a particular discipline $\mathbf{s}$, and the smaller pre-candidate $\mathcal{V}_{\mathbf{s}}$ of only strongly normalizing values and co-values.

$$\mathcal{W}_{\mathbf{s}} \triangleq (\{v_{\mathbf{s}} \mid v_{\mathbf{s}} \text{ is strongly normalizing}\}, \{e_{\mathbf{s}} \mid e_{\mathbf{s}} \text{ is strongly normalizing}\})$$
$$\mathcal{V}_{\mathbf{s}} \triangleq (\{v_{\mathbf{s}} \in \mathcal{W}_{\mathbf{s}} \mid v_{\mathbf{s}} \text{ is a value of } \mathbf{s}\}, \{e_{\mathbf{s}} \in \mathcal{W}_{\mathbf{s}} \mid e_{\mathbf{s}} \text{ is a co-value of } \mathbf{s}\})$$

Since pre-candidates are effectively two-sided objects, there is more than one way to relate them compared with just a single set, since the two sides can be either in agreement or opposed to one another. The first relation is just straightforward containment where both sides are treated the same which we call "refinement," which corresponds to the ordinary subset relation. The second relation has the two sides contrary to one another which we call "subtyping," since it corresponds to the notion of behavioral subtyping on candidates. The idea behind subtyping $\mathcal{A} \leq \mathcal{B}$ is that every value of $\mathcal{A}$ is also a valid value of $\mathcal{B}$, but also dually every observation of $\mathcal{B}$ can be used on values of $\mathcal{A}$.

Definition 5 (Subtyping and Refinement)**.** The *refinement* ($\sqsubseteq$) and *subtyping* ($\leq$) orders on pre-candidates $\mathcal{A} = (\mathcal{A}^+, \mathcal{A}^-)$ and $\mathcal{B} = (\mathcal{B}^+, \mathcal{B}^-)$ is:

$$\mathcal{A} \sqsubseteq \mathcal{B} \triangleq (\mathcal{A}^+ \subseteq \mathcal{B}^+) \wedge (\mathcal{A}^- \subseteq \mathcal{B}^-)$$
$$\mathcal{A} \leq \mathcal{B} \triangleq (\mathcal{A}^+ \subseteq \mathcal{B}^+) \wedge (\mathcal{A}^- \supseteq \mathcal{B}^-)$$

When $\mathcal{A} \sqsubseteq \mathcal{B}$ we say "$\mathcal{A}$ *refines* $\mathcal{B}$" (dually, "$\mathcal{B}$ *extends* $\mathcal{A}$") and when $\mathcal{A} \leq \mathcal{B}$ we say that "$\mathcal{A}$ is a *subtype* of $\mathcal{B}$" (dually, "$\mathcal{B}$ is a *supertype* of $\mathcal{A}$"). Note that refinement between pre-candidates says that one is wholly contained within the other, whereas subtyping order is inverted on negative side of pre-candidates. Both orders form separate complete lattices which come with their own notions of union and intersections (written $\sqcup$ and $\sqcap$ for refinement and $\vee$ and $\wedge$ for subtyping), defined as:

$$\mathcal{A} \sqcup \mathcal{B} \triangleq (\mathcal{A}^+ \cup \mathcal{B}^+, \mathcal{A}^- \cup \mathcal{B}^-) \qquad \mathcal{A} \sqcap \mathcal{B} \triangleq (\mathcal{A}^+ \cap \mathcal{B}^+, \mathcal{A}^- \cap \mathcal{B}^-)$$
$$\mathcal{A} \vee \mathcal{B} \triangleq (\mathcal{A}^+ \cup \mathcal{B}^+, \mathcal{A}^- \cap \mathcal{B}^-) \qquad \mathcal{A} \wedge \mathcal{B} \triangleq (\mathcal{A}^+ \cap \mathcal{B}^+, \mathcal{A}^- \cup \mathcal{B}^-)$$

Property 5. *The union and intersection operations of both refinement and subtyping are monotonic in each argument with respect to each ordering. For example, the notable case is, for any pre-candidates $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ of* **s***:*

$$\textit{if } \mathcal{A} \leq \mathcal{B} \textit{ then } \mathcal{C} \sqcup \mathcal{A} \leq \mathcal{C} \sqcup \mathcal{B}.$$

Proof. This follows from analogous properties of the underlying set operations. That is, if $A \subset B$ then $C \cup A \subset C \cup B$ and $C \cap A \subset C \cap B$. □

6.2. Orthogonality

The sequent calculus has more than just terms and co-terms; it also has commands which represent execution states. What role do they play in the domain of pre-candidates? They are the key variable in the most fundamental operation on pre-candidates: orthogonality. The idea behind orthogonality is to take a set of especially well-behaved commands—for example, the set of all strongly normalizing commands—and use those commands as a test for generating well-behaved partners. That is, given some co-terms as a set of observations, orthogonality can give back all terms that are strongly normalizing under each of those observations. Or dually, given some terms as a set of results, we can give back all co-terms that are strongly normalizing when given each those results.

Definition 6 (Orthogonality)**.** Given *any* set of commands $\mathcal{P}$, the **s** *orthogonality* operation is defined dually on any set of **s**-terms ($\mathcal{A}^+$) and set of **s**-co-terms ($\mathcal{A}^-$) as:

$$\mathcal{A}^{+\mathcal{P}} \triangleq \{e \in \mathcal{W}_{\mathbf{s}} \mid \forall v \in \mathcal{A}^+, \langle v \| e \rangle \in \mathcal{P}\}$$
$$\mathcal{A}^{-\mathcal{P}} \triangleq \{v \in \mathcal{W}_{\mathbf{s}} \mid \forall e \in \mathcal{A}^-, \langle v \| e \rangle \in \mathcal{P}\}$$

The fact that the appropriate $\mathcal{W}_{\mathbf{s}}$ is chosen to match the (co-)terms of $\mathcal{A}^+$ and $\mathcal{A}^-$ ensures that the above commands are well-disciplined. Since pre-candidates are two-sided objects, we can lift these two operations independently onto pre-candidates to get the **s** *pre-orthogonality* operation, which uses different sets of commands ($\mathcal{P}$ and $\mathcal{Q}$) for each side as follows:

$$(\mathcal{A}^+, \mathcal{A}^-)^{(\mathcal{P},\mathcal{Q})} \triangleq (\mathcal{A}^{-\mathcal{P}}, \mathcal{A}^{+\mathcal{Q}})$$

Pre-orthogonality is asymmetric, in that both sides of a pre-candidate may be treated differently. In turn, the **s** *orthogonality* operation imposes symmetry on pre-orthogonality, and is defined as follows:

$$\mathcal{A}^{\mathcal{P}} \triangleq \mathcal{A}^{(\mathcal{P},\mathcal{P})} = (\mathcal{A}^{-\mathcal{P}}, \mathcal{A}^{+\mathcal{P}})$$

The most important application of (pre-)orthogonality is with respect to the set of all strongly normalizing commands, which we write as $\perp\!\!\!\perp$:

$$\perp\!\!\!\perp \triangleq \{c \mid c \text{ is strongly normalizing}\}$$

So that the traditional notation $\mathcal{A}^{\perp\!\!\!\perp}$ refers to the pre-candidate which includes all normalizing (co-)terms which form normalizing commands with the (co-)terms of $\mathcal{A}$. Inlining the definitions, $\mathcal{A}^{\perp\!\!\!\perp}$ is the usual of pair operations—for converting a set of terms to a set of co-terms, and vice-versa—which is:

$$\mathcal{A}^{\perp\!\!\!\perp} = (\{v \in \mathcal{W}_{\mathbf{s}} \mid \forall e \in \mathcal{A}, \langle v \| e \rangle \in \perp\!\!\!\perp\}, \{e \in \mathcal{W}_{\mathbf{s}} \mid \forall v \in \mathcal{A}, \langle v \| e \rangle \in \perp\!\!\!\perp\})$$

It follows then the following facts hold about a pre-candidate $\mathcal{A}$ exactly when it is a fixed point of this instance of orthogonality ($\mathcal{A} = \mathcal{A}^{\perp\!\!\!\perp}$):

(i) (Derived from $\mathcal{A} \sqsubseteq \mathcal{A}^{\perp\!\!\!\perp}$): For all $v, e \in \mathcal{A}$, $\langle v \| e \rangle \in \perp\!\!\!\perp$.

(ii) (Derived from $\mathcal{A}^{\perp\!\!\!\perp} \sqsubseteq \mathcal{A}$): If $\langle v \| e \rangle \in \perp\!\!\!\perp$ for all $e \in \mathcal{A}$ then it must be that $v \in \mathcal{A}$. Dually, if $\langle v \| e \rangle \in \perp\!\!\!\perp$ for all $v \in \mathcal{A}$ then $e \in \mathcal{A}$.

The first fact represents a form of "soundness" about $\mathcal{A}$, where every combination of terms and co-terms drawn from $\mathcal{A}$ is safe (*i.e.,* is a strongly-normalizing command in $\perp\!\!\!\perp$). The second fact represents a form of "completeness" about $\mathcal{A}$, where every such "safe" (co-)term is already included in $\mathcal{A}$. The importance of fixed points for deriving these kinds of soundness and completeness properties will be a recurring theme in our development.

Besides the obvious connection between subtyping of pre-candidates and subtyping of types, one reason motivating the two separate orderings is that they each have a very different relationship with the fundamental orthogonality operation. In particular, refinement order exhibits the following usual standard properties that arise in the study of *biorthogonality* (Girard, 1987; Pitts, 2000; Mellies and Vouillon, 2005) and *classical realizability* (Krivine, 2005; Munch-Maccagnoni, 2009), whereas subtyping order is stable under orthogonality. First, note how orthogonality interacts with the two different pre-candidate relationships.

Property 6. *For any sets of commands $\mathcal{P}$ and $\mathcal{Q}$, and any pre-candidates $\mathcal{A}$ and $\mathcal{B}$ of* $\mathbf{s}$*:*

(i) Extension: *If $\mathcal{P}' \subseteq \mathcal{P}$ and $\mathcal{Q}' \subseteq \mathcal{Q}$ then $\mathcal{A}^{(\mathcal{P}',\mathcal{Q}')} \sqsubseteq \mathcal{A}^{(\mathcal{P},\mathcal{Q})}$*

(ii) Antitonicity (*a.k.a* contrapositive): *If* $\mathcal{A} \sqsubseteq \mathcal{B}$ *then* $\mathcal{B}^{(\mathcal{P},\mathcal{Q})} \sqsubseteq \mathcal{A}^{(\mathcal{P},\mathcal{Q})}$

(iii) Monotonicity: *If* $\mathcal{A} \leq \mathcal{B}$ *then* $\mathcal{A}^{(\mathcal{P},\mathcal{Q})} \leq \mathcal{B}^{(\mathcal{P},\mathcal{Q})}$

Furthermore all four union ($\sqcup$, $\vee$*) and intersection (*$\sqcap$, $\wedge$*) operations are monotonic in both arguments with respect to both orders. For example, two of the most important instances are, for any* $\mathcal{A} \leq \mathcal{B}$ *and constant* $\mathcal{C}$*:*

$$\mathcal{A} \sqcap \mathcal{C} \leq \mathcal{B} \sqcap \mathcal{C} \qquad\qquad \mathcal{A} \sqcup \mathcal{C} \leq \mathcal{B} \sqcap \mathcal{C}$$

Proof. Suppose the pre-candidates are $\mathcal{A} = (\mathcal{A}^+, \mathcal{A}^-)$ and $\mathcal{B} = (\mathcal{B}^+, \mathcal{B}^-)$.

