Uniform Strong Normalization for Multi-Discipline Calculi

Paul Downen, Philip Johnson-Freyd, and 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.

1 Introduction

Picking a programming language means choosing not just a concrete syntax and set of features, but also a calling convention. As Simon Peyton Jones [19] 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 [24,10]. 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 [30,18] and call-by-push-value [15] 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.

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 *et al.* [23], calculi for different calling conventions can be summarized as instances of a common calculus parameterized by a *substitution discipline* [4] 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 $\lambda x.Vx = V$; 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 [1] and the non-deterministic evaluation of the symmetric λ-calculus [2].

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. Currently, there are separate proofs of strong normalization for calculi of different 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 [5], which is based on both biorthogonal [7,13,21] and symmetric candidate [2] models, and extend it to accommodate multi-discipline languages. Furthermore, 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 [13,21]. 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 uses a sequent calculus with impredicative polymorphism based on [5] and 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 (Section 4). Our contributions are:

– A uniform proof of strong normalization based on orthogonality and symmetric candidates that parametrically accounts for multiple disciplines (Section 5).
– A more precise model than [5] which subsumes biorthogonality models for call-by-name, -value, and -push-value as special instances, and the first proof of strong normalization for multi-discipline call-by-need and its dual (Section 6).

The proofs for Sections 4 to 6 can be found in Appendices D to F. As an application of this work, a polymorphic $\lambda\mu$-calculus that mixes call-by-value, -name, and -need can be found in Appendices B and C, along with a derived proof of strong normalization by translation in Appendix G.

2 A Language 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 $\Box\ 2$, where $\Box$ 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 [12], which requires only two rules:

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

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 we assume the term is evaluated in a context named α:

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

So the machine returns 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 $\langle \Box \| v \cdot e\rangle$ 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 ordinary λ calculus application are two ways of getting at the same concept. It is thus natural to wonder if the redundancy can be eliminated by unifying the two.

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. The next is to abstract over co-variables like α. That is the role of the μ-abstraction, written as $\mu\alpha.c$, which is reduced like so:

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

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 $\mu\beta.\langle \lambda x.\lambda y.x + y \| 1 \cdot 2 \cdot \beta\rangle$. Note that this term behaves the same as the original one:

$$\langle \mu\beta.\langle \lambda x.\lambda y.x + y \| 1 \cdot 2 \cdot \beta\rangle \| \alpha\rangle \to \langle \lambda x.\lambda y.x + y \| 1 \cdot 2 \cdot \alpha\rangle$$

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

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:

$$\langle \mu\beta.\langle \lambda x.\lambda y.x + y \| \alpha\rangle \| 1 \cdot 2 \cdot \alpha\rangle \to \langle \lambda x.\lambda y.x + y \| \alpha\rangle$$

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 $V\ \square$ (where V denotes a value: a variable or a λ-abstraction) in addition to $\square\ v$. 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 following three reduction rules:

$$\langle v\ v' \| e\rangle \to \langle v \| v' \cdot e\rangle \quad \langle V \| v' \cdot e\rangle \to \langle v' \| V \circ e\rangle \quad \langle V' \| (\lambda x.v) \circ e\rangle \to \langle v\{V'/x\} \| e\rangle$$

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:

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

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—$\langle(\lambda x.v)\ v'\|e\rangle$, $\langle\lambda x.v\|v'\cdot e\rangle$, 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 [3], 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 $\tilde{\mu}x.\langle v\|x\cdot e\rangle$. Expanding this notational definition, the second rule is:

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

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

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

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 [17]. 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:

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

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

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

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 [3]! 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 allow one to focus on an assumption or conclusion.

3 Substitution Disciplines

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 [3]:

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

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 [22], 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 [26]. 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 [23,4]; 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 [1].

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 [20], 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 **v** (for call-by-value) and **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 **n** variable x, on the left we end up with a **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

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

Fig. 1. Syntax of a multi-discipline, polymorphic sequent calculus.

β reduction does not apply to $\langle \lambda x.v \| v' \cdot e \rangle$ when v' is not a value. This is done by *lifting* v' out of the call-stack [4]: $\langle \lambda x^{\mathbf{v}}.v \| v' \cdot e \rangle \to \langle \lambda x^{\mathbf{v}}.v \| \tilde{\mu} x.\langle v' \| \tilde{\mu} y.\langle x \| y \cdot e \rangle \rangle \rangle$. 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.

Syntax As in the abstract machine language of Section 2, the syntax of our calculus is 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 $x \cdot \alpha$. This change of notation is syntactic in nature—note that $\lambda x.v$ is equivalent to $\mu(x \cdot \alpha).\langle v \| \alpha \rangle$—which helps to emphasize the role of functions as responders 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.

The second thing to notice about the syntax is that terms and co-terms are divided by their discipline as discussed in Section 3, 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 aligns with the 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 $\mathit{Co\text{-}Term}_{\mathbf{s}}$. 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

$$\langle \mu\boldsymbol{\alpha}.c \| E\rangle \succ_{\mu} c\{E/\boldsymbol{\alpha}\} \qquad \langle V \| \tilde{\mu}\boldsymbol{x}.c\rangle \succ_{\tilde{\mu}} c\{V/\boldsymbol{x}\} \qquad \mu\boldsymbol{\alpha}.\langle v\|\boldsymbol{\alpha}\rangle \succ_{\eta_\mu} v \qquad \tilde{\mu}\boldsymbol{x}.\langle \boldsymbol{x}\|e\rangle \succ_{\eta_{\tilde{\mu}}} e$$

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

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

$$v_{\mathbf{t}} \textcircled{s} e \succ_{\varsigma^{\to}} \tilde{\mu}x^{\mathbf{s}}.\langle v_{\mathbf{t}} \| \tilde{\mu}y^{\mathbf{t}}.\langle x^{\mathbf{s}} \| y^{\mathbf{t}} \textcircled{s} e\rangle\rangle \qquad (\nexists V_{\mathbf{t}}, v_{\mathbf{t}} = V_{\mathbf{t}})$$

$$V \textcircled{s} e_{\mathbf{r}} \succ_{\varsigma^{\to}} \tilde{\mu}x^{\mathbf{s}}.\langle \mu\beta^{\mathbf{r}}.\langle x^{\mathbf{s}} \| V \textcircled{s} \beta^{\mathbf{s}_2}\rangle \| e_{\mathbf{r}}\rangle \qquad (\nexists E_{\mathbf{r}}, e_{\mathbf{r}} = E_{\mathbf{r}})$$

$$A \textcircled{s} e_{\mathbf{r}} \succ_{\varsigma^{\forall}} \tilde{\mu}x^{\mathbf{s}}.\langle \mu\beta^{\mathbf{r}}.\langle x^{\mathbf{s}} \| A \textcircled{s} \beta^{\mathbf{s}}\rangle \| e_{\mathbf{r}}\rangle \qquad (\nexists E_{\mathbf{r}}, e_{\mathbf{r}} = E_{\mathbf{r}})$$

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

Fig. 2. Rewriting theory for multi-discipline, polymorphic sequent calculus.

belong to exactly one syntactic category $\mathit{Term}_{\mathbf{s}}$ and $\mathit{Co\text{-}Term}_{\mathbf{s}}$, the call-stack dot is also annotated with a discipline symbol. That way, it is immediately apparent that $v \textcircled{s} e$ is an $\mathbf{s}$ co-term and $\mu(\boldsymbol{x} \textcircled{s} \boldsymbol{\alpha}).c$ is an $\mathbf{s}$ term. For example, a wholly call-by-value function can be written as $\mu(x^{\mathbf{v}} \textcircled{v} \alpha^{\mathbf{v}}).c$ that matches a call-stack of the form $v_{\mathbf{v}} \textcircled{v} e_{\mathbf{v}}$. The $\mathbf{v}$ in the $\textcircled{v}$ 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 [15] and polarized languages [30] would have the form $\mu(x^{\mathbf{v}} \textcircled{n} \alpha^{\mathbf{n}}).c$ and $v_{\mathbf{v}} \textcircled{n} e_{\mathbf{n}}$, with a call-by-value argument and call-by-name function and result.

Parameterized Reduction Theory The reduction theory, denoted by $\to$ shown in Fig. 2, 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.