(i) Let $v \in \mathcal{A}^{+\mathcal{P}'}$. For any $e \in \mathcal{A}^-$, we know that $\langle v \| e \rangle \in \mathcal{P}' \subseteq \mathcal{P}$, and thus $v \in \mathcal{A}^{+\mathcal{P}}$ as well. Similarly, every $e \in \mathcal{A}^{-\mathcal{Q}'}$ is also in $\mathcal{A}^{-\mathcal{Q}}$.

(ii) Let $v \in \mathcal{B}^{+\mathcal{P}}$. For any $e \in \mathcal{A}^-$ it must be that $e \in \mathcal{B}^-$ as well (due to $\mathcal{A} \sqsubseteq \mathcal{B}$), which forces $\langle v \| e \rangle \in \mathcal{P}$ and thus $v \in \mathcal{A}^{+\mathcal{P}}$. Similarly, it must be that $\mathcal{B}^{-\mathcal{Q}} \subseteq \mathcal{A}^{-\mathcal{Q}}$.

(iii) The assumption $(\mathcal{A}^+, \mathcal{A}^-) \leq (\mathcal{B}^+, \mathcal{B}^-)$ can be equivalently restated as $(\mathcal{A}^+, \mathcal{B}^-) \sqsubseteq (\mathcal{B}^+, \mathcal{A}^-)$. From the previous point (antitonicity), we know

$$(\mathcal{A}^{-\mathcal{P}}, \mathcal{B}^{+\mathcal{Q}}) = (\mathcal{B}^+, \mathcal{A}^-)^{(\mathcal{P},\mathcal{Q})} \sqsubseteq (\mathcal{A}^+, \mathcal{B}^-)^{(\mathcal{P},\mathcal{Q})} = (\mathcal{B}^{-\mathcal{P}}, \mathcal{A}^{+\mathcal{Q}})$$

which is equivalent to

$$(\mathcal{A}^{-\mathcal{P}}, \mathcal{A}^{+\mathcal{Q}}) = (\mathcal{A}^+, \mathcal{A}^-)^{(\mathcal{P},\mathcal{Q})} \leq (\mathcal{B}^+, \mathcal{B}^-)^{(\mathcal{P},\mathcal{Q})} = (\mathcal{B}^{-\mathcal{P}}, \mathcal{B}^{+\mathcal{Q}})$$

The monotonicity of the intersection and union operations follows from monotonicity of their underlying set operations. That is, for any sets $A \subseteq B$ and C, both $A \cup C \subseteq B \cup C$ and $A \cap C \subseteq B \cap C$ hold. □

In addition to these basic ordering properties, pre-orthogonality also generalizes the standard properties of orthogonality in terms of refinement of pre-candidates.

Property 7. *The following properties hold for any sets of commands* $\mathcal{P}$ *and* $\mathcal{Q}$*, and any pre-candidates* $\mathcal{A}$ *and* $\mathcal{B}$ *of* $\mathbf{s}$*:*

(i) Double orthogonal introduction: $\mathcal{A} \sqsubseteq \mathcal{A}^{(\mathcal{P},\mathcal{Q})(\mathcal{Q},\mathcal{P})}$

(ii) Triple orthogonal elimination: $\mathcal{A}^{(\mathcal{P},\mathcal{Q})(\mathcal{Q},\mathcal{P})(\mathcal{P},\mathcal{Q})} = \mathcal{A}^{(\mathcal{P},\mathcal{Q})}$

(iii) Orthogonal of union: $\mathcal{A}^{(\mathcal{P},\mathcal{Q})} \sqcap \mathcal{B}^{(\mathcal{P},\mathcal{Q})} = (\mathcal{A} \sqcup \mathcal{B})^{(\mathcal{P},\mathcal{Q})}$

(iv) Orthogonal of intersection: $\mathcal{A}^{(\mathcal{P},\mathcal{Q})} \sqcup \mathcal{B}^{(\mathcal{P},\mathcal{Q})} \sqsubseteq (\mathcal{A} \sqcap \mathcal{B})^{(\mathcal{P},\mathcal{Q})}$

However, it is not *necessarily the case that the reverse orders* $\mathcal{A}^{(\mathcal{P},\mathcal{Q})(\mathcal{Q},\mathcal{P})} \sqsubseteq \mathcal{A}$ *or* $(\mathcal{A} \sqcap \mathcal{B})^{(\mathcal{P},\mathcal{Q})} \sqsubseteq \mathcal{A}^{(\mathcal{P},\mathcal{Q})} \sqcup \mathcal{B}^{(\mathcal{P},\mathcal{Q})}$ *hold.*

Proof. Suppose the pre-candidates are $\mathcal{A} = (\mathcal{A}^+, \mathcal{A}^-)$ and $\mathcal{B} = (\mathcal{B}^+, \mathcal{B}^-)$.

(i) Note that $\mathcal{A}^{(\mathcal{P},\mathcal{Q})(\mathcal{Q},\mathcal{P})} = (\mathcal{A}^{-\mathcal{P}}, \mathcal{A}^{+\mathcal{Q}})^{(\mathcal{Q},\mathcal{P})} = (\mathcal{A}^{+\mathcal{Q}\mathcal{Q}}, \mathcal{A}^{-\mathcal{P}\mathcal{P}})$. Suppose $v \in \mathcal{A}^+$ and $e \in \mathcal{A}^{+\mathcal{Q}}$. By definition of $\mathcal{A}^{+\mathcal{Q}}$ we know that $\langle v \| e \rangle \in \mathcal{Q}$, which forces $v \in \mathcal{A}^{+\mathcal{Q}\mathcal{Q}}$ as well. Similarly, $\mathcal{A}^- \subseteq \mathcal{A}^{-\mathcal{P}\mathcal{P}}$.

(ii) First, we know that $\mathcal{A} \sqsubseteq \mathcal{A}^{(\mathcal{P},\mathcal{Q})(\mathcal{Q},\mathcal{P})}$ by double orthogonal introduction on $\mathcal{A}$, and so $\mathcal{A}^{(\mathcal{P},\mathcal{Q})(\mathcal{Q},\mathcal{P})(\mathcal{P},\mathcal{Q})} \sqsubseteq \mathcal{A}^{(\mathcal{P},\mathcal{Q})}$ by contrapositive. Second, we know that $\mathcal{A}^{(\mathcal{P},\mathcal{Q})} \sqsubseteq \mathcal{A}^{(\mathcal{P},\mathcal{Q})(\mathcal{Q},\mathcal{P})(\mathcal{P},\mathcal{Q})}$ by double orthogonal introduction on $\mathcal{A}^{(\mathcal{P},\mathcal{Q})}$ with $\mathcal{P}$ and $\mathcal{Q}$ reversed. Therefore, the two are equal.

(iii) Since both $\mathcal{A}, \mathcal{B} \sqsubseteq \mathcal{A} \sqcup \mathcal{B}$, we know that $(\mathcal{A} \sqcup \mathcal{B})^{(\mathcal{P},\mathcal{Q})} \sqsubseteq \mathcal{A}^{(\mathcal{P},\mathcal{Q})}, \mathcal{B}^{(\mathcal{P},\mathcal{Q})}$ by contrapositive, and so $(\mathcal{A} \sqcup \mathcal{B})^{(\mathcal{P},\mathcal{Q})} \sqsubseteq \mathcal{A}^{(\mathcal{P},\mathcal{Q})} \sqcap \mathcal{B}^{(\mathcal{P},\mathcal{Q})}$. Now, let $v \in \mathcal{A}^{(\mathcal{P},\mathcal{Q})} \sqcap \mathcal{B}^{(\mathcal{P},\mathcal{Q})}$ and $e \in \mathcal{A} \sqcup \mathcal{B}$. In either case $e \in \mathcal{A}$ or $e \in \mathcal{B}$, it must be that $\langle v \| e \rangle \in \mathcal{P}$, so $v \in (\mathcal{A} \sqcup \mathcal{B})^{\mathcal{P}}$ as well. *I.e.,* $\mathcal{A}^{-\mathcal{P}} \cap \mathcal{B}^{-\mathcal{P}} \subseteq (\mathcal{A}^- \cup \mathcal{B}^-)^{\mathcal{P}}$. The same holds for co-terms, so that $\mathcal{A}^{+\mathcal{Q}} \cap \mathcal{B}^{+\mathcal{Q}} \subseteq (\mathcal{A}^+ \cup \mathcal{B}^+)^{\mathcal{Q}}$.

(iv) Since both $\mathcal{A}, \mathcal{B} \sqsupseteq \mathcal{A} \sqcap \mathcal{B}$, we know that both $(\mathcal{A} \sqcap \mathcal{B})^{(\mathcal{P},\mathcal{Q})} \sqsupseteq \mathcal{A}^{(\mathcal{P},\mathcal{Q})}, \mathcal{B}^{(\mathcal{P},\mathcal{Q})}$ by contrapositive, and so $(\mathcal{A} \sqcap \mathcal{B})^{(\mathcal{P},\mathcal{Q})} \sqsupseteq \mathcal{A}^{(\mathcal{P},\mathcal{Q})} \sqcup \mathcal{B}^{(\mathcal{P},\mathcal{Q})}$. □

Intuitively, these properties can be read as laws of logical negation if we read $\mathcal{A}^{\perp\!\!\!\perp}$ as "not $\mathcal{A}$", $\mathcal{A} \sqsubseteq \mathcal{B}$ as "$\mathcal{A}$ implies $\mathcal{B}$, $\sqcup$ as "or," and $\sqcap$ as "and." By instantiating the above properties to the traditional orthogonality operation, we have the correspondences of $\mathcal{A} \sqsubseteq \mathcal{A}^{\perp\!\!\!\perp\perp\!\!\!\perp}$ to double negation introduction, $\mathcal{A}^{\perp\!\!\!\perp\perp\!\!\!\perp\perp\!\!\!\perp} = \mathcal{A}^{\perp\!\!\!\perp}$ to triple negation elimination, and both $\mathcal{A}^{\perp\!\!\!\perp} \sqcap \mathcal{B}^{\perp\!\!\!\perp} = (\mathcal{A} \sqcup \mathcal{B})^{\perp\!\!\!\perp}$ and $\mathcal{A}^{\perp\!\!\!\perp} \sqcup \mathcal{B}^{\perp\!\!\!\perp} \sqsubseteq (\mathcal{A} \sqcap \mathcal{B})^{\perp\!\!\!\perp}$ to three of the fourth De Morgan laws. Additionally, the fact that double negation elimination ($\mathcal{A}^{\perp\!\!\!\perp\perp\!\!\!\perp} \sqsubseteq \mathcal{A}$) and the fourth De Morgan law ($(\mathcal{A} \sqcap \mathcal{B})^{\perp\!\!\!\perp} \sqsubseteq \mathcal{A}^{\perp\!\!\!\perp} \sqcup \mathcal{B}^{\perp\!\!\!\perp}$) do not necessarily holds means that orthogonality is a model of *intuitionistic*, rather than *classical*, negation.

Finally, we note that orthogonality has the ability to transfer some important facts about the main safety property of interest—the set $\perp\!\!\!\perp$ of strongly normalizing commands—to pre-candidates generated from it. For example, closure under reduction is a crucial fact about strong normalization which is needed for proofs based on reducibility candidates.

Definition 7. A set of commands, $\mathcal{P}$, is *closed under reduction* when $c \in \mathcal{P}$ and $c \twoheadrightarrow c'$ implies $c' \in \mathcal{P}$. Likewise, a pre-candidate $\mathcal{A}$ is *closed under reduction* when both $v \in \mathcal{A}$ and $v \twoheadrightarrow v'$ implies $v' \in \mathcal{A}$ and $e \in \mathcal{A}$ and $e \twoheadrightarrow e'$ implies $e' \in \mathcal{A}$.

Property 8. *If a set of commands $\mathcal{P}$ is closed under reduction, so is $\mathcal{A}^{\mathcal{P}}$.*

Proof. Let $v \in \mathcal{A}$ and $v \twoheadrightarrow v'$. For any $e \in \mathcal{A}$, we know that $\langle v \| e \rangle \in \mathcal{P}$, $\langle v \| e \rangle \twoheadrightarrow \langle v' \| e \rangle$, and therefore $\langle v' \| e \rangle \in \mathcal{P}$ by assumption. Therefore $v' \in \mathcal{A}$ as well. $\mathcal{A}$ is closed under reduction of co-terms for the same reason. □

7. Reducibility Candidates for Confluent Normalization

For our first, simpler, approach to candidate semantics for types, we will look to the *reducibility candidates* and *biorthogonality* methods which are appropriate for deterministic evaluation (*e.g.,* like the call-by-value **v** and call-by-name **n** disciplines). The main key any reducibility candidates proof is a form of "expansion" lemma, which says that things which step to something good must have been good to begin with (*i.e.,* "goodness" is closed under $\rightsquigarrow$ expansion). For our specific setting, this expansion lemma takes the following form, which holds under the assumption that the chosen discipline **s** is deterministic.

Lemma 4 (Deterministic Expansion)**.** *Given any deterministic admissible* **s***, if $v, e \in \mathcal{W}_{\mathbf{s}}$ (*i.e., *v and e are normalizing) and $\langle v \| e \rangle \rightsquigarrow c \in \perp\!\!\!\perp$ (*i.e., $\langle v \| e \rangle$ steps to c which is normalizing) then $\langle v \| e \rangle \in \perp\!\!\!\perp$ (*i.e., $\langle v \| e \rangle$ is normalizing).*

Proof. We will show that $\langle v \| e \rangle$ must be strongly normalizing by assuming there is an infinite reduction sequence starting from $\langle v \| e \rangle$ and demonstrating a contradiction.[1] By Lemma 1, any infinite reduction sequence starting from $\langle v \| e \rangle$ must begin as

$$\langle v \| e \rangle \twoheadrightarrow \langle v' \| e' \rangle \rightsquigarrow c' \qquad \text{where} \qquad v \twoheadrightarrow v' \qquad \text{and} \qquad e \twoheadrightarrow e'$$

and c' has a further infinite reduction sequence. And by applying Lemma 3 to $\langle v' \| e' \rangle \twoheadleftarrow \langle v \| e \rangle \rightsquigarrow c$, we know there is some c'' such that $\langle v' \| e' \rangle \rightsquigarrow^{?} c'' \twoheadleftarrow c$.

[1] In fact, this lemma is a consequence of the later Lemma 7, but we show the self-contained proof here for the purpose of illustration.

Because $c' \leftsquigarrow \langle v' \| e' \rangle \rightsquigarrow^? c''$, it follows that $c'' \rightsquigarrow^? c'$ from the assumption that $\mathbf{s}$ is deterministic, which leads to $c \twoheadrightarrow c'' \rightsquigarrow^? c'$. But this is a contradiction, since strong normalization is closed under reduction, and yet the normalizing c reduces to the non-normalizing c'. □

7.1. Defining Reducibility Candidates

However, disciplined reduction has a serious consequence on this form of expansion. For example, consider the challenge of justifying in the call-by-value $\mathbf{v}$ discipline that a $\tilde{\mu}$-abstraction $\tilde{\mu}x.c$ is a co-term of some $\mathcal{A}$. We would like to argue behaviorally about how $\tilde{\mu}x.c$ behaves when paired with terms of $\mathcal{A}$. That is, we might know that for any $V_{\mathbf{v}} \in \mathcal{A}$,

$$\langle V_{\mathbf{v}} \| \tilde{\mu}x.c \rangle \rightsquigarrow_{\tilde{\mu}_{\mathbf{v}}} c\{V_{\mathbf{v}}/x\} \in \perp\!\!\!\perp$$

or in other words, $\tilde{\mu}x.c$ forms a strongly-normalizing command after one step when paired with any $\mathbf{v}$-value of $\mathcal{A}$. But not every term is a $\mathbf{v}$-value, so we can't use $\tilde{\mu}_{\mathbf{v}}$-reduction to say anything about the other terms of $\mathcal{A}$! For this reason, we need to consider the (co-)value restriction as part of the definition of reducibility candidates, corresponding to Munch-Maccagnoni's (2009) notion of *generation*.

Definition 8 (Restriction)**.** The *restriction* on a pre-candidate $\mathcal{A}$ of $\mathbf{s}$ by another pre-candidate $\mathcal{B}$ of $\mathbf{s}$ is defined as: $\mathcal{A}^{\mathcal{B}} \triangleq \mathcal{A} \sqcap \mathcal{B}$. Note that $\mathcal{A}^{\mathcal{B}} \sqsubseteq \mathcal{A}$ by definition. The *(co-)value restriction* on a pre-candiate $\mathcal{A}$ of $\mathbf{s}$, written as $\mathcal{A}^{\mathcal{V}_{\mathbf{s}}}$, is a special case of restriction on $\mathcal{A}$ by the pre-candidate of all $\mathcal{S}$-(co-)values $\mathcal{V}_{\mathbf{s}}$. When the discipline $\mathbf{s}$ can be inferred from context, we will leave it implicit and write the (co-)value restriction as just $\mathcal{A}^{\mathcal{V}}$.

Definition 9 (Reducibility Candidate)**.** A *reducibility candidate (of* $\mathbf{s}$*)* is a pre-candidate $\mathcal{A}$ (of $\mathbf{s}$) such that $\mathcal{A}^{\mathcal{V}\perp\!\!\!\perp} \sqsubseteq \mathcal{A} \sqsubseteq \mathcal{A}^{\perp\!\!\!\perp}$. In other words, a reducibility candidate $\mathcal{A}$ of $\mathbf{s}$ is any pre-candidate of $\mathbf{s}$ such that the following two properties hold:

(i) *Soundness* ($\mathcal{A} \sqsubseteq \mathcal{A}^{\perp\!\!\!\perp}$): For all $v, e \in \mathcal{A}$, the command $\langle v \| e \rangle$ is strongly normalizing.

(ii) *Completeness* ($\mathcal{A}^{\mathcal{V}\perp\!\!\!\perp} \sqsubseteq \mathcal{A}$): If v is strongly normalizing as well as $\langle v \| E \rangle$ for all $E \in \mathcal{A}^{\mathcal{V}}$, then $v \in \mathcal{A}$. Dually, if e is strongly normalizing as well as $\langle V \| e \rangle$ for all $V \in \mathcal{A}^{\mathcal{V}}$, then $e \in \mathcal{A}$.

We denote the set of all reducibility candidates of $\mathbf{s}$ by $CR_{\mathbf{s}}$.

Intuitively, the soundness property of reducibility candidates ensures that they don't have too many (co-)terms, which justifies that the *Cut* rule is sound, whereas the completeness property ensures that there are enough (co-)terms, which justifies the generic *ActR* and *ActL* rules of every type along with type-specific inference rules. The fact that the completeness requirement of reducibility candidates ($\mathcal{A}^{\mathcal{V}\perp\!\!\!\perp} \sqsubseteq \mathcal{A}$) only requires checking potential members against the *values* or *co-values* of that candidate means that the standard reduction rules are enough to justify membership of complex computations by expansion. Returning back to the above example, if we know that $\mathcal{A}$ is a reducibility candidate and $c\{V/x\} \in \perp\!\!\!\perp$ for any $V \in \mathcal{A}$, then we can conclude from expansion (Proposition 4) and completeness (Definition 9) that $\tilde{\mu}x.c \in \mathcal{A}$.

It turns out there is another equivalent definition of reducibility candidates, based on the standard properties of orthogonality from Proposition 7: reducibility candidates are exactly the fixed points of (co-)value-restricted orthogonality.

Property 9. *For any deterministic admissible discipline* $\mathbf{s}$ *and pre-candidate* $\mathcal{A}$ *of* $\mathbf{s}$*,* $\mathcal{A}$ *is a reducibility candidate of* $\mathbf{s}$ *if and only if* $\mathcal{A} = \mathcal{A}^{\mathcal{V}\perp\!\!\!\perp}$.

Fixed points play a prominent role later in Section 8, where we give a proof of this property.

7.2. Constructing Reducibility Candidates

So far, we have said *what* reducibility candidates are, but not yet *how* to make one of them. Here are two dual ways (one *Pos*itively-oriented and the other *Neg*atively-oriented) to construct a reducibility candidate of $\mathbf{s}$ from a set of $\mathbf{s}$-values (C) or $\mathbf{s}$-co-values (O), along with a common operation $\mathcal{R}(-)$ for generating a complete reducibility candidate from a sound pre-candidate $\mathcal{A}$ made up of $\mathbf{s}$-(co-)values.

$$Pos(C) \triangleq (C, C^{\perp\!\!\!\perp})^{\mathcal{V}\perp\!\!\!\perp\mathcal{V}} \qquad Neg(O) \triangleq (O^{\perp\!\!\!\perp}, O)^{\mathcal{V}\perp\!\!\!\perp\mathcal{V}} \qquad \mathcal{R}(\mathcal{A}) \triangleq \mathcal{A}^{\perp\!\!\!\perp}$$

Inlining the definitions, we can see that the two dual ways of constructing a reducibility candidate appears to use *three* applications of orthogonality:

$$\mathcal{R}(Pos(C)) = (C, C^{\perp\!\!\!\perp\mathcal{V}})^{\perp\!\!\!\perp\mathcal{V}\perp\!\!\!\perp} = (C^{\perp\!\!\!\perp\mathcal{V}\perp\!\!\!\perp}, C^{\perp\!\!\!\perp\mathcal{V}\perp\!\!\!\perp\mathcal{V}\perp\!\!\!\perp})$$
$$\mathcal{R}(Neg(O)) = (O^{\perp\!\!\!\perp\mathcal{V}}, O)^{\perp\!\!\!\perp\mathcal{V}\perp\!\!\!\perp} = (O^{\perp\!\!\!\perp\mathcal{V}\perp\!\!\!\perp\mathcal{V}\perp\!\!\!\perp}, O^{\perp\!\!\!\perp\mathcal{V}\perp\!\!\!\perp})$$

Why three, instead of just the usual two? Because the correctness of this construction relies on a weaker version of the triple orthogonality elimination property in order to accommodate the (co-)value restriction.

Property 10. *For any sets of commands $\mathcal{P}$ and $\mathcal{Q}$, and any pre-candidates $\mathcal{A}$ and $\mathcal{B}$ of $\mathbf{s}$:*

(i) Restricted orthogonal: $\mathcal{A}^{(\mathcal{P},\mathcal{Q})} \sqsubseteq \mathcal{A}^{\mathcal{B}(\mathcal{P},\mathcal{Q})}$

(ii) Restricted double orthogonal introduction: $\mathcal{A}^{\mathcal{B}} \sqsubseteq \mathcal{A}^{\mathcal{B}(\mathcal{P},\mathcal{Q})\mathcal{B}(\mathcal{Q},\mathcal{P})\mathcal{B}}$

(iii) Restricted triple orthogonal elimination: $\mathcal{A}^{\mathcal{B}(\mathcal{P},\mathcal{Q})\mathcal{B}(\mathcal{P},\mathcal{Q})\mathcal{B}(\mathcal{P},\mathcal{Q})\mathcal{B}} = \mathcal{A}^{\mathcal{B}(\mathcal{P},\mathcal{Q})\mathcal{B}}$

Proof. (i) Follows from the antitonicity of pre-orthogonality on the definition of restriction, $\mathcal{A}^{\mathcal{B}} = \mathcal{A} \sqcap \mathcal{B} \sqsubseteq \mathcal{A}$.