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 $\mathbf{s}$. For example, we write $\langle \mu\boldsymbol{\alpha}.c\|E\rangle \to_{\mu} c\{E/\boldsymbol{\alpha}\}$ and by the syntactic requirement that the two sides of a command agree on a discipline, it must be that the disciplines of E and $\boldsymbol{\alpha}$ match. Disciplines are not just restrictive but also enabling in the case of the ς rules (originally due to

$$
\begin{aligned}
V_{\mathbf{u}} &::= v_{\mathbf{u}} & E_{\mathbf{u}} &::= e_{\mathbf{u}} \\
V_{\mathbf{v}} &::= x^{\mathbf{v}} \mid \mu(\boldsymbol{x} \textcircled{v} \boldsymbol{\alpha}).c \mid \mu(\boldsymbol{a} \textcircled{v} \boldsymbol{\alpha}).c & E_{\mathbf{v}} &::= e_{\mathbf{v}} \\
V_{\mathbf{n}} &::= v_{\mathbf{n}} & E_{\mathbf{n}} &::= \alpha^{\mathbf{n}} \mid V \textcircled{n} E \mid A \textcircled{n} E
\end{aligned}
$$

$$
\begin{aligned}
V_{\mathbf{lv}} &::= x^{\mathbf{lv}} \mid \mu(\boldsymbol{x} \textcircled{lv} \boldsymbol{\alpha}).c \mid \mu(\boldsymbol{a} \textcircled{lv} \boldsymbol{\alpha}).c \\
E_{\mathbf{lv}} &::= \alpha^{\mathbf{lv}} \mid \tilde{\mu} x^{\mathbf{lv}}.D[\langle x^{\mathbf{lv}} \| E_{\mathbf{lv}} \rangle] \mid V \textcircled{lv} E \mid A \textcircled{lv} E \\
V_{\mathbf{ln}} &::= x^{\mathbf{ln}} \mid \mu(\boldsymbol{x} \textcircled{ln} \boldsymbol{\alpha}).c \mid \mu(\boldsymbol{a} \textcircled{ln} \boldsymbol{\alpha}).c \mid \mu \alpha^{\mathbf{ln}}.D[\langle V_{\mathbf{ln}} \| \alpha^{\mathbf{ln}} \rangle] \\
E_{\mathbf{ln}} &::= \alpha^{\mathbf{ln}} \mid V \textcircled{ln} E \mid A \textcircled{ln} E \\
D &::= \square \mid \langle v_{\mathbf{lv}} \| \tilde{\mu} y^{\mathbf{lv}}.D \rangle \mid \langle \mu \alpha^{\mathbf{ln}}.D \| e_{\mathbf{ln}} \rangle
\end{aligned}
$$

Fig. 3. (Co-)values in by-name (**n**), -value (**v**), -need (**lv**), -co-need (**ln**) and **u**.

$$
\frac{}{\Gamma, \boldsymbol{x} : A \vdash_{\Theta} \boldsymbol{x} : A \mid \Delta}\ Var
\qquad
\frac{}{\Gamma \mid \boldsymbol{\alpha} : A \vdash_{\Theta} \boldsymbol{\alpha} : A, \Delta}\ Co\text{-}Var
$$

$$
\frac{c : (\Gamma \vdash_{\Theta} \boldsymbol{\alpha} : A, \Delta)}{\Gamma \vdash_{\Theta} \mu \boldsymbol{\alpha}.c : A \mid \Delta}\ Act
\qquad
\frac{c : (\Gamma, \boldsymbol{x} : A \vdash_{\Theta} \Delta)}{\Gamma \mid \tilde{\mu} \boldsymbol{x}.c : A \vdash_{\Theta} \Delta}\ Co\text{-}Act
$$

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

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

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

Fig. 4. Type system for the multi-discipline, polymorphic sequent calculus.

Wadler [28]) that lift unevaluated components out of call-stacks to be computed, so there is no "largest" reduction theory that subsumes all others.

Values and Co-Values We can now give interpretations of some specific discipline symbols: the call-by-value (**v**), -name (**n**), -need (**lv** for "lazy value" [1]), -co-need (**ln** for "lazy name") and non-deterministic (**u**) disciplines are defined by the values and co-values in Fig. 3.

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 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. 4. 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 criteria on the right $\forall$ rule that the abstracted type variable in the premise is not free in the conclusion. 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.

Admissible Disciplines Our proof of strong normalization is parameterized by a collection of discipline symbols and their interpretation. However, there are two important properties on disciplines needed for our proof.

Definition 1. *A discipline is* stable *exactly when (co-)values are closed under reduction and substitution,* focalizing *exactly when* at least *all (1) variables, $\mu(\boldsymbol{x} \circledS \boldsymbol{\alpha}).c$, and $\mu(\boldsymbol{a} \circledS \boldsymbol{\alpha}).c$ are values, and (2) co-variables, $V \circledS E$, and $A \circledS E$, are co-values, and* admissible *exactly when it is stable and focalizing.*

Property 1. The $\mathbf{n}$, $\mathbf{v}$, $\mathbf{lv}$, $\mathbf{ln}$, and $\mathbf{u}$ disciplines are collectively admissible.

Our proof of strong normalization works uniformly for any collection of admissible disciplines. As we present the proof in the next section 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 Strong Normalization

While some properties, like type safety, are straightforward enough to prove directly [29], 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 [25] and reducibility candidates [8]: set up an interpretation for types that serves as a waypoint between syntax and safety. The interpretation for a type should encompass all programs of that type (adequacy) and also fit inside the intended candidate property (safety). 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 [2], 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-types* which encompasses all possible candidates but does not impose the necessary safety conditions.

Pre-Types In the biorthogonality family of methods [7,21,13], a type has a two-sided interpretation described by both a set of terms and a set of co-terms.

Intuitively, a model of a type describes some desired behavior of programs (like an algorithm, or specification), where the term side can be seen as a collection of implementations and the co-term side can be seen as set of test operations. By analogy, *orthogonality* is an operation that evaluates implementations (terms) with operations (co-terms). On the one hand, orthogonality selects only those implementations that pass a comprehensive set of tests, and on the other hand, orthogonality also selects only those test that pass the reference implementation(s). The biorthogonal interpretation of types then is safe by construction, where the co-terms (tests) of a type are exactly everything orthogonal to (here, forming a strongly normalizing command with) any term (implementation) of the type, and vice versa. Since orthogonality can always complete one half from the other, only one side is necessary.

However, the method we use here cannot rely on such luxuries. While constructing the interpretation of types, we will have to consider incremental steps which may include extra (co-)terms that create unsafe interactions and exclude necessary (co-)terms that would be safe. Therefore, the new insight is to work in a domain where terms and co-terms are grouped *together* as a single unit, and which includes many *pre-types* that are not candidate interpretations for types.

Definition 2. *A* pre-type $\mathcal{A}$ *(of discipline* $\mathbf{r}$*) is a pair* (A_v, A_e) *where* A_v *is a set of strongly normalizing* $\mathbf{r}$*-terms and* A_e *is a set of strongly normalizing* $\mathbf{r}$*-co-terms.*

We use ordinary set membership to refer to the underlying sets: given $\mathcal{A} = (A_v, A_e)$, we write $v \in \mathcal{A}$ for $v \in A_v$ and $e \in \mathcal{A}$ for $e \in A_e$. We write $\mathcal{SN}_\mathbf{r}$ for the pre-type containing all strongly normalizing (co-)terms of discipline $\mathbf{r}$ and $\perp\!\!\!\perp$ for the set of all strongly normalizing commands.

We can compare pre-types like we do with sets. But because they are built with two opposite sets, there are two different methods of comparison. The first comparison is *containment* which just checks that the (co-)terms of one pre-type also belong to the other. The second comparison corresponds instead to behavioral *sub-typing* [16]: A is a sub-type of B if every program fragment of A can be used in every context of B. Intuitively, the subsumption of sub-typing sends every producer of As (*i.e.,* terms) to B and dually sends every consumer of Bs (*i.e.,* co-terms) to A. We can also combine pre-types with unions and intersections that go along with these two comparisons: for containment this just means the union and intersection, respectively, of the sets underlying pre-types, but for sub-typing this corresponds to the intuition behind union and intersection types in programming languages.