(ii) From double orthogonal introduction and the previous fact we have $\mathcal{A}^{\mathcal{B}} \sqsubseteq \mathcal{A}^{\mathcal{B}(\mathcal{P},\mathcal{Q})(\mathcal{Q},\mathcal{P})} \sqsubseteq \mathcal{A}^{\mathcal{B}(\mathcal{P},\mathcal{Q})\mathcal{B}(\mathcal{Q},\mathcal{P})}$, and so the result follows from monotonicity and idempotency of $\sqcap$.

(iii) Follows from restricted orthogonality introduction analogously to ordinary triple orthogonal elimination from Property 7. □

Lemma 5. *For any deterministic admissible discipline $\mathbf{s}$, set of $\mathbf{s}$-values C, set of $\mathbf{s}$-co-values O, and pre-candidate $\mathcal{A} = \mathcal{A}^{\perp\!\!\!\perp\mathcal{V}}$ of $\mathbf{s}$,*

(i) $Pos(C) = Pos(C)^{\perp\!\!\!\perp\mathcal{V}}$ *and* $(C, \{\}) \sqsubseteq Pos(C)$,

(ii) $Neg(O) = Neg(O)^{\perp\!\!\!\perp\mathcal{V}}$ *and* $(\{\}, O) \sqsubseteq Neg(O)$, *and*

(iii) $\mathcal{R}(\mathcal{A})$ *is a reducibility candidate of* $\mathbf{s}$ *such that* $\mathcal{R}(\mathcal{A})^{\mathcal{V}} = \mathcal{A}$.

Proof. Note that

$$Pos(C) = (C^{\perp\!\!\!\perp\mathcal{V}\perp\!\!\!\perp\mathcal{V}}, C^{\perp\!\!\!\perp\mathcal{V}}) \qquad Pos(C)^{\perp\!\!\!\perp\mathcal{V}} = (C^{\perp\!\!\!\perp\mathcal{V}\perp\!\!\!\perp\mathcal{V}}, C^{\perp\!\!\!\perp\mathcal{V}\perp\!\!\!\perp\mathcal{V}\perp\!\!\!\perp\mathcal{V}})$$
$$Neg(O) = (O^{\perp\!\!\!\perp\mathcal{V}}, O^{\perp\!\!\!\perp\mathcal{V}\perp\!\!\!\perp\mathcal{V}}) \qquad Neg(O)^{\perp\!\!\!\perp\mathcal{V}} = (O^{\perp\!\!\!\perp\mathcal{V}\perp\!\!\!\perp\mathcal{V}\perp\!\!\!\perp\mathcal{V}}, O^{\perp\!\!\!\perp\mathcal{V}\perp\!\!\!\perp\mathcal{V}})$$

The fact that $(C, \{\}) \sqsubseteq Pos(C)$ and $(\{\}, O) \sqsubseteq Neg(C)$ follows from restricted double orthogonal introduction (on the term side for Pos and the co-term side for Neg). The fact that $Pos(C) = Pos(C)^{\perp\!\!\!\perp\mathcal{V}}$ and $Neg(O) = Neg(O)^{\perp\!\!\!\perp\mathcal{V}}$ follows from restricted triple orthogonal elimination (on the co-term side for Pos and on the term side for Neg). Finally, we know that $\mathcal{R}(\mathcal{A}) = \mathcal{A}^{\perp\!\!\!\perp} = \mathcal{A}^{\perp\!\!\!\perp\mathcal{V}\perp\!\!\!\perp} = \mathcal{R}(\mathcal{A})^{\mathcal{V}\perp\!\!\!\perp}$ and so $\mathcal{R}(\mathcal{A})$ is a reducibility candidate due to Property 9, such that $\mathcal{R}(\mathcal{A})^{\mathcal{V}} = \mathcal{A}^{\perp\!\!\!\perp\mathcal{V}} = \mathcal{A}$. □

The use of all three applications of orthogonality are necessary in general for correctly modeling a deterministic discipline $\mathbf{s}$, like the call-by-need $\mathbf{lv}$. However, for the specific call-by-value ($\mathbf{v}$) and call-by-name ($\mathbf{n}$) disciplines, much simpler constructions with only two applications of orthogonality (and hence the nomenclature "biorthogonality") are possible. In fact, these simpler special cases for call-by-value and call-by-name calculi can be derived from the general construction above, due to the fact that *all* co-terms are co-values in call-by-value and *all* terms are values in call-by-name.

Property 11. *For any deterministic admissible discipline* $\mathbf{s}$*, two reducibility candidates* $\mathcal{A}$ *and* $\mathcal{B}$ *of* $\mathbf{s}$ *are equal if and only if* $\mathcal{A}^{\mathcal{V}} = \mathcal{B}^{\mathcal{V}}$.

Proof. Clearly, if $\mathcal{A} = \mathcal{B}$ then $\mathcal{A}^{\mathcal{V}} = \mathcal{B}^{\mathcal{V}}$. Conversely, from Property 9 and the assumption that $\mathcal{A}^{\mathcal{V}} = \mathcal{B}^{\mathcal{V}}$, we know $\mathcal{A} = \mathcal{A}^{\mathcal{V}\perp\!\!\!\perp} = \mathcal{B}^{\mathcal{V}\perp\!\!\!\perp} = \mathcal{B}$. □

Lemma 6. *For any set of* $\mathbf{v}$*-values* C *and set of* $\mathbf{n}$*-co-values* O*:*

(i) $\mathcal{R}(Pos(C)) = (C, C^{\perp\!\!\!\perp})^{\perp\!\!\!\perp} = (C^{\perp\!\!\!\perp\perp\!\!\!\perp}, C^{\perp\!\!\!\perp})$*, and*

(ii) $\mathcal{R}(Neg(O)) = (O^{\perp\!\!\!\perp}, O)^{\perp\!\!\!\perp} = (O^{\perp\!\!\!\perp}, O^{\perp\!\!\!\perp\perp\!\!\!\perp})$.

Proof. First, note that every co-term $e \in \mathcal{W}_{\mathbf{v}}$ is also in $\mathcal{V}_{\mathbf{v}}$. It follows that for every pre-candidate of $\mathbf{v}$ has the property that $(\mathcal{A}^+, \mathcal{A}^-)^{\mathcal{V}} = (\mathcal{A}^+ \cap \mathcal{V}^+, \mathcal{A}^-)$. Thus, so long as $C \subseteq \mathcal{V}^+$, we have

$$Pos(C) = (C, C^{\perp\!\!\!\perp})^{\mathcal{V}\perp\!\!\!\perp\mathcal{V}} = (C, C^{\perp\!\!\!\perp})^{\perp\!\!\!\perp\mathcal{V}} = (C^{\perp\!\!\!\perp\perp\!\!\!\perp}, C^{\perp\!\!\!\perp})^{\mathcal{V}}$$

And since the reducibility candidate $\mathcal{R}(Pos(C))^{\mathcal{V}} = Pos(C) = (C^{\perp\!\!\!\perp\perp\!\!\!\perp}, C^{\perp\!\!\!\perp})^{\mathcal{V}}$ (by Lemma 5), we know that $\mathcal{R}(Pos(C)) = (C^{\perp\!\!\!\perp\perp\!\!\!\perp}, C^{\perp\!\!\!\perp})$ (by Property 11). Similarly, the dual holds for the negative call-by-name ($\mathbf{n}$) construction. □

8. Symmetric Candidates for Non-Confluent Normalization

So, we have only considered the possibility of modeling confluent reduction as found with deterministic disciplines. This assumption showed up in the key expansion property (Lemma 4), which only works when top reduction is deterministic. But in the non-deterministic case, we could have a critical pair of top reductions $c_1 \leftsquigarrow \langle v \| e \rangle \rightsquigarrow c_2$ where just because we know that c_2 is strongly normalizing, that doesn't mean that c_1 must *also* be strongly normalizing, and in fact, it need not be (Lengrand and Miquel, 2008). Since expansion is not guaranteed for non-deterministic disciplines (like with $\mathbf{u}$), we need another approach. Instead of reducibility candidates, here we will consider the strictly more general "symmetric candidates."

8.1. Expansion as the Soundness of Pre-Candidates

The usual expansion property is a statement about the consequences of a single step of an individual term-co-term pair. This hypothesis is too weak to prove the desired result when faced with critically non-confluent reduction induced by two unrelated top reductions. So to counter the problem, we will strengthen the hypothesis of expansion to be about a whole collection of terms and co-terms that together all have the desired expansion property. Fortunately, the notion of pre-candidates that we have been using all along is a convenient way to organizing the collection of terms and co-terms, and orthogonality is a useful way of expressing the hypothesis. We only need to find the right set of commands (*i.e.,* the right predicate on states of computation) that captures the correct notion of "expansion" which we can prove is sound (*i.e.,* all term-co-term interactions are strongly normalizing).

The set of commands which are strongly normalizing after zero or one steps of top reduction with charge p, and which *cannot* reach a non-normalizing command with p top reduction, is:

$$\begin{aligned}\perp\!\!\!\perp^?_p &\triangleq \perp\!\!\!\perp \cup \{c \mid \emptyset \subset \{c' \mid c \rightsquigarrow_p c'\} \subseteq \perp\!\!\!\perp\} \\ &= \perp\!\!\!\perp \cup \{c \mid (\exists c' \in \perp\!\!\!\perp, c \rightsquigarrow_p c') \wedge (\forall c' \notin \perp\!\!\!\perp, c \not\rightsquigarrow_p c')\}\end{aligned}$$

Note that by choosing a different charge p, we can stipulate that only positive, negative or neutral steps are allowed to be used, or any combination thereof. For example, $c \in \perp\!\!\!\perp^?_{+-}$ either when $c \in \perp\!\!\!\perp$ now, or when there is at least one positive or negative step $c \rightsquigarrow_{+-} c' \in \perp\!\!\!\perp$ and *every* such step ends up in $\perp\!\!\!\perp$. Thus, $\perp\!\!\!\perp^?_{+-}$ ensures that a different reduction path introduced by a critical pair of top reductions cannot break normalization. As with the top reduction relation $\rightsquigarrow$, we write $\perp\!\!\!\perp^?$ as shorthand for $\perp\!\!\!\perp^?_{+-0}$.

When the $\rightsquigarrow_p$ reduction relation happens to be deterministic, the set of commands $\perp\!\!\!\perp^?_p$ simplifies to:

$$\perp\!\!\!\perp^?_p = \{c \mid \exists c' \in \perp\!\!\!\perp, c \rightsquigarrow^?_p c'\} \qquad \text{if } c_1 \leftsquigarrow_p c \rightsquigarrow_p c_2 \text{ implies } c_1 = c_2$$

But when the $\rightsquigarrow_p$ reduction relation is not deterministic, the extra generality of $\perp\!\!\!\perp^?_p$ ensures that if we only know that the command is strongly normalizing after one step, then every other possible step also leads to a strongly normalizing command. This extra assurance is the key fact that lets us side-step the problem noted above with reasoning about non-confluent critical pairs.

We can now express a more general form of the expansion property than Lemma 4, which states that for any pre-candidate where every one of its term-co-value and value-co-term combinations is "good" (*i.e.,* in $\perp\!\!\!\perp$) after zero or one steps, then *all* of its term-co-term combinations are in fact "good" now.

Lemma 7 (Pre-Candidate Expansion)**.** *For any admissible discipline* $\mathbf{s}$ *and pre-candidate* $\mathcal{A}$ *of* $\mathbf{s}$*, if* $\mathcal{A}$ *is forward closed and* $\mathcal{A} \sqsubseteq \mathcal{A}^{\mathcal{V}\perp\!\!\!\perp^?}$ *then* $\mathcal{A} \sqsubseteq \mathcal{A}^{\perp\!\!\!\perp}$*.*

Proof. Let $v, e \in \mathcal{A}$. Since v and e must be strongly normalizing, any infinite reduction sequence starting from the command $\langle v \| e \rangle$ has the form (Lemma 1)

$$\langle v \| e \rangle \twoheadrightarrow \langle v' \| e' \rangle \rightsquigarrow c \qquad \text{where} \qquad v \twoheadrightarrow v' \qquad \text{and} \qquad e \twoheadrightarrow e'$$

such that and the remainder of the infinite reduction sequence continues from c (*i.e.,* $c \notin \perp\!\!\!\perp$). Since $\langle v' \| e' \rangle \rightsquigarrow c$ we know that at least one of v' or e' is a (co-)value (Property 4), and $\mathcal{A}$ is closed under reduction, we know that $v', e' \in \mathcal{A} = \mathcal{A}^{\mathcal{V}\perp\!\!\!\perp^?}$, which together implies that $\langle v' \| e' \rangle \not\rightsquigarrow c$: a contradiction. Therefore, there cannot be any infinite reduction sequence starting from $\langle v \| e \rangle$, *i.e.,* the command is strongly normalizing. □

This broad fact can be applied to derive other useful instances of expansion. For example, one variant is that $\perp\!\!\!\perp$ is closed under expansions of a neutral charge, even when dealing with non-deterministic (co-)terms.

Corollary 1 (Neutral Expansion)**.** *Given any admissible discipline* $\mathbf{s}$ *(even a non-deterministic one) and* $v, e \in \mathcal{W}_{\mathbf{s}}$*, if* $\langle v \| e \rangle \rightsquigarrow_0 c \in \perp\!\!\!\perp$ *then* $\langle v \| e \rangle \in \perp\!\!\!\perp$*.*

Proof. Note that the pre-candidate defined as

$$\mathcal{A} = (\{v' \mid v \twoheadrightarrow v'\}, \{e' \mid e \twoheadrightarrow e'\})$$

is closed under reduction and $\mathcal{A} \sqsubseteq \mathcal{A}^{\mathcal{V}\perp\!\!\!\perp^?}$ due to Property 1 and Lemma 3. Therefore $\mathcal{A} \sqsubseteq \mathcal{A}^{\perp\!\!\!\perp}$ by Lemma 7 and thus $\langle v \| e \rangle \in \perp\!\!\!\perp$. □

Similarly, the original form of the expansion property (Lemma 4) on deterministic disciplines is also a corollary of Lemma 7 with the same construction as above. This is the sense in which Lemma 7 strictly generalizes the simpler Lemma 4 in the face of non-confluence.

8.2. Defining Symmetric Candidates

In contrast with the previous reducibility candidates approach that revolves around symmetric operations like $\mathcal{A}^{\perp\!\!\!\perp}$ or $\mathcal{A}^{\mathcal{V}\perp\!\!\!\perp}$, we can define an asymmetric "saturation" operation which ensures that all the necessary (co-)terms are in a pre-candidate as dictated by the typing rules, but which may be overaggressive and include *too many* (co-)terms with non-sound (*i.e.,* non-normalizing) interactions with one another.

Definition 10 (Saturation)**.** The *saturation* of any pre-candidate $\mathcal{A}$ of $\mathbf{s}$ is:

$$\mathcal{A}^S \triangleq \mathcal{A}^{\mathcal{V}(\perp\!\!\!\perp^?_{+0},\perp\!\!\!\perp^?_{-0})}$$

Expanding the definitions, because both $\rightsquigarrow_{+0}$ and $\rightsquigarrow_{-0}$ are deterministic reduction relations (Property 1), the saturation of a pre-candidate $\mathcal{A}$ of $\mathbf{s}$ is:

$$\mathcal{A}^{S+} \triangleq \{v \in \mathcal{W}_{\mathbf{s}} \mid \forall E{\in}\mathcal{A}, \exists\, c{\in}\perp\!\!\!\perp, \langle v \| E\rangle \rightsquigarrow^?_{+0} c\}$$
$$\mathcal{A}^{S-} \triangleq \{e \in \mathcal{W}_{\mathbf{s}} \mid \forall V{\in}\mathcal{A}, \exists\, c{\in}\perp\!\!\!\perp, \langle V \| e\rangle \rightsquigarrow^?_{-0} c\}$$

The notion of saturation lets us give an alternative definition for "candidate" which has a more generous lower bound. That is, $\mathcal{A}^S$ extends $\mathcal{A}^{\mathcal{V}\perp\!\!\!\perp}$, which means that it is easier to show that a (co-)term is in $\mathcal{A}^S$ than in $\mathcal{A}^{\mathcal{V}\perp\!\!\!\perp}$. The difference is that with saturation, we only need to justify a (co-)term by its own behavior, and can ignore any behavior induced by its partner. For example, it is enough to know that $\langle \mu\alpha^{\mathbf{u}}\!\cdot c_2 \| \tilde{\mu}x^{\mathbf{u}}\!\cdot c_1\rangle \rightsquigarrow_{\tilde{\mu}_{\mathbf{u}}} c_1\{\mu\alpha^{\mathbf{u}}\!\cdot c_2/x^{\mathbf{u}}\} \in \perp\!\!\!\perp$ even if it's possible that $\langle \mu\alpha^{\mathbf{u}}\!\cdot c_2 \| \tilde{\mu}x^{\mathbf{u}}\!\cdot c_1\rangle \rightsquigarrow_{\mu_{\mathbf{u}}} c_2\{\tilde{\mu}x^{\mathbf{u}}\!\cdot c_1/\alpha^{\mathbf{u}}\} \notin \perp\!\!\!\perp$ as well.

Definition 11 (Symmetric Candidate)**.** A *symmetric candidate (of* $\mathbf{s}$*)* is any pre-candidate $\mathcal{A}$ (of $\mathbf{s}$) such that $\mathcal{A}^S \sqsubseteq \mathcal{A} \sqsubseteq \mathcal{A}^{\perp\!\!\!\perp}$. In other words, a symmetric candidate $\mathcal{A}$ is a pre-candidate satisfying the following properties:

(i) *Soundness* ($\mathcal{A} \sqsubseteq \mathcal{A}^{\perp\!\!\!\perp}$): For all $v, e \in \mathcal{A}$, the command $\langle v \| e\rangle$ is strongly normalizing.

(ii) *Completeness* ($\mathcal{A}^S \sqsubseteq \mathcal{A}$): If v is strongly normalizing and $\langle v \| E\rangle \rightsquigarrow^?_{+0} c$ where c is strongly normalizing for any $E \in \mathcal{A}$, then $v \in \mathcal{A}$. Dually, if e is strongly normalizing and $\langle V \| e\rangle \rightsquigarrow^?_{-0} c$ where c is strongly normalizing for any $V \in \mathcal{A}$, then $e \in \mathcal{A}$.

We denote the set of all symmetric candidates of $\mathbf{s}$ by CS.

As with reducibility candidates, symmetric candidates can also be equivalently rephrased as the fixed point of their completeness operation, $\mathcal{A}^S$.

Lemma 8. *For any admissible discipline* $\mathbf{s}$ *and pre-candidate* $\mathcal{A}$ *of* $\mathbf{s}$*, if* $\mathcal{A} = \mathcal{A}^S$ *then* $\mathcal{A} = \mathcal{A}^{\mathcal{V}\perp\!\!\!\perp^?}$.

Proof. Suppose $v, e \in \mathcal{A}$. First, note that there cannot be a step $\langle v \| e \rangle \rightsquigarrow_{+0} c$ where neither $\langle v \| e \rangle$ nor c is in $\perp\!\!\!\perp$ because then e must be a co-value of $\mathcal{A}$ (Property 4) which violates the assumption that $v \in \mathcal{A}^S = \mathcal{A}$. Similarly, there cannot be a step $\langle v \| e \rangle \rightsquigarrow_{-0} c$ where $\langle v \| e \rangle, c \notin \perp\!\!\!\perp$. Therefore, if either one of v or e is a (co-)value, then either $\langle v \| e \rangle \in \perp\!\!\!\perp$ immediately, or there is some c such that $\langle v \| e \rangle \rightsquigarrow c \perp\!\!\!\perp$, and there is no possible c' such that $\langle v \| e \rangle \rightsquigarrow c' \notin \perp\!\!\!\perp$. □

Property 12. *For any admissible discipline* $\mathbf{s}$ *and pre-candidate* $\mathcal{A}$ *of* $\mathbf{s}$*,* $\mathcal{A}$ *is a symmetric candidate of* $\mathbf{s}$ *if and only if* $\mathcal{A} = \mathcal{A}^S$.

We will return to this property shortly, and present the proof, when we compare symmetric candidates with reducibility candidates in Section 8.4.

8.3. Constructing Symmetric Candidates

Due to the potential for non-determinacy and the extra leniency of saturation, symmetric candidates are much more difficult to construct when compared with reducibility candidates. The first step in the construction is to isolate the terms and co-terms which still behave deterministically with respect to strong normalization ($\perp\!\!\!\perp$). Terms like $\mu[\boldsymbol{x} \, \textcircled{\scriptsize s} \, \boldsymbol{\alpha}].c$ and $V \, \textcircled{\scriptsize s} \, E$ can only participate in at most one standard reduction, so they can serve as part of the initial core of (co-)values that determine a candidate.

Definition 12 (Deterministic Normalization)**.** The set of commands for whom top reduction deterministically results in only normalizing or only non-normalizing commands is:

$$\perp\!\!\!\perp^d \triangleq \{c \mid \forall c_1, c_2.(c_1 \nleftarrow c \nrightarrow c_2) \implies (c_1 \in \perp\!\!\!\perp \iff c_2 \in \perp\!\!\!\perp)\}$$

For any discipline $\mathbf{s}$, the pre-candidate $\mathcal{D}_\mathbf{s}$ is the *deterministically-normalizing* terms and co-terms defined as $\mathcal{W}_\mathbf{s}^{\perp\!\!\!\perp^d}$. Expanding the definitions, $\mathcal{D}_\mathbf{s}$ consists of the following two sets:

$$\mathcal{D}^+ \triangleq \{v \in \mathcal{W}_\mathbf{s}^+ \mid \forall e \in \mathcal{W}_\mathbf{s}^-, (c_1 \nleftarrow \langle v \| e \rangle \nrightarrow c_2) \implies (c_1 \in \perp\!\!\!\perp \iff c_2 \in \perp\!\!\!\perp)\}$$
$$\mathcal{D}^- \triangleq \{e \in \mathcal{W}_\mathbf{s}^- \mid \forall v \in \mathcal{W}_\mathbf{s}^+, (c_1 \nleftarrow \langle v \| e \rangle \nrightarrow c_2) \implies (c_1 \in \perp\!\!\!\perp \iff c_2 \in \perp\!\!\!\perp)\}$$

Just like the (co-)value restriction, $\mathcal{A}^{\mathcal{V}}$, we will just write $\mathcal{A}^{\mathcal{D}}$ as shorthand for $\mathcal{A}^{\mathcal{D}_{\mathbf{s}}}$ (which is defined as $\mathcal{A} \sqcap \mathcal{D}_{\mathbf{s}}$ in Definition 8) when the discipline $\mathbf{s}$ is unambiguous from context.