Definition 3. *Let* $\mathcal{A} = (A_v, A_e)$ *and* $\mathcal{B} = (B_v, B_e)$ *be pre-types.* $\mathcal{A}$ *is* contained *in* $\mathcal{B}$*, written* $\mathcal{A} \sqsubseteq \mathcal{B}$*, and* $\mathcal{A}$ *is a* sub-type *of* $\mathcal{B}$*, written* $\mathcal{A} \leq \mathcal{B}$*, as follows:*

$$\mathcal{A} \sqsubseteq \mathcal{B} \text{ iff } A_v \subseteq B_v \text{ and } A_e \subseteq B_e \qquad \mathcal{A} \leq \mathcal{B} \text{ iff } A_v \subseteq B_v \text{ and } A_e \supseteq B_e$$

The union and intersection for containment $(\sqcup, \sqcap)$ *and sub-typing* $(\vee, \wedge)$ *are:*

$$\begin{aligned} \mathcal{A} \sqcup \mathcal{B} &= (A_v \cup B_v, A_e \cup B_e) & \qquad \mathcal{A} \vee \mathcal{B} &= (A_v \cup B_v, A_e \cap B_e) \\ \mathcal{A} \sqcap \mathcal{B} &= (A_v \cap B_v, A_e \cap B_e) & \qquad \mathcal{A} \wedge \mathcal{B} &= (A_v \cap B_v, A_e \cup B_e) \end{aligned}$$

Orthogonality The orthogonality operation on pre-types, $\mathcal{A}^\perp$, uses one pre-type to generate another one containing everything it can safely interact with and nothing more. Together, orthogonality and containment capture the notion of safety in terms of pre-types: $\mathcal{A} \sqsubseteq \mathcal{A}^\perp$ means $\langle v \| e \rangle \in \perp\!\!\!\perp$ for all $v, e \in \mathcal{A}$.

Definition 4. *The orthogonal of any pre-type $\mathcal{A}$ of $\mathbf{r}$, written $\mathcal{A}^\perp$, is:*

$$v_{\mathbf{r}} \in \mathcal{A}^\perp \iff \forall e_{\mathbf{r}} \in \mathcal{A}. \langle v_{\mathbf{r}} \| e_{\mathbf{r}} \rangle \in \perp\!\!\!\perp \qquad e_{\mathbf{r}} \in \mathcal{A}^\perp \iff \forall v_{\mathbf{r}} \in \mathcal{A}. \langle v_{\mathbf{r}} \| e_{\mathbf{r}} \rangle \in \perp\!\!\!\perp$$

A pre-type $\mathcal{A}$ of $\mathbf{r}$ is safe *exactly when $\mathcal{A} \sqsubseteq \mathcal{A}^\perp$.*

Although we have generalized the notion of orthogonality to pre-types, it still exhibits the properties which mimic negation in intuitionistic logic.

Property 2. Contrapositive: If $\mathcal{A} \sqsubseteq \mathcal{B}$ then $\mathcal{B}^\perp \sqsubseteq \mathcal{A}^\perp$. *Double orthogonal introduction*: $\mathcal{A} \sqsubseteq \mathcal{A}^{\perp\perp}$. *Triple orthogonal elimination*: $\mathcal{A}^{\perp\perp\perp} = \mathcal{A}^\perp$.

However, because pre-types also come with another notion of comparison, we get an additional *new* property of orthogonality that follows from sub-typing.

Property 3. Monotonicity: If $\mathcal{A} \leq \mathcal{B}$ then $\mathcal{A}^\perp \leq \mathcal{B}^\perp$.

This fact is key to our entire endeavor: the monotonicity of orthogonality (and similar operations) with respect to sub-typing guarantees that there are always fixed points of orthogonality. This is the fact that powers the fixed-point construction of symmetric candidates [2] that we generalize by rephrasing the construction in terms of a two-sided model of sub-typing.

Top Reduction Another standard part of a strong normalization proof is to identify a subset of reductions that are important to check for the purpose of normalization. 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:

$$\frac{c \succ_{\beta^{\to},\beta^{\forall}} c'}{c \rightsquigarrow_0 c'} \qquad \frac{c \succ_{\mu} c'}{c \rightsquigarrow_+ c'} \qquad \frac{c \succ_{\tilde{\mu}} c'}{c \rightsquigarrow_- c'} \qquad \frac{e_{\mathbf{r}} \succ_{\varsigma^{\to},\varsigma^{\forall}} e'_{\mathbf{r}}}{\langle V_{\mathbf{r}} \| e_{\mathbf{r}} \rangle \rightsquigarrow_- \langle V_{\mathbf{r}} \| e'_{\mathbf{r}} \rangle} \qquad \frac{c \rightsquigarrow_{+,0,-} c'}{c \rightsquigarrow c'}$$

Note that top reductions are distinguished based on a "charge:" the positive $\rightsquigarrow_+$ let the term of a command take control of computation, the negative $\rightsquigarrow_-$ let the co-term take control, and the neutral $\rightsquigarrow_0$ require that both the term and co-term cooperate to proceed. The purpose of this distinction is to help tame the potential for non-determinism: notice that both $\rightsquigarrow_{+,0}$ and $\rightsquigarrow_{-,0}$ are deterministic for *all* disciplines, but $\rightsquigarrow_{+,-}$ *may not* be depending on the discipline. We need to pay attention to non-determinism because it breaks the expected expansion property used in strong normalization proofs [9]. Normally, top expansion says that if $\langle v \| e \rangle \rightsquigarrow c$ and v, e, and c are all strongly normalizing then so is $\langle v \| e \rangle$. However, this might not work if there is another top reduction $\langle v \| e \rangle \rightsquigarrow c'$ where

c' loops forever. So generalizing top expansion to accommodate non-determinism must quantify over *all* possible top reductions; even after some other internal reductions have happened. Within a pre-type $\mathcal{A}$, non-deterministic top expansion assumes $\mathcal{A}$ is closed under reduction—if $v, e \in \mathcal{A}$ and $v \twoheadrightarrow v'$ and $e \twoheadrightarrow e'$ then $v', e' \in \mathcal{A}$—and that every possible top reduction of $\mathcal{A}$ commands leads to a strongly normalizing command. If so, every $\mathcal{A}$ command *is* strongly normalizing.

Lemma 1 (Nondeterministic Top Expansion). *If $\mathcal{A}$ is closed under reduction and for all $v, e \in \mathcal{A}$ and c, $\langle v \| e \rangle \rightsquigarrow c$ implies $c \in \perp\!\!\!\perp$, then $\mathcal{A}$ is safe.*

So we have a top expansion property for the general non-deterministic case, but what about when we are dealing with (co-)terms from a deterministic discipline like $\mathbf{v}$ or $\mathbf{n}$? We can identify a pre-type of deterministically-normalizing (co-)terms of $\mathbf{r}$ ($\mathcal{DN}_\mathbf{r}$) where *all* their possible top reductions either land in $\perp\!\!\!\perp$ or not after any number of other reductions have occurred, which is defined as:

$$v_\mathbf{r} \in \mathcal{DN}_\mathbf{r} \iff \forall e_\mathbf{r} \in \mathcal{SN}_\mathbf{r}.(v_\mathbf{r} \twoheadrightarrow v'_\mathbf{r} \wedge \langle v'_\mathbf{r} \| e_\mathbf{r} \rangle \rightsquigarrow c \wedge \langle v'_\mathbf{r} \| e_\mathbf{r} \rangle \rightsquigarrow c') \Rightarrow (c \in \perp\!\!\!\perp \Leftrightarrow c' \in \perp\!\!\!\perp)$$
$$e_\mathbf{r} \in \mathcal{DN}_\mathbf{r} \iff \forall v_\mathbf{r} \in \mathcal{SN}_\mathbf{r}.(e_\mathbf{r} \twoheadrightarrow e'_\mathbf{r} \wedge \langle v_\mathbf{r} \| e'_\mathbf{r} \rangle \rightsquigarrow c \wedge \langle v_\mathbf{r} \| e'_\mathbf{r} \rangle \rightsquigarrow c') \Rightarrow (c \in \perp\!\!\!\perp \Leftrightarrow c' \in \perp\!\!\!\perp)$$

As shorthand, we write $\mathcal{A}^d$ to mean $\mathcal{A} \sqcap \mathcal{DN}_\mathbf{r}$ for a pre-type $\mathcal{A}$ of $\mathbf{r}$. Now, we get an improved top expansion property for deterministically-normalizing (co-)terms.

Lemma 2 (Deterministic top expansion). *If $\mathbf{r}$ is stable, $v, e \in \mathcal{SN}_\mathbf{r}$, either $v \in \mathcal{DN}_\mathbf{r}$ or $e \in \mathcal{DN}_\mathbf{r}$, and $\langle v \| e \rangle \rightsquigarrow c \in \perp\!\!\!\perp$ then $\langle v \| e \rangle \in \perp\!\!\!\perp$.*

Deterministic top expansion relies on commutation between top and non-top reductions based on the stability of $\mathbf{r}$. Note that for any discipline $\mathbf{r}$ where top reduction is deterministic, it follows that $\mathcal{SN}_\mathbf{r} = \mathcal{DN}_\mathbf{r}$, and so the above deterministic top expansion property holds for *any* term and co-term of $\mathbf{r}$. Since the $\mathbf{n}$, $\mathbf{v}$, $\mathbf{lv}$, and $\mathbf{ln}$ disciplines all meet this criteria, they all enjoy the usual expansion property unlike $\mathbf{u}$.