The positively- and negatively-oriented constructions from Section 7.2 must be further restricted in terms of deterministically normalizing (co-)values. The bigger challenge is to complete the candidate to include all sensible terms and co-terms allowed by saturation. Just applying a final orthogonal, as in $\mathcal{R}(-)$, is no longer enough (Lengrand and Miquel, 2008), since that doesn't yield a fixed point of saturation. Instead, we can rely on the Knaster-Tarski fixed point theorem to provide such a solution, since saturation is defined in terms of pre-orthogonality, it is monotonic with respect to the subtyping relation on pre-candidates (Property 6). Since subtyping pre-candidates forms a complete lattice, we are guaranteed both a largest and a smallest such solution with respect to subtyping. These definitions are given as follows:

$$Pos_d(C) \triangleq (C, C^{\perp\!\!\!\perp\mathcal{DV}})^{\perp\!\!\!\perp\mathcal{DV}} \qquad \mathcal{S}_{\perp}(\mathcal{A}) \triangleq \bigwedge\{\mathcal{B} \mid \mathcal{B} \geq \mathcal{A} \sqcup \mathcal{B}^S\}$$

$$Neg_d(O) \triangleq (O^{\perp\!\!\!\perp\mathcal{DV}}, O)^{\perp\!\!\!\perp\mathcal{DV}} \qquad \mathcal{S}_{\top}(\mathcal{A}) \triangleq \bigvee\{\mathcal{B} \mid \mathcal{B} \leq \mathcal{A} \sqcup \mathcal{B}^S\}$$

Note that since this definition is based on subtyping instead of refinement, the "smallest" solution $\mathcal{S}_{\perp}(-)$ contains the least terms and the *most* co-terms, and dually the "largest" solution $\mathcal{S}_{\top}(-)$ contains the most terms and the *least* co-terms.

Lemma 9. *For any admissible discipline* $\mathbf{s}$ *and pre-candidates* $\mathcal{A}$ *and* $\mathcal{B}$ *of* $\mathbf{s}$*, if* $\mathcal{A} = \mathcal{A}^{\perp\!\!\!\perp\mathcal{DV}}$ *and* $\mathcal{B} = \mathcal{A} \sqcup \mathcal{B}^S$*, then* $\mathcal{B}$ *is a symmetric candidate of* $\mathbf{s}$*.*

Proof. Because $\mathcal{B}^S \sqsubseteq \mathcal{B}$ by assumption and the sets $\perp\!\!\!\perp$ and $\perp\!\!\!\perp^?$ are closed under reduction (Lemma 2), it suffices to show that $\mathcal{B} \sqsubseteq \mathcal{B}^{\mathcal{V}\perp\!\!\!\perp^?}$ due to Property 8 and Lemma 7. Let $v, E \in \mathcal{B}$, so that $\langle v \| E \rangle \in \perp\!\!\!\perp^?$ by one of the following cases:

- If $v, E \in \mathcal{A}$ then $\langle v \| E \rangle \in \perp\!\!\!\perp$ because $\mathcal{A} \sqsubseteq \mathcal{A}^{\perp\!\!\!\perp}$.
- If $v, E \in \mathcal{B}^S$ then $\langle v \| E \rangle \rightsquigarrow^?_{+0} c \in \perp\!\!\!\perp$, and if $\langle v \| E \rangle \rightsquigarrow_{-} c'$ then v is a value (Property 4) so it must be that $c' \in \perp\!\!\!\perp$.
- If $v \in \mathcal{B}^S$ and $E \in \mathcal{A}$ then $\langle v \| E \rangle \rightsquigarrow^?_{+0} c \in \perp\!\!\!\perp$, and if $\langle v \| E \rangle \rightsquigarrow c'$ then $c' \in \perp\!\!\!\perp$ too because $\mathcal{A} \sqsubseteq \mathcal{D}$.

- If $v \in \mathcal{A}$ and $E \in \mathcal{B}^S$ then v must be a value since $\mathcal{A} \sqsubseteq \mathcal{V}$, so that $\langle v \| E \rangle \rightsquigarrow^?_{-0} c \in \perp\!\!\!\perp$ and if $\langle v \| E \rangle \rightsquigarrow c'$ then $c' \in \perp\!\!\!\perp$ because $\mathcal{A} \sqsubseteq \mathcal{D}$.

Likewise, for every $V, e \in \mathcal{B}$, $\langle V \| e \rangle \in \perp\!\!\!\perp^?$. Therefore, $\mathcal{B} \sqsubseteq \mathcal{B}^{\mathcal{V}\perp\!\!\!\perp^?}$. □

Lemma 10. *For any admissible discipline* $\mathbf{s}$*, set* $\mathbf{s}$*-values of* C*, set of* $\mathbf{s}$*-co-values of* O*, and pre-candidate* $\mathcal{A} = \mathcal{A}^{\perp\!\!\!\perp\mathcal{D}\mathcal{V}}$ *of* $\mathbf{s}$*:*

(i) $Pos_d(C) = Pos_d(C)^{\perp\!\!\!\perp\mathcal{D}\mathcal{V}}$ *and* $(C, \{\}) \sqsubseteq Pos_d(C)$*,*

(ii) $Neg_d(O) = Neg_d(O)^{\perp\!\!\!\perp\mathcal{D}\mathcal{V}}$ *and* $(\{\}, O) \sqsubseteq Neg_d(O)$*, and*

(iii) there exists at least one symmetric candidate extending $\mathcal{A}$ *(*i.e.*, there is a symmetric candidate* $\mathcal{B}$ *such that* $\mathcal{A} \sqsubseteq \mathcal{B}$*), where* $\mathcal{S}_\perp(\mathcal{A})$ *is the smallest one and* $\mathcal{S}_\top(\mathcal{A})$ *is the largest one (with respect to* $\leq$*).*

Proof. Parts (i) and (ii) are analogous to Lemma 5. For part (iii), note that

$$\mathcal{S}_\top(\mathcal{A}) = \mathcal{A} \sqcup \mathcal{S}_\top(\mathcal{A})^S \qquad\qquad \mathcal{S}_\perp(\mathcal{A}) = \mathcal{A} \sqcup \mathcal{S}_\perp(\mathcal{A})^S$$

are the largest and smallest such fixed points via the Knaster-Tarski fixed point theorem since the operation $\mathcal{A} \sqcup -^S$ is monotonic with respect to subtyping (6). From Lemma 9, it follows that both are symmetric candidates. □

8.4. Symmetric Candidates versus Reducibility Candidates

We've alluded to the fact that symmetric candidates "generalize" reducibility candidates. This is well-known to be true in the weak sense that symmetric candidates are powerful enough to capture fundamentally non-deterministic standard reduction whereas reducibility candidates and biorthogonality apply to deterministic system. However, it is also true in the much stronger sense that the notions of symmetric candidate and reducibility candidate are the same for deterministic systems, such that the two constructions are exactly equal.

In terms of the fundamental operations that serve as the upper- and lower-bounds for both reducibility candidates and symmetric candidates, there is a natural ordering between them.

Property 13. *For any* $\mathbf{s}$ *and pre-candidate* $\mathcal{A}$ *of* $\mathbf{s}$*,* $\mathcal{A}^{\perp\!\!\!\perp} \sqsubseteq \mathcal{A}^{\mathcal{V}\perp\!\!\!\perp} \sqsubseteq \mathcal{A}^S$*. Furthermore, if* $\mathbf{s}$ *is deterministic and admissible, then* $\mathcal{A}^{\mathcal{V}\perp\!\!\!\perp} = \mathcal{A}^S$*.*

Proof. The first part follows from extension (Property 6) since $\perp\!\!\!\perp \subseteq \perp\!\!\!\perp_p^?$. The second part follows from deterministic expansion (Lemma 4) to show that $\mathcal{A}^S \sqsubseteq \mathcal{A}^{\mathcal{V}\perp\!\!\!\perp}$. □

Theorem 3. *For any admissible discipline* $\mathbf{s}$*, every symmetric candidate of* $\mathbf{s}$ *is a reducibility candidate of* $\mathbf{s}$*, and if* $\mathbf{s}$ *is also deterministic, then every reducibility candidate of* $\mathbf{s}$ *is a symmetric candidate of* $\mathbf{s}$*.*

Proof. Follows directly from Property 13 and Definitions 9 and 11. □

This ordering also lets us prove that both forms of candidates are exactly the same as the fixed points of their completeness operation (which define their lower bound).

Property 9. *For any deterministic admissible discipline* $\mathbf{s}$ *and pre-candidate* $\mathcal{A}$ *of* $\mathbf{s}$*,* $\mathcal{A}$ *is a reducibility candidate of* $\mathbf{s}$ *if and only if* $\mathcal{A} = \mathcal{A}^{\mathcal{V}\perp\!\!\!\perp}$*.*

Proof. Follows from Property 13 (to show that every reducibility candidate $\mathcal{A}$ is a fixed point $\mathcal{A} = \mathcal{A}^{\mathcal{V}\perp\!\!\!\perp} = \mathcal{A}^{\perp\!\!\!\perp}$) and from Lemma 7 (to show that every fixed point $\mathcal{A} = \mathcal{A}^{\mathcal{V}\perp\!\!\!\perp} \sqsubseteq \mathcal{A}^{\mathcal{V}\perp\!\!\!\perp^?}$ is a reducibility candidate). □

Property 12. *For any admissible discipline* $\mathbf{s}$ *and pre-candidate* $\mathcal{A}$ *of* $\mathbf{s}$*,* $\mathcal{A}$ *is a symmetric candidate of* $\mathbf{s}$ *if and only if* $\mathcal{A} = \mathcal{A}^S$*.*

Proof. Follows from Property 13 (to show that every symmetric candidate $\mathcal{A}$ is a fixed point $\mathcal{A} = \mathcal{A}^S = \mathcal{A}^{\mathcal{V}\perp\!\!\!\perp} = \mathcal{A}^{\perp\!\!\!\perp}$) and from Lemma 9 (to show that every fixed point $\mathcal{A} = \mathcal{A}^S$ is a symmetric candidate). □

As a consequence, since reducibility candidates are unique up to their values (Property 11), then so too are symmetric candidates of deterministic disciplines. That means that for any deterministic admissible $\mathbf{s}$ and $\mathcal{A} = \mathcal{A}^{\perp\!\!\!\perp\mathcal{V}}$ of $\mathbf{s}$, then $\mathcal{S}_\perp(\mathcal{A}) = \mathcal{S}_\top(\mathcal{A}) = \mathcal{R}(\mathcal{A})$ is the unique reducibility/symmetric candidate extending $\mathcal{A}$. As such, there is no loss of information in using the symmetric candidate construction over the reducibility candidate (or biorthogonality) construction: in the end, they will amount to exactly the same as the simplest and most-specific construction depending on the discipline $\mathbf{s}$. This fact is also interesting because, in general for a non-deterministic discipline, like $\mathbf{u}$, we do not have a similar uniqueness guarantee, and do not know whether $\mathcal{S}_\perp(\mathcal{A}) = \mathcal{S}_\top(\mathcal{A})$ for $\mathbf{u}$.

9. A Uniform Model of Strong Normalization

The final step is to interpret syntactic types as symmetric candidates (which are the same thing as reducibility candidates for deterministic disciplines), and typing judgements as logical statements. At this point, these following definitions are standard, as per the approaches of logical relations and biorthogonality.