Reducibility Candidates The interpretation of a type should be both adequate and safe. Safety, which tells us a type's interpretation contains only good interactions, was already captured by orthogonality ($\mathcal{A}$ is safe when $\mathcal{A} \sqsubseteq \mathcal{A}^\perp$). Adequacy, which tells us a type's interpretation contains all programs dictated by the typing rules, is a little more involved, however. Certainly, interpretations should include everything that interacts well with the type ($\mathcal{A}^\perp \sqsubseteq \mathcal{A}$), but this is not enough. We need to be able to show type membership looking at a single top reduction, but reduction isn't in general deterministic, so we must explicitly require that a (co-)term that interacts well with $\mathcal{A}$ after it causes one top reduction is also in $\mathcal{A}$. This extra condition only tests the (co-)values of $\mathcal{A}$: $\langle v_\mathbf{r} \| e_\mathbf{r} \rangle \rightsquigarrow_+ c$ only when $e_\mathbf{r}$ is a co-value of $\mathbf{r}$ and $\langle v_\mathbf{r} \| e_\mathbf{r} \rangle \rightsquigarrow_- c$ only when $v_\mathbf{r}$ is a value of $\mathbf{r}$.

Definition 5. *The* saturation *of a pre-type $\mathcal{A}$ of $\mathbf{r}$ is defined as:*

$$v_\mathbf{r} \in \mathcal{A}^s \iff \forall E_\mathbf{r} \in \mathcal{A}.\langle v_\mathbf{r} \| E_\mathbf{r} \rangle \rightsquigarrow^=_{+,0} c \in \perp\!\!\!\perp \qquad e_\mathbf{r} \in \mathcal{A}^s \iff \forall V_\mathbf{r} \in \mathcal{A}.\langle V_\mathbf{r} \| e_\mathbf{r} \rangle \rightsquigarrow^=_{-,0} c \in \perp\!\!\!\perp$$

where $\rightsquigarrow^=_{+,0}$ and $\rightsquigarrow^=_{-,0}$ are the reflexive closures of $\rightsquigarrow_{+,0}$ and $\rightsquigarrow_{-,0}$, respectively. A pre-type $\mathcal{A}$ of $\mathbf{r}$ is adequate *exactly when $\mathcal{A}^s \sqsubseteq \mathcal{A}$.*

Now that we know how to phrase safety in terms of orthogonality and adequacy in terms of saturation, we can say that *reducibility candidates*, which are the potential interpretations of types, are pre-types that lie between their own saturation and orthogonal.

Definition 6. *A* reducibility candidate $\mathcal{A}$ (of $\mathbf{r}$) *is a safe and adequate pre-type* $\mathcal{A}$ *of* $\mathbf{r}$ *(*i.e., $\mathcal{A}^s \sqsubseteq \mathcal{A} \sqsubseteq \mathcal{A}^{\perp}$*). $CR_{\mathbf{r}}$ is the set of all reducibility candidates of* $\mathbf{r}$.

In practice, the $\mathcal{A}^{\perp}$ upper-bound is used to justify the cut rule for forming commands, and the $\mathcal{A}^s$ lower-bound is used to justify the left and right rules for activation, implication, and universal quantification. Also, the $\mathcal{A}^s$ lower-bound serves a second purpose by ensuring that reducibility candidates are all inhabited by (co-)variables, which will be needed to show that typing implies strong normalization even for open commands and (co-)terms.

As it turns out, there is an equivalent way of identifying reducibility candidates of admissible disciplines: they are all fixed points of saturation.

Lemma 3 (Reducibility fixed-point). *For any pre-type $\mathcal{A}$ of an admissible discipline* $\mathbf{r}$*, $\mathcal{A}$ is a reducibility candidate of* $\mathbf{r}$ *if and only if $\mathcal{A} = \mathcal{A}^s$.*

Reducibility candidates *are* fixed points of saturation because $\mathcal{A}^{\perp} \sqsubseteq \mathcal{A}^s$ for any $\mathcal{A}$, and the reverse follows from the focalization of $\mathbf{r}$ because the participants in neutral β-reductions—abstractions and call stacks—are (co-)values that can be tested by saturation. The equivalence between candidates and fixed points gives us a general-purpose construction method for candidates of *any* admissible discipline by solving recursive pre-type equations.

Fixed-Point Solutions The fixed-point construction of types is powered by the pervasive monotonicity properties of sub-typing between pre-types. Monotonicity isn't limited to just orthogonality; other operations, like saturation and containment-union with a constant pre-type, are also monotonic: for any $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ of $\mathbf{r}$, if $\mathcal{A} \leq \mathcal{B}$ then $\mathcal{A}^s \leq \mathcal{B}^s$ and $\mathcal{A} \sqcup \mathcal{C} \leq \mathcal{A} \sqcup \mathcal{C}$. Therefore, if we describe the essence of a type with some pre-type $\mathcal{C}$, we can build a fully-saturated pre-type around it by finding a solution to the equation $\mathcal{A} = \mathcal{C} \sqcup \mathcal{A}^s$. Combined with the fact that sub-typing (and containment) forms a lattice on pre-types, the Knaster-Tarksi fixed point theorem ensures that this equation has a fixed point, giving us the basis of a function for generating saturated pre-types.

Lemma 4 (Fixed-point construction). *For every discipline* $\mathbf{r}$*, there is a function $\mathcal{F}_{\mathbf{r}}(-)$ such that for any pre-type $\mathcal{C}$ of* $\mathbf{r}$*, $\mathcal{F}_{\mathbf{r}}(\mathcal{C}) = \mathcal{C} \sqcup \mathcal{F}_{\mathbf{r}}(\mathcal{C})^s$.*

The Knaster-Tarski fixed point theorem, however, does not ensure that there is a *unique* fixed point satisfying the equation. Therefore, the $\mathcal{F}_{\mathbf{r}}(-)$ operations must somehow pick which among the possible solutions is *the* result. Two readily available options are the largest or smallest such fixed points with respect to sub-typing, but note that neither one is "more principled" than the other: the largest one has the most terms but fewest co-terms, and the smallest one has the fewest terms but most co-terms. Either one will work for demonstrating strong normalization, however, as long as we are consistent. Moreover, we will prove in the next section (Lemma 7) that for deterministic $\mathbf{r}$ the solutions will be unique.

So now we know how to build a saturated extension of any pre-type $\mathcal{C}$ of $\mathbf{r}$ that satisfies one of the conditions for being a reducibility candidate by definition: $\mathcal{F}_{\mathbf{r}}(\mathcal{C})^s \sqsubseteq \mathcal{F}_{\mathbf{r}}(\mathcal{C})$. But we still need to make sure that this extension is safe: we must show that $\mathcal{F}_{\mathbf{r}}(\mathcal{C}) \sqsubseteq \mathcal{F}_{\mathbf{r}}(\mathcal{C})^{\perp}$. It turns out that the safety condition of reducibility candidates follows when $\mathcal{C}$ is a pre-type consisting of only deterministically-normalizing (co-)values that only form strongly-normalizing commands, because then the result of $\mathcal{F}_{\mathbf{r}}(\mathcal{C})$ is itself a fixed point of saturation.

Lemma 5 (Fixed-point validity). *If $\mathcal{C} \sqsubseteq \mathcal{C}^{\perp dv}$ then $\mathcal{F}_{\mathbf{r}}(\mathcal{C}) = \mathcal{F}_{\mathbf{r}}(\mathcal{C})^s$.*

Where we write $\mathcal{V}_{\mathbf{r}}$ for the pre-type of strongly normalizing (co-)values of discipline $\mathbf{r}$, and use the shorthand $\mathcal{A}^v = \mathcal{A} \sqcap \mathcal{V}_{\mathbf{r}}$ for pre-types $\mathcal{A}$ in $\mathbf{r}$.

Interpretations of Types With a uniform method for generating reducibility candidates in hand, we can now construct the candidates for particular types. Both implication and universal quantification are *negative* types defined by their observations—call stacks—so their interpretation starts with the negative construction of a pre-type that selects terms compatible with some co-terms: for a set of strongly-normalizing $\mathbf{r}$-co-terms O, $Neg(O)$ is the following pre-type of $\mathbf{r}$:

$$v_{\mathbf{r}} \in Neg(O) \iff \forall E_{\mathbf{r}} \in O. \langle v_{\mathbf{r}} \| E_{\mathbf{r}} \rangle \in \perp\!\!\!\perp \qquad e_{\mathbf{r}} \in Neg(O) \iff e_{\mathbf{r}} \in O$$

The above negative construction satisfies the validity criteria for the fixed-point reducibility candidates from Lemma 5 ($\mathcal{C} \sqsubseteq \mathcal{C}^{\perp dv}$) by keeping only its deterministically-normalizing (co-)values and closing it under orthogonality.

Lemma 6. *For any set O of deterministically-normalizing $\mathbf{r}$-co-values,*
$Neg(O)^{dv} \sqsubseteq Neg(O)^{dv\perp dv} = Neg(O)^{dv\perp dv\perp dv}$.

We now have a negative interpretation for the specific type constructors:

- For all $\mathcal{A}$ and $\mathcal{B}$, $\mathcal{A} \xrightarrow{\mathbf{r}} \mathcal{B} \triangleq \mathcal{F}_{\mathbf{r}}(Neg(\{V \,❶\, E \mid V \in \mathcal{A}, E \in \mathcal{B}\})^{dv\perp dv}) \in CR_{\mathbf{r}}$.
- For all $K \subseteq CR_{\mathbf{t}}$, $\forall^{\mathbf{r}}\mathbf{t}.K \triangleq \mathcal{F}_{\mathbf{r}}(Neg(\{A_{\mathbf{t}} \,❶\, E \mid \mathcal{B} \in K, E \in \mathcal{B}\})^{dv\perp dv}) \in CR_{\mathbf{r}}$.

Adequacy The final step is to give an interpretation for syntactic types, environments, and sequents as reducibility candidates, substitutions, and propositions, respectively, where we write CR for $\bigcup_{\mathbf{r}} CR_{\mathbf{r}}$:

$$[\![a^{\mathbf{r}}]\!]\phi \triangleq \phi(a) \quad [\![A \xrightarrow{\mathbf{r}} B]\!]\phi \triangleq [\![A]\!]\phi \xrightarrow{\mathbf{r}} [\![B]\!]\phi \quad [\![\forall^{\mathbf{r}} a^{\mathbf{t}}.B]\!]\phi \triangleq \forall^{\mathbf{r}}\mathbf{t}.\{[\![B]\!](\phi, \mathcal{A}/a^{\mathbf{t}}) \mid \mathcal{A} \in CR_{\mathbf{t}}\}$$

$$[\![\Theta]\!] \triangleq \{\phi \mid \forall a^{\mathbf{r}} \in \Theta. \phi(a) \in CR_{\mathbf{r}}\}$$

$$[\![\Gamma \vdash_{\Theta} \Delta]\!]\phi \triangleq \{\rho \mid \forall a^{\mathbf{r}} \in \Theta. a\{\rho\} \in Type_{\mathbf{r}} \land \forall \boldsymbol{x}{:}A \in \Gamma. \boldsymbol{x}\{\rho\} \in [\![A]\!]\phi \land \forall \boldsymbol{\alpha}{:}A \in \Delta. \boldsymbol{\alpha}\{\rho\} \in [\![A]\!]\phi\}$$

$$c : (\Gamma \vDash_{\Theta} \Delta) \triangleq \forall \phi \in [\![\Theta]\!], \rho \in [\![\Gamma \vdash_{\Theta} \Delta]\!]\phi. c\{\rho\} \in \perp\!\!\!\perp$$

$$\Gamma \vDash_{\Theta} v : A \mid \Delta \triangleq \forall \phi \in [\![\Theta]\!], \rho \in [\![\Gamma \vdash_{\Theta} \Delta]\!]\phi. [\![A]\!]\phi \in CR \land v\{\rho\} \in [\![A]\!]\phi$$

$$\Gamma \mid e : A \vDash_{\Theta} \Delta \triangleq \forall \phi \in [\![\Theta]\!], \rho \in [\![\Gamma \vdash_{\Theta} \Delta]\!]\phi. [\![A]\!]\phi \in CR \land e\{\rho\} \in [\![A]\!]\phi$$

Typing derivations are adequate with respect to the interpretation of their conclusion for any admissible disciplines, which in turn gives us strong normalization.

Theorem 1 (Adequacy). *(1) $c : (\Gamma \vdash_\Theta \Delta)$ implies $c : (\Gamma \vDash_\Theta \Delta)$, (2) $\Gamma \vdash_\Theta v : A \mid \Delta$ implies $\Gamma \vDash_\Theta v : A \mid \Delta$, and (3) $\Gamma \mid e : A \vdash_\Theta \Delta$ implies $\Gamma \mid e : A \vDash_\Theta \Delta$.*

Adequacy follows by induction on the typing derivation. Note that the requirement that disciplines are focalizing is used to justify the left and right rules of functions and polymorphism so that abstractions and call stacks end up in the meaning of those types. This also ensures that (co-)variables are (co-)values (*resp.*) that inhabit every reducibility candidate, so that every environment has a suitable substitution used to extract strong normalization for reduction of open commands, terms, and co-terms.

Corollary 1 (Strong normalization). *Typed commands, terms, and co-terms are strongly normalizing.*

6 Biorthogonals are Fixed Points

The candidate-based approach to strong normalization—tracing back to Tait [25] and Girard [6] and fitting in the general area of logical relations [27] and realizability [11]—easily accommodates impredicative polymorphism by outlining the candidate meanings of types before defining any particular type. Tait's original method doesn't work for us because we need types to classify co-terms in addition to terms. The use of orthogonality for modeling types appears in multiple places, including Girard's [7] linear logic, Krivine's [13] classical realizability, and Pitts' [21] $\top\top$-closed relations, and can prove strong normalization for certain disciplines. For call-by-name we could start by defining types via their observations (so for functions, valid call stacks), the set of terms of that type as anything orthogonal to these observations, and, finally, the set of co-terms of that type as the double orthogonal of the defining observations. The dual approach, starting with the constructions of values, works for call-by-value.

Munch-Maccagnoni [17] identified a key feature of the orthogonal construction of types: all call-by-value and -name types are generated by their values and co-values, respectively. That is, the meaning of a type *is* the orthogonal of its (co-)values; in our notation, $\mathcal{A} = \mathcal{A}^{v\perp}$. As it turns out, these are exactly the reducibility candidates produced by our fixed-point framework for well-behaved disciplines that induce enough determinism. In the general case, the inherent non-determinism of disciplines like $\mathbf{u}$ allows for many different and incompatible candidate meanings for a particular type [14], but for disciplines like $\mathbf{v}$, $\mathbf{n}$, $\mathbf{lv}$, and $\mathbf{ln}$ that eliminate the fundamental non-deterministic choice, there can only be one meaning for each type and it must be the fixed point of $-^{v\perp}$.

Lemma 7. *For any admissible discipline $\mathbf{r}$ where $\mathcal{SN}_\mathbf{r} = \mathcal{DN}_\mathbf{r}$ and pre-type $\mathcal{A}$ of $\mathbf{r}$, $\mathcal{A}$ is the* unique *reducibility candidate containing $\mathcal{A}^v$ if and only if $\mathcal{A} = \mathcal{A}^{v\perp}$.*

This extra uniqueness property of candidates provided by determinism gives us a more direct method of building them in a finite number of steps, as opposed to using the existence of solutions to recursive equations. In particular, note that there is a *positive* construction of pre-types, dual to $Neg(-)$ from Section 5, which uses some set of terms C to generate all compatible co-terms:

$$v \in Pos(C) \iff v \in C \qquad e \in Pos(C) \iff \forall v \in C. \langle v \| e \rangle \in \perp\!\!\!\perp$$

Both the positive and negative construction of pre-types can be used to directly construct reducibility candidates of any deterministic discipline (as in Lemma 7). In the special cases of call-by-value and call-by-name there is an even simpler construction because they trivialize the co-value- and value-restriction, respectively.