9.1. Interpreting Types, Judgements, and Rules

The interpretation of types is parameterized by a product of functions, $\theta : \Pi_{\mathbf{s}}(\mathit{TypeVar} \to \mathit{CS}_{\mathbf{s}})$, from type variables to symmetric candidates of the appropriate discipline, and is defined as follows:

$$\begin{aligned}
[\![a^{\mathbf{s}}]\!]_\theta &\triangleq \theta_{\mathbf{s}}(a) \\
[\![A \xrightarrow{\mathbf{s}} B]\!]_\theta &\triangleq \mathcal{S}_\top(\mathit{Neg}\{V \bullet_{\mathbf{s}} E \mid V \in [\![A]\!]_\alpha, E \in [\![B]\!]_\beta\}) \\
[\![\forall^{\mathbf{s}} a^{\mathbf{t}}.B]\!]_\theta &\triangleq \mathcal{S}_\top(\mathit{Neg}\{A_{\mathbf{t}} \bullet_{\mathbf{s}} E \mid \mathcal{A} \in \mathit{CS}_{\mathbf{t}}, E \in [\![B]\!]_{\theta, \mathcal{A}/a^{\mathbf{t}}}\})
\end{aligned}$$

where $\theta, \mathcal{A}/a^{\mathbf{t}}$ denotes the usual extension of the $\mathbf{t}$-component of θ:

$$\begin{aligned}
(\theta, \mathcal{A}/a^{\mathbf{t}})_{\mathbf{t}}(a) &= \mathcal{A} \\
(\theta, \mathcal{A}/a^{\mathbf{t}})_{\mathbf{s}}(b) &= \theta_{\mathbf{s}}(b) \qquad\qquad \text{if } \mathbf{s} \neq \mathbf{t} \text{ or } b \neq a
\end{aligned}$$

The interpretation of entire typing judgements then follows the interpretation of individual types as usual:

$$\begin{aligned}
[\![\Theta]\!] &\triangleq \Pi_{\mathbf{s}}(\mathit{TypeVar} \to \mathit{symcans}_{\mathbf{s}}) \\
[\![\Gamma \vdash \Delta]\!]_\theta &\triangleq \{\sigma \in \mathit{Subst} \mid \forall x{:}A \in \Gamma, x\{\sigma\} \in [\![A]\!]_\theta\} \\
&\quad \cap \{\sigma \in \mathit{Subst} \mid \forall \alpha{:}A \in \Delta, \alpha\{\sigma\} \in [\![A]\!]_\theta\} \\
[\![c : (\Gamma \vdash_\Theta \Delta)]\!] &\triangleq \forall \theta \in [\![\Theta]\!].\forall \sigma \in [\![\Gamma \vdash \Delta]\!]_\theta.c\{\sigma\} \in \perp\!\!\!\perp \\
[\![\Gamma \vdash_\Theta v : A \mid \Delta]\!] &\triangleq \forall \theta \in [\![\Theta]\!].\forall \sigma \in [\![\Gamma \vdash \Delta]\!]_\theta.v\{\sigma\} \in [\![A]\!]_\theta \\
[\![\Gamma \mid e : A \vdash_\Theta \Delta]\!] &\triangleq \forall \theta \in [\![\Theta]\!].\forall \sigma \in [\![\Gamma \vdash \Delta]\!]_\theta.e\{\sigma\} \in [\![A]\!]_\theta
\end{aligned}$$

In addition, we also give an interpretation of inference rules of the form

$$\frac{H_1 \quad \ldots \quad H_n}{J}\ \mathit{Rule}$$

as the statement

$$[\![Rule]\!] \triangleq [\![H_1]\!] \wedge \ldots \wedge [\![H_n]\!] \implies [\![J]\!]$$

and say that *Rule* is *sound* when $[\![Rule]\!]$ is true. That way, the interpretation of any derivation's conclusion must be true if it was derived from only sound inference rules.

Lemma 11. *For any judgement J derivable from sound rules, $[\![J]\!]$ is true.*

Proof. By induction on the given derivation tree. □

9.2. Adequacy and Strong Normalization

Having interpreted the meaning of types and judgements, it now remains to show that the inference rules of our type system are indeed sound with respect to the interpretation, or in other words, the interpretation is *adequate*. Since we are dealing with an open property—reduction of *any* sub-expression, even under binders—we will need to know that the presence of free variables does not cause any harm. In particular, it is important to note that strong normalization does not change in a command made from a free (co-)variable versus the underlying term or co-term. This, in turn, tells us that (co-)variables inhabit every type: a usual fact one proves on the road to strong normalization.

Lemma 12. *$\langle v \| \boldsymbol{\alpha} \rangle$ is strongly normalizing if and only if v is, and $\langle \boldsymbol{x} \| e \rangle$ is strongly normalizing if and only if e is.*

Proof. We will only show the first part, as the second is exactly dual. If $\langle v \| \boldsymbol{\alpha} \rangle$ is strongly normalizing then so is v because it is a sub-term of the command. The other implication is more interesting, as it means that (nearly) every reduction of $\langle v \| \boldsymbol{\alpha} \rangle$ can be done on v alone. In particular, the only counter-example is an infinite reduction sequence of the usual form (Lemma 1) $\langle v \| \boldsymbol{\alpha} \rangle \twoheadrightarrow \langle v' \| \boldsymbol{\alpha} \rangle \rightsquigarrow c$ where $v \twoheadrightarrow v'$. Without loss of generality, we can assume that $\langle v' \| \boldsymbol{\alpha} \rangle \rightsquigarrow_\mu c$ since a top ς reduction can be considered internal to the term-side, and no other top reduction is possible. Therefore we can assume (up to α-renaming) that $v' = \mu\boldsymbol{\alpha}.c$ so that $\langle \mu\boldsymbol{\alpha}.c \| \alpha \rangle \rightsquigarrow_\mu c$. But v' is strongly normalizing since it is a reduct of v and so c is strongly normalizing since it is a sub-command of v'. This contradicts the assumption that there is an infinite reduction sequence beginning from c, so $\langle v \| \boldsymbol{\alpha} \rangle$ must be strongly normalizing. □

Lemma 13. *Every symmetric candidate $\mathcal{A}$ of $\mathbf{s}$ contains all variables $x^{\mathbf{s}}$ and co-variables $\alpha^{\mathbf{s}}$.*

Proof. From antitonicity of $\mathcal{A} \sqsubseteq \mathcal{W}_{\mathbf{s}}$ we have $\mathcal{W}_{\mathbf{s}}^{\perp\!\!\!\perp} \sqsubseteq \mathcal{A}^{\perp\!\!\!\perp} = \mathcal{A}$ (Property 12), and from Lemma 12, we know that $x^{\mathbf{s}}, \alpha^{\mathbf{s}} \in \mathcal{W}_{\mathbf{s}}^{\perp\!\!\!\perp}$. □

We can now prove that each individual inference rule of the type system is sound. Due to the presence of quantifiers (namely $\forall$), we will also rely on a typical substitution lemma stating that syntactic substitution is interchangeable with substitution in the environment.

Lemma 14. $[\![A\{B_{\mathbf{s}}/b^{\mathbf{s}}\}]\!]_\theta = [\![A]\!]_{\theta,[\![B_{\mathbf{s}}]\!]_\theta/b^{\mathbf{s}}}$

Proof. By induction on the definition of $[\![A]\!]_\theta$. □

Lemma 15 (Inference Soundness)**.** *For any collection of admissible disciplines, every inference rule in Fig. 6 is sound.*

Proof.

- *Cut*: Suppose $[\![\Gamma \vdash_\Theta v : A \mid \Delta]\!]$ and $[\![\Gamma \mid e : A \vdash_\Theta \Delta]\!]$, and let $\theta \in [\![\Theta]\!]$ and $\sigma \in [\![\Gamma \vdash \Delta]\!]_\Theta$. We know from the assumptions that $v\{\theta\}, e\{\theta\} \in [\![A]\!]_\theta$ and, since $[\![A]\!]_\theta \sqsubseteq [\![A]\!]_\theta{}^{\perp\!\!\!\perp}$, $\langle v \| e \rangle\{\theta\} = \langle v\{\theta\} \| e\{\theta\} \rangle \in \perp\!\!\!\perp$ as required.

- *VarL*: For any $\theta \in [\![\Theta]\!]$ and $\sigma \in [\![\Gamma, x : A \vdash \Delta]\!]_\theta$, we know that $x\{\sigma\} \in [\![A]\!]_\theta$ by definition.

- *VarL*: Dual to the case for *VarR*.

- *ActR*: Suppose $[\![c : (\Gamma \vdash_\Theta \alpha^{\mathbf{s}} : A_{\mathbf{s}}, \Delta)]\!]$ and let $\theta \in [\![\Theta]\!]$ and $\sigma \in [\![\Gamma \vdash \Delta]\!]_\Theta$. First, note that $\mu\alpha^{\mathbf{s}}\!\!:\!\!c\{\sigma\} \in \mathcal{W}_{\mathbf{s}}$ because $\alpha^{\mathbf{s}} \in [\![A_{\mathbf{s}}]\!]_\theta$ (Lemma 13) which implies that $c\{\sigma\} = c\{\sigma, \alpha^{\mathbf{s}}/\alpha^{\mathbf{s}}\} \in \perp\!\!\!\perp$. Second, it follows that for any $E \in [\![A_{\mathbf{s}}]\!]_\theta$, $\langle \mu\alpha^{\mathbf{s}}\!\!:\!\!c\{\sigma\} \| E \rangle \rightsquigarrow_+ c\{\sigma, E/\alpha^{\mathbf{s}}\} \in \perp\!\!\!\perp$ and so $\mu\alpha^{\mathbf{s}}\!\!:\!\!c\{\sigma\} \in [\![A_{\mathbf{s}}]\!]_\theta{}^S \sqsubseteq [\![A_{\mathbf{s}}]\!]_\theta$ by definition.

- *ActL*: Dual to the case for *ActR*.

- $\forall L$: Suppose $[\![\Gamma \mid e : B\{A_{\mathbf{t}}/a^{\mathbf{t}}\} \vdash_\Theta \Delta]\!]$ and let $\theta \in [\![\Theta]\!]$ and $\sigma \in [\![\Gamma \vdash \Delta]\!]_\theta$. We can now proceed by cases on whether e is a co-value:

– If e is a co-value, then note that $A_{\mathbf{t}} \mathbin{⦾} e \in \mathcal{W}_{\mathbf{s}}$ because and e is strongly normalizing (where the only possible reductions are within e), and that

$$(A_{\mathbf{t}} \mathbin{⦾} e)\{\sigma\} = A_{\mathbf{t}}\{\sigma\} \mathbin{⦾} e\{\sigma\} \in \{A'_{\mathbf{t}} \mathbin{⦾} E \mid \mathcal{A} \in CS_{\mathbf{t}}, E \in [\![B]\!]_{\theta,\mathcal{A}/a^{\mathbf{t}}}\}$$

because $[\![B\{A_{\mathbf{t}}/a^{\mathbf{t}}\}]\!]_\theta = [\![B]\!]_{\theta,[\![A]\!]_\theta/a^{\mathbf{t}}}$ (Lemma 14) and so we have $(A_{\mathbf{s}} \mathbin{⦾} e)\{\sigma\} \in [\![\forall^{\mathbf{s}} a^{\mathbf{t}}{:}B]\!]_\theta$ by Lemma 5.

– If e is not a co-value, then note that for any $V \in [\![\forall[\mathbf{s}]a[\mathbf{t}]B]\!]_\theta$,

$$\langle V \| (A_{\mathbf{t}} \mathbin{⦾} e)\{\sigma\}\rangle \rightsquigarrow_{-} \langle V \| \tilde{\mu} x^{\mathbf{s}}{:}\langle \mu\boldsymbol{\beta}.\langle x \| A_{\mathbf{t}}\{\sigma\} \mathbin{⦾} \boldsymbol{\beta}\rangle \| e\{\sigma\}\rangle\rangle \in \perp\!\!\!\perp$$

because $\tilde{\mu} x^{\mathbf{s}}{:}\langle \mu\boldsymbol{\beta}.\langle x \| A_{\mathbf{t}}\{\sigma\} \mathbin{⦾} \boldsymbol{\beta}\rangle \| e\{\sigma\}\rangle \in [\![\forall^{\mathbf{s}} a^{\mathbf{t}}{:}B]\!]_\theta$ by applying the interpretations of the *Cut*, *VarL*, *VarR*, *ActL*, *ActR* rules, and the case of $\forall L$ on a co-value, already proved sound above.