Theorem 2. *Let* $\mathbf{r}$ *be any admissible discipline with deterministic top reduction (including* $\mathbf{v}$*,* $\mathbf{n}$*,* $\mathbf{lv}$*, and* $\mathbf{ln}$*, among others),* C *be a set of* $\mathbf{r}$*-values, and* O *be a set of* $\mathbf{r}$*-co-values. Both* $Pos(C)^{v\perp v\perp}$ *and* $Neg(O)^{v\perp v\perp}$ *are reducibility candidates of* $\mathbf{r}$*. Furthermore,* $Pos(C)^{\perp} \in CR_{\mathbf{v}}$ *if* $\mathbf{r} = \mathbf{v}$ *and* $Neg(O)^{\perp} \in CR_{\mathbf{n}}$ *if* $\mathbf{r} = \mathbf{n}$*.*

The finitely-constructed candidates $Neg(O)^{\perp} \in CR_{\mathbf{n}}$ and $Pos(C)^{\perp} \in CR_{\mathbf{v}}$ are exactly the usual biorthogonal meanings of types in call-by-name and call-by-value languages: both $Neg(O)$ and $Pos(C)$ include a built-in orthogonal on one side of the pre-type to get started, and the second orthogonal is a closure operation since any more are redundant ($Neg(O)^{\perp\perp} = Neg(O)^{\perp}$ and $Pos(C)^{\perp\perp} = Pos(C)^{\perp}$). For example, let the set of call-stacks for two pre-types $\mathcal{A}$ and $\mathcal{B}$ be $\mathcal{A} \textcircled{r} \mathcal{B} = \{V \textcircled{r} E \mid V \in \mathcal{A}, E \in \mathcal{B}\}$, so that the interpretation of call-by-name and -value function types *must* be

$$\mathcal{A} \xrightarrow{\mathbf{n}} \mathcal{B} = ((\mathcal{A} \textcircled{n} \mathcal{B})^{\perp}, (\mathcal{A} \textcircled{n} \mathcal{B})^{\perp\perp}) \quad \mathcal{A} \xrightarrow{\mathbf{v}} \mathcal{B} = ((\mathcal{A} \textcircled{v} \mathcal{B})^{\perp v \perp\perp}, (\mathcal{A} \textcircled{v} \mathcal{B})^{\perp v \perp})$$

Theorem 2 also gives the first finite construction of reducibility candidates for call-by-need and its dual, which only differs from the simple biorthogonal meanings by being careful about (co-)values and using one more level of orthogonality to reach a fixed point. For example, lazy function types, where $\mathbf{l}$ is $\mathbf{lv}$ or $\mathbf{ln}$, *must* be

$$\mathcal{A} \xrightarrow{\mathbf{l}} \mathcal{B} = ((\mathcal{A} \textcircled{l} \mathcal{B})^{\perp v \perp v \perp}, (\mathcal{A} \textcircled{l} \mathcal{B})^{\perp v \perp})$$

The uniqueness condition of Lemma 7 removes any other possibilities for call-by-name and -value specifically—$Neg(O)^{\perp} \in CR_{\mathbf{n}}$ and $Pos(C)^{\perp} \in CR_{\mathbf{v}}$ are the *only* candidates containing $Neg(O)^{\perp v}$ and $Pos(C)^{\perp v}$—and similarly for the general-purpose positive and negative candidates. That means the candidates of $\mathbf{n}$, $\mathbf{v}$, $\mathbf{lv}$, and $\mathbf{ln}$, and any other deterministic, admissible discipline produced by our general-purpose fixed-point construction must be exactly these, so our framework subsumes the existing discipline-specific biorthogonal methods for (any combinations of) call-by-name and -value.

In comparison with Barbanera and Berardi's symmetric candidates method [2] for the symmetric λ-calculus—a calculus corresponding to $\mathbf{u}$ since all (co-)terms are substitutable and there are no ς-reductions—there are more differences. The main underlying idea to generate candidates as the fixed point of some saturation operation is the same, as is the definition of candidates as something in between saturation and orthogonality, but the meaning of "saturation" used here is more general. In particular, symmetric candidates defines saturation in terms of the syntax of programs, requiring that (co-)variables and certain μ- and $\tilde{\mu}$-abstractions satisfying some conditions are present. We instead define saturation in terms of the behavior of programs, requiring that they work—either now or in one step—with all relevant (co-)values. When considering only the **u** discipline, the approaches produce identical candidates. However, basing saturation on dynamic structure instead of syntactic structure has two benefits. First, it is straightforward to extend the basic method to accommodate additional language features, like multiple disciplines and focusing via ς-reductions as we have done here, since the meaning of saturation does not have to change: run-time behavior is enough to uniformly describe new features. Second, our definition of saturation is strictly more inclusive than the one of symmetric candidates: *everything* that works *must* be included. The larger saturation is key for Lemma 7 and Theorem 2 and for subsuming the biorthogonal methods in the more general multi-discipline setting: since we know that candidates include all the sensible (co-)terms, there is less room for spurious variations making the final result more precise.

7 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-types already comes with a built-in notion of sub-typing 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 intersection 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.

References

1. Ariola, Z.M., Herbelin, H., Saurin, A.: Classical call-by-need and duality. In: TLCA'11 (2011)
2. Barbanera, F., Berardi, S.: A symmetric lambda calculus for "classical" program extraction. In: TACS'94 (1994)
3. Curien, P.L., Herbelin, H.: The duality of computation. In: ICFP'00 (2000)
4. Downen, P., Ariola, Z.M.: The duality of construction. In: ESOP'14 (2014)

5. Downen, P., Johnson-Freyd, P., Ariola, Z.M.: Structures for structural recursion. In: ICFP'15 (2015)
6. Girard, J.Y.: Interprétation fonctionnelle et elimination des coupures de l'arithmétique d'ordre supérieur. These d'état, Université de Paris 7 (1972)
7. Girard, J.Y.: Linear logic. Theoretical Computer Science 50, 1–102 (1987)
8. Girard, J.Y., Taylor, P., Lafont, Y.: Proofs and Types. Cambridge University Press, New York, NY, USA (1989)
9. Graham-Lengrand, S.: Polarities & Focussing: a journey from Realisability to Automated Reasoning. Habilitation thesis, Université Paris-Sud (2014)
10. Herbelin, H., Zimmermann, S.: An operational account of call-by-value minimal and classical λ-calculus in "natural deduction" form. In: TLCA'09 (2009)
11. Kleene, S.C.: On the interpretation of intuitionistic number theory. Journal of Symbolic Logic 10(4), 109–124 (12 1945)
12. Krivine, J.L.: A call-by-name lambda-calculus machine. Higher-Order and Symbolic Computation 20(3), 199–207 (2007)
13. Krivine, J.L.: Realizability in classical logic. In: Interactive models of computation and program behaviour, vol. 27. Société Mathématique de France (2009)
14. Lengrand, S., Miquel, A.: Classical Fω, orthogonality and symmetric candidates. Annals of Pure and Applied Logic 153(1), 3–20 (2008)
15. Levy, P.B.: Call-By-Push-Value. Ph.D. thesis, University of London (Aug 2001)
16. Liskov, B.: Keynote address-data abstraction and hierarchy. In: OOPSLA '87 (1987)
17. Munch-Maccagnoni, G.: Focalisation and classical realisability. In: CSL'09 (2009)
18. Munch-Maccagnoni, G.: Syntax and Models of a non-Associative Composition of Programs and Proofs. Ph.D. thesis, Université Paris Diderot (2013)
19. Peyton Jones, S.: https://www.red-gate.com/simple-talk/opinion/geek-of-the-week/simon-peyton-jones-geek-of-the-week/
20. Peyton Jones, S.L., Launchbury, J.: Unboxed values as first class citizens in a non-strict functional language. In: FPCA. pp. 636–666 (1991)
21. Pitts, A.M.: Parametric polymorphism and operational equivalence. Mathematical Structures in Computer Science 10(3), 321–359 (2000)
22. Plotkin, G.D.: Call-by-name, call-by-value and the lambda-calculus. Theoretical Computer Science 1, 125–159 (1975)
23. Ronchi Della Rocca, S., Paolini, L.: The Parametric λ-Calculus: a Metamodel for Computation. Springer-Verlag (2004)
24. Sabry, A., Felleisen, M.: Reasoning about programs in continuation-passing style. In: LFP'92. pp. 288–298 (1992)
25. Tait, W.W.: Intensional interpretations of functionals of finite type I. Journal of Symbolic Logic 32, 198–212 (1967)
26. Turbak, F., Gifford, D., Sheldon, M.A.: Design Concepts in Programming Languages. The MIT Press (2008)
27. Wadler, P.: Theorems for free! In: FPCA'89 (1989)
28. Wadler, P.: Call-by-value is dual to call-by-name. In: ICFP'03 (2003)
29. Wright, A., Felleisen, M.: A syntactic approach to type soundness. Information and Computation 115(1) (1994)
30. Zeilberger, N.: The Logical Basis of Evaluation Order and Pattern-Matching. Ph.D. thesis, Carnegie Mellon University (2009)

A Appendix

This appendix contains additional details, applications, and proofs supporting the paper *Uniform Strong Normalization for Multi-Discipline Calculi*. In order, the topics to follow are:

B A multi-discipline $\lambda\mu$-calculus; a language which corresponds to natural deduction instead of the sequent calculus. This calculus serves as a practical distillation of the ideas covered in the above paper into a more conventional form, and illustrates how the results and techniques can be applied to functional programming languages (based on the λ-calculus) that make use of combinations of call-by-value, call-by-name, and call-by-need evaluation orders.

C Operational semantics for both the (**v**, **n**, **lv**, **ln**, **u** instance of the) multi-discipline sequent calculus and the above multi-discipline $\lambda\mu$-calculus, which are restrictions on the respective reduction theories. This section discusses how to run programs by eliminating spurious non-determinism, how to read results, and illustrates the standard type safety property for both calculi.

D A proof that the five specific disciplines mentioned in the paper (**v**, **n**, **lv**, **ln**, **u**) are indeed admissible; that is to say, they satisfy the *stable* and *focalizing* properties from Definition 1.

E The full details of the proof of strong normalization for the parameterized, multi-discipline sequent calculus, to accompany the proof sketch described in Section 5.

F The proof that biorthogonal interpretations of types classifying deterministic (co-)terms are the unique solutions to the fixed point model. Specifically, the extended biorthogonal definition from Theorem 2 is a valid reducibility candidate for a type of any deterministic discipline, and the standard positive and negative biorthogonal definitions are valid for the special cases of call-by-value and call-by-name types, respectively.

G A proof that typed reduction of the above multi-discipline $\lambda\mu$-calculus is strongly normalizing, which is carried out by translating reductions in the $\lambda\mu$-calculus to reductions in the sequent calculus.

B A Multi-Discipline $\boldsymbol{\lambda\mu}$-Calculus

The results thus far has been for a language of the classical sequent calculus, but they can be but they can be put to use in functional programming, too. We do this with a multi-discipline $\lambda\mu$-calculus that mixes the three disciplines used in practice: **n**, **v**, and **lv** disciplines. The syntax, given in Fig. 5, imports the same notions of annotated (co-)variables, types, and kinds (except for our restriction down to just three concrete kinds). However, there is a problem with the terse λ-calculus notation for application: in an application $v_{\mathbf{s}}\ v_{\mathbf{t}}$ we know the discipline of the function $v_{\mathbf{s}}$ and argument $v_{\mathbf{t}}$ parts, but we have no idea what the discipline of the returned result might be. And unfortunately there is no room in the empty space between the two to put this extra information. Therefore, we write function application with a triangle operator (used in F#)

$$v_{\mathbf{s}} \in \mathit{Term}_{\mathbf{s}} ::= \mu\alpha^{\mathbf{s}}.c \mid x^{\mathbf{s}} \mid \lambda^{\mathbf{s}}\boldsymbol{x}.v \mid \Lambda^{\mathbf{s}}\boldsymbol{a}.v \mid v \lhd_{\mathbf{s}} v' \mid v \lhd_{\mathbf{s}} A \mid \mathbf{let}\, x^{\mathbf{t}} = v_{\mathbf{t}}\,\mathbf{in}\, v_{\mathbf{s}}$$

$$c \in \mathit{Command} ::= [\alpha^{\mathbf{s}}]v_{\mathbf{s}} \quad v \in \mathit{Term} ::= v_{\mathbf{s}} \quad \boldsymbol{x} ::= x^{\mathbf{s}} \quad \boldsymbol{a} ::= a^{\mathbf{s}} \quad \mathbf{r}, \mathbf{s}, \mathbf{t} ::= \mathbf{v} \mid \mathbf{n} \mid \mathbf{lv}$$

Fig. 5. Syntax of a multi-discipline, polymorphic $\lambda\mu$-calculus.

$$\frac{}{\Gamma, \boldsymbol{x} : A \vdash_{\Theta} \boldsymbol{x} : A \mid \Delta}\ \mathit{Var} \qquad \frac{\Gamma \vdash_{\Theta} v' : B \mid \Delta \quad \Gamma, \boldsymbol{x} : B \vdash_{\Theta} v : A \mid \Delta}{\Gamma \vdash_{\Theta} \mathbf{let}\, \boldsymbol{x} = v'\,\mathbf{in}\, v : A \mid \Delta}\ \mathit{Let}$$

$$\frac{c : (\Gamma \vdash_{\Theta} \boldsymbol{\alpha} : A, \Delta)}{\Gamma \vdash_{\Theta} \mu\boldsymbol{\alpha}.c : A \mid \Delta}\ \mathit{Act} \qquad \frac{\Gamma \vdash_{\Theta} v : A \mid \boldsymbol{\alpha} : A, \Delta}{[\boldsymbol{\alpha}]v : (\Gamma \vdash_{\Theta} \boldsymbol{\alpha} : A, \Delta)}\ \mathit{Pass}$$

$$\frac{\Gamma, \boldsymbol{x} : A \vdash_{\Theta} v : B \mid \Delta}{\Gamma \vdash_{\Theta} \lambda^{\mathbf{s}}\boldsymbol{x}.v : A \xrightarrow{\mathbf{s}} B \mid \Delta}\ {\to}I \qquad \frac{\Gamma \vdash_{\Theta} v : A \xrightarrow{\mathbf{s}} B_{\mathbf{r}} \mid \Delta \quad \Gamma \vdash_{\Theta} v' : A \mid \Delta}{\Gamma \vdash_{\Theta} v \lhd_{\mathbf{r}} v' : B_{\mathbf{r}} \mid \Delta}\ {\to}E$$

$$\frac{\Gamma, \vdash_{\Theta,\boldsymbol{a}} v : B \mid \Delta}{\Gamma \vdash_{\Theta} \Lambda^{\mathbf{s}}\boldsymbol{a}.v : \forall^{\mathbf{s}}\boldsymbol{a}.B \mid \Delta}\ \forall I \qquad \frac{\Gamma \vdash_{\Theta} v : \forall^{\mathbf{s}}a^{\mathbf{t}}.B_{\mathbf{r}} \mid \Delta}{\Gamma \vdash_{\Theta} v \lhd_{\mathbf{r}} A_{\mathbf{t}} : B_{\mathbf{r}}\{A_{\mathbf{t}}/a^{\mathbf{t}}\} \mid \Delta}\ \forall E$$

Fig. 6. Type system for multi-discipline, polymorphic $\lambda\mu$-calculus.

which gives us a place to mark the disciplines of application results: $v \lhd_{\mathbf{s}} v'$ is an **s** term for any v and v'. The type system in Fig. 6 is essentially standard, combining polymorphism and classical natural deduction.

The reduction theory given in Fig. 7 includes (1) the usual β rules for λ and **let**, along with a rule corresponding to the sequent's ς, which are enough to compute answers for call-by-name and -value, (2) a rule for commuting contexts with **let**s (generalizing call-by-need's reassociation axioms) which are needed to push computation through delayed **lv** bindings, and (3) extra rules for μ-abstractions. Values (V) and frame contexts (F) given in Fig. 8; evaluation contexts (E) are compositions of several frames. We know this multi-discipline $\lambda\mu$-calculus is strongly normalizing by translating it to the sequent calculus.

Theorem 3. *Typed multi-discipline $\lambda\mu$ is strongly normalizing.*

C Operational Semantics

To get a better understanding of the computational behavior of the multi-discipline calculi, let's briefly consider their operational semantics.

Multi-Discipline Sequent Calculus

Due to the machine-like nature of the sequent calculus, the operational semantics is straightforward to define; at least for a particular choice of disciplines. The operational semantics for the combined **v**, **n**, **lv**, **ln**, and **u** instance of the parametric, multi-discipline sequent calculus is the closure of top reduction ($c \rightsquigarrow c'$) under D contexts from Fig. 3, plus one less restrictive use of the ς rules

$$v_{\mathbf{s}} \lhd_{\mathbf{r}} v_{\mathbf{t}} \to \mathbf{let}\, x^{\mathbf{s}} = v_{\mathbf{s}}\, \mathbf{in}\, \mathbf{let}\, y^{\mathbf{t}} = v_{\mathbf{t}}\, \mathbf{in}\, x^{\mathbf{s}} \lhd_{\mathbf{r}} y^{\mathbf{t}} \quad (\nexists V_{\mathbf{t}}, v_{\mathbf{t}} = V_{\mathbf{t}}) \qquad \mu\boldsymbol{\alpha}.[\boldsymbol{\alpha}]v \to v$$

$$(\lambda^{\mathbf{s}} x^{\mathbf{t}}.v_{\mathbf{r}}) \lhd_{\mathbf{r}} V_{\mathbf{t}} \to v_{\mathbf{r}}\{V_{\mathbf{t}}/x^{\mathbf{t}}\} \qquad F[\mathbf{let}\, \boldsymbol{x} = v'\, \mathbf{in}\, v] \to \mathbf{let}\, \boldsymbol{x} = v'\, \mathbf{in}\, F[v]$$

$$(\Lambda^{\mathbf{s}} a^{\mathbf{t}}.v_{\mathbf{r}}) \lhd_{\mathbf{r}} A_{\mathbf{t}} \to v_{\mathbf{r}}\{A_{\mathbf{t}}/a^{\mathbf{t}}\} \qquad \mathbf{let}\, \boldsymbol{x} = v\, \mathbf{in}\, \mu\boldsymbol{\alpha}.[\boldsymbol{\beta}]v' \to \mu\boldsymbol{\alpha}.[\boldsymbol{\beta}](\mathbf{let}\, \boldsymbol{x} = v\, \mathbf{in}\, v')$$

$$\mathbf{let}\, \boldsymbol{x} = V\, \mathbf{in}\, v \to v\{V/\boldsymbol{x}\} \qquad [\boldsymbol{\beta}](\mu\boldsymbol{\alpha}.c) \to c\{\boldsymbol{\beta}/\boldsymbol{\alpha}\} \qquad F[\mu\boldsymbol{\alpha}.c] \to \mu\boldsymbol{\beta}.c\{[\boldsymbol{\beta}]F/[\boldsymbol{\alpha}]\square\}$$

Fig. 7. Rewriting theory for multi-discipline, polymorphic $\lambda\mu$-calculus.

$$V_{\mathbf{n}} ::= v_{\mathbf{n}} \qquad V_{\mathbf{v}} ::= x^{\mathbf{v}} \mid \lambda^{\mathbf{v}}\boldsymbol{x}.v \mid \Lambda^{\mathbf{v}}\boldsymbol{a}.v \qquad V_{\mathbf{lv}} ::= x^{\mathbf{lv}} \mid \lambda^{\mathbf{lv}}\boldsymbol{x}.v \mid \Lambda^{\mathbf{lv}}\boldsymbol{a}.v$$

$$F ::= \square \lhd_{\mathbf{r}} V \mid \square \lhd_{\mathbf{r}} A \mid \mathbf{let}\, x^{\mathbf{v}} = \square\, \mathbf{in}\, v \mid \mathbf{let}\, x^{\mathbf{lv}} = \square\, \mathbf{in}\, D[E[x^{\mathbf{lv}}]]$$

$$E ::= \square \mid F[E] \qquad D ::= \square \mid \mathbf{let}\, x^{\mathbf{lv}} = v\, \mathbf{in}\, D$$

Fig. 8. $\lambda\mu$ values and evaluation contexts in by-name ($\mathbf{n}$), -value ($\mathbf{v}$), and -need ($\mathbf{lv}$).

for **lv** call stacks applied to non-values (recall that we use $e \succ_{\varsigma^{\to}} e'$ to denote an application of the $\varsigma^{\to}$ rule as-is to e and *not* its compatible closure).

$$\frac{c \rightsquigarrow c'}{c \mapsto c'} \qquad \frac{e_{\mathbf{lv}} \succ_{\varsigma^{\to}} e'_{\mathbf{lv}}}{\langle v_{\mathbf{lv}} \| e_{\mathbf{lv}} \rangle \mapsto \langle v_{\mathbf{lv}} \| e'_{\mathbf{lv}} \rangle} \qquad \frac{c \mapsto c'}{D[c] \mapsto D[c']}$$

In other words, standard reduction can usually proceed at the top of a command except in the case where there are delayed lazy **lv** or **ln** bindings, in which case standard reduction computes under the binders. 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 $D[\langle x^{\mathbf{lv}} \| E_{\mathbf{lv}} \rangle]$ is x demanded by the fact that $\tilde{\mu}x^{\mathbf{lv}}.D[\langle x^{\mathbf{lv}} \| E_{\mathbf{lv}} \rangle]$ is a co-value that can be substituted for $\alpha^{\mathbf{lv}}$ in order to run c_1 to get the result of the term.

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 an **n-v-lv-ln-u-**disciplined sequent calculus command c is *done* exactly when $c = D[\langle \boldsymbol{x} \| \boldsymbol{\alpha} \rangle]$ and neither $\boldsymbol{x}$ nor $\boldsymbol{\alpha}$ are bound by D. We can now sketch out type safety in terms of progress and preservation.

Theorem 4 (Progress). *Given a multi-discipline sequent calculus command* $c : (x : \boldsymbol{a}, \ldots \vdash_{\Theta} \alpha : \boldsymbol{b}, \ldots)$*, either* c *is done or* $c \mapsto c'$ *for some* c'*.*

Proof. Consider the commands with no standard reduction, $c \not\mapsto$. The different cases for these commands are:

- Variable-co-variable pairs: where we have $D[\langle \boldsymbol{x} \| \boldsymbol{\alpha} \rangle] \not\mapsto$, which is another way of phrasing our above definition of done commands.
- Variable-call-stack pairs: where we have $D[\langle x^{\mathbf{s}} \| V \textcircled{s} E \rangle] \not\mapsto$ or $D[\langle x^{\mathbf{s}} \| A \textcircled{s} E \rangle] \not\mapsto$. But notice that for these commands to type check, we would need to have the variable $x^{\mathbf{s}} : A \xrightarrow{\mathbf{s}} B$ or $x^{\mathbf{s}} : \forall^{\mathbf{s}} \boldsymbol{a}.B$ in our environment, which is not allowed by the assumption that all free (co-)variables are typed by type variables. Therefore, this case cannot happen.
- λ-co-variable pairs: where we have $D[\langle \mu(\boldsymbol{x} \textcircled{s} \boldsymbol{\beta}).c \| \alpha^{\mathbf{s}} \rangle]$ or $D[\langle \mu(\boldsymbol{a} \textcircled{s} \boldsymbol{\beta}).c \| \alpha^{\mathbf{s}} \rangle]$. As before, this case requires free (co-)variables in the environment with function or universally polymorphic type, which is forbidden by assumption so this case cannot happen.
- 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 $D[\langle \mu(\boldsymbol{x} \textcircled{s} \boldsymbol{\beta}).c \| A \textcircled{s} E \rangle]$ and also where there is a discipline miss-matched β-reduction in a nonetheless well-disciplined command, as in $D[\langle \mu(x^{\mathbf{t}} \textcircled{s} \beta^{\mathbf{r}}).c \| V_{\mathbf{t}'} \textcircled{s} E_{\mathbf{r}'} \rangle]$ where either $\mathbf{t} \neq \mathbf{t}'$, or $\mathbf{r} \neq \mathbf{r}'$. 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 $A_{\mathbf{t}} \xrightarrow{\mathbf{s}} B_{\mathbf{r}}$ can only match with $A_{\mathbf{t}'} \xrightarrow{\mathbf{s}} B_{\mathbf{r}'}$ when $\mathbf{t} = \mathbf{t}'$ and $\mathbf{r} = \mathbf{r}'$—so they, too, cannot happen.

All other cases of commands can take a standard reduction.

Theorem 5 (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 a substitution lemma that states that substituting a value for a variable of the same type, a co-value for a co-variable of the same type, and a type for a type variable of the same kind preserves typing.

Multi-Discipline $\boldsymbol{\lambda\mu}$-Calculus

The operational semantics for the multi-discipline $\lambda\mu$-calculus—featuring the **v**, **n**, and **lv** disciplines—uses a two-part context. In particular, evaluation contexts E aren't enough to point out where the next reduction is due to laziness. Intuitively, evaluation contexts correspond to call stacks or continuations that can be captured by μ, but implementations of call-by-need also need a heap for storing delayed thunks. This heap can be represented by the delayed contexts (D) of lazy **lv let**-bindings from Fig. 8. With a representation of heaps as a context of delayed bindings, we