- $\forall R$: Suppose $[\![c : (\Gamma \vdash_{\Theta, a^{\mathbf{t}}} \beta^{\mathbf{r}} : B_{\mathbf{r}}, \Delta)]\!]$ and let $\theta \in [\![\Theta]\!]$ and $\sigma \in [\![\Gamma \vdash \Delta]\!]_\theta$. First, note that $\mu[a^{\mathbf{t}} \mathbin{⦾} \beta^{\mathbf{r}}].c \in \mathcal{W}_{\mathbf{s}}$ because $\beta^{\mathbf{r}} \in [\![B_{\mathbf{r}}]\!]_{\theta'}$ for all θ' (Lemma 13) which implies that $c\{\sigma\} = c\{\sigma, a^{\mathbf{t}}/a^{\mathbf{t}}, \beta^{\mathbf{r}}/\beta^{\mathbf{r}}\} \in \perp\!\!\!\perp$. Second, it follows that for any type $A_{\mathbf{t}}$, symmetric candidate $\mathcal{A}$ of $\mathbf{t}$, and co-value $E \in [\![B]\!]_{\theta,\mathcal{A}/a^{\mathbf{t}}}$, we have the extension $\sigma, A_{\mathbf{t}}/a^{\mathbf{t}}, E/\beta^{\mathbf{r}} \in [\![\Gamma \vdash \beta^{\mathbf{r}} : B_{\mathbf{r}}, \Delta]\!]_{\theta,\mathcal{A}/a^{\mathbf{t}}}$ from the side-condition that $a^{\mathbf{t}} \notin FV(\Gamma \vdash_\Theta \Delta)$, which leads to

$$\langle \mu[a^{\mathbf{t}} \mathbin{⦾} \beta^{\mathbf{r}}].c\{\sigma\} \| A_{\mathbf{t}} \mathbin{⦾} E\rangle \rightsquigarrow_0 c\{\sigma, A_{\mathbf{t}}/a^{\mathbf{t}}, E/\beta^{\mathbf{r}}\} \in \perp\!\!\!\perp$$

and so $\mu[a^{\mathbf{t}} \mathbin{⦾} \beta^{\mathbf{r}}].c\{\sigma\} \in [\![\forall^{\mathbf{s}} a^{\mathbf{t}}{:}B]\!]_\theta$ by Corollary 1 and Lemma 10.

- $\to L$ and $\to R$: Similar to the cases for $\forall L$ and $\forall R$, respectively. □

Since each individual inference rule is sound, we have adequacy for the entire model, which means that typing implies strong normalization.

Corollary 2 (Adequacy)**.** *For any collection of admissible disciplines:*

1. *if* $c : (\Gamma \vdash_\Theta \Delta)$ *is derivable then* $[\![c : (\Gamma \vdash_\Theta \Delta)]\!]$ *is true,*
2. *if* $\Gamma \vdash v : A \mid \Delta$ *is derivable, then* $[\![\Gamma \vdash v : A \mid \Delta]\!]$ *is true, and*
3. *if* $\Gamma \mid e : A \vdash \Delta$ *is derivable then* $[\![\Gamma \mid e : A \vdash \Delta]\!]$ *is true.*

Proof. From Lemmas 11 and 15. □

Theorem 4 (Strong Normalization)**.** *Every well-typed command, term, and co-term is strongly normalizing under any collection of admissible disciplines.*

Proof. An application of adequacy (Corollary 2) to an initial mapping of type variables, $\theta_{\mathbf{s}}(a) = \mathcal{S}_{\top}(Neg\{\})$ for every $\mathbf{s}$ and a, and the identity substitution of (co-)variables since they are present in every symmetric candidate (Lemma 13). □

10. Conclusion

We have explored multi-discipline calculi with polymorphism and control, based on the sequent calculus. The sequent calculus setting is good for exploring multi-discipline programming since it provides a clean separation between the different disciplines and allows us to treat them abstractly as an object of study. As our main objective, we established strong normalization by using a model of types based on both orthogonality and fixed points. Our model is uniform over multiple disciplines, with a generic characterization of which ones are admissible, and strictly generalizes several previous models. This study illustrates the benefits of both the sequent calculus and discipline-agnostic reasoning: we can give a single explanation for several calculi in one fell swoop and without losing anything from the discipline-specific models. Our setting of pre-candidates already comes with a built-in notion of subtyping along with the union and intersection of types, it would be interesting to relate these ideas to filter models and the characterization of strong normalization in terms of dual intersection and union types. More practically, we would like to relate our formal study of mixing disciplines to the way current languages combine strict and lazy features, with an ultimate aim of improving multi-disciplined programming and compilation.

Acknowledgments This work is supported by the National Science Foundation under grants CCF-1719158 and CCF-1423617.

References

Ariola, Z.M., Herbelin, H., Saurin, A., 2011. Classical call-by-need and duality, in: Typed Lambda Calculi and Applications - 10th International Conference, TLCA 2011, Novi Sad, Serbia, June 1-3, 2011. Proceedings, pp. 27–44. doi:10.1007/978-3-642-21691-6_6.

Barbanera, F., Berardi, S., 1994. A symmetric lambda calculus for "classical" program extraction, in: Theoretical Aspects of Computer Software, International Conference TACS '94, Sendai, Japan, April 19-22, 1994, Proceedings, pp. 495–515. doi:10.1007/3-540-57887-0_112.

Curien, P., Herbelin, H., 2000. The duality of computation, in: Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00), Montreal, Canada, September 18-21, 2000., pp. 233–243. doi:10.1145/351240.351262.

Curien, P., Munch-Maccagnoni, G., 2010. The duality of computation under focus, in: Theoretical Computer Science - 6th IFIP TC 1/WG 2.2 International Conference, TCS 2010, Held as Part of WCC 2010, Brisbane, Australia, September 20-23, 2010. Proceedings, pp. 165–181. doi:10.1007/978-3-642-15240-5_13.

David, R., Nour, K., 2005. Why the usual candidates of reducibility do not work for the symmetric $\lambda\mu$-calculus. Electronic Notes in Theoretical Computer Science 140, 101 – 111. URL: http://www.sciencedirect.com/science/article/pii/S1571066105051303, doi:https://doi.org/10.1016/j.entcs.2005.06.020. proceedings of the Second Workshop on Computational Logic and Applications (CLA 2004).

Downen, P., 2017. Sequent Calculus: A Logic and a Language for Computation and Duality. Ph.D. thesis. University of Oregon.

Downen, P., Ariola, Z.M., 2014. The duality of construction, in: Programming Languages and Systems - 23rd European Symposium on Programming, ESOP 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings, pp. 249–269. doi:10.1007/978-3-642-54833-8_14.

Downen, P., Ariola, Z.M., 2019. The duality of classical intersection and union types. Fundamenta Informaticae 170, 1–54.

Downen, P., Johnson-Freyd, P., Ariola, Z.M., 2015. Structures for structural recursion, in: Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming, ICFP 2015, Vancouver, BC, Canada, September 1-3, 2015, pp. 127–139. doi:10.1145/2784731.2784762.

Downen, P., Johnson-Freyd, P., Ariola, Z.M., 2018. Uniform strong normalization for multi-discipline calculi, in: Rewriting Logic and Its Applications - 12th International Workshop, WRLA 2018, Held as a Satellite Event of ETAPS, Thessaloniki, Greece, June 14-15, 2018, Proceedings, pp. 205–225. doi:10.1007/978-3-319-99840-4_12.

Gentzen, G., 1935. Untersuchungen über das logische schließen. I. Mathematische Zeitschrift 39, 176–210.

Girard, J., 1987. Linear logic. Theoretical Computer Science 50, 1–102. doi:10.1016/0304-3975(87)90045-4.

Girard, J.Y., Taylor, P., Lafont, Y., 1989. Proofs and Types. Cambridge University Press, New York, NY, USA.

Herbelin, H., Zimmermann, S., 2009. An operational account of call-by-value minimal and classical lambda-calculus in "natural deduction" form, in: Typed Lambda Calculi and Applications, 9th International Conference, TLCA 2009, Brasilia, Brazil, July 1-3, 2009. Proceedings, pp. 142–156. doi:10.1007/978-3-642-02273-9_12.

Krivine, J., 2007. A call-by-name lambda-calculus machine. Higher-Order and Symbolic Computation 20, 199–207. doi:10.1007/s10990-007-9018-9.

Krivine, J.L., 2005. Realizability in classical logic, in: Interactive models of computation and program behaviour. Société Mathématique de France. volume 27.

Lengrand, S., Miquel, A., 2008. Classical F_ω, orthogonality and symmetric candidates. Annals of Pure and Applied Logic 153, 3–20. doi:10.1016/j.apal.2008.01.005.

Levy, P.B., 2001. Call-By-Push-Value. Ph.D. thesis. University of London.

Liskov, B., 1987. Keynote address-data abstraction and hierarchy, in: Addendum to the Proceedings on Object-oriented Programming Systems, Languages and Applications (Addendum), ACM, New York, NY, USA. pp. 17–34. doi:10.1145/62138.62141.

Mellies, P.A., Vouillon, J., 2005. Recursive polymorphic types and parametricity in an operational framework, in: Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science, IEEE Computer Society, Washington, DC, USA. pp. 82–91. doi:10.1109/LICS.2005.42.

Miquey, É., Herbelin, H., 2018. Realizability interpretation and normalization of typed call-by-need λ-calculus with control, in: Baier, C., Dal Lago, U. (Eds.), Foundations of Software Science and Computation Structures, Springer International Publishing, Cham. pp. 276–292.

Munch-Maccagnoni, G., 2009. Focalisation and classical realisability, in: Computer Science Logic, 23rd international Workshop, CSL 2009, 18th Annual Conference of the EACSL, Coimbra, Portugal, September 7-11, 2009. Proceedings, pp. 409–423. doi:10.1007/978-3-642-04027-6_30.

Munch-Maccagnoni, G., 2013. Syntax and Models of a non-Associative Composition of Programs and Proofs. Ph.D. thesis. Université Paris Diderot.

Peyton Jones, S., 2009. https://www.red-gate.com/simple-talk/opinion/geek-of-the-week/simon-peyton-jones-geek-of-the-week/.

Peyton Jones, S.L., Launchbury, J., 1991. Unboxed values as first class citizens in a non-strict functional language, in: Proceedings of the 5th ACM Conference on Functional Programming Languages and Computer Architecture, Springer-Verlag, London, UK, UK. pp. 636–666.

Pitts, A.M., 2000. Parametric polymorphism and operational equivalence. Mathematical Structures in Computer Science 10, 321–359. doi:10.1017/S0960129500003066.

Plotkin, G.D., 1975. Call-by-name, call-by-value and the lambda-calculus. Theoretical Computer Science 1, 125–159. doi:10.1016/0304-3975(75)90017-1.

Ronchi Della Rocca, S., Paolini, L., 2004. The Parametric λ-Calculus: a Metamodel for Computation. Springer-Verlag.

Sabry, A., Felleisen, M., 1992. Reasoning about programs in continuation-passing style, in: LFP'92, pp. 288–298.

Tait, W.W., 1967. Intensional interpretations of functionals of finite type I. Journal of Symbolic Logic 32, 198–212. doi:10.2307/2271658.

Turbak, F., Gifford, D., Sheldon, M.A., 2008. Design Concepts in Programming Languages. The MIT Press.

Wadler, P., 2003. Call-by-value is dual to call-by-name, in: Proceedings of the eighth ACM SIGPLAN international conference on Functional programming, pp. 189–201. doi:10.1145/944705.944723.

Wright, A.K., Felleisen, M., 1994. A syntactic approach to type soundness. Information and Computation 115, 38–94. doi:10.1006/inco.1994.1093.

Zeilberger, N., 2009. The Logical Basis of Evaluation Order and Pattern-Matching. Ph.D. thesis. Carnegie Mellon University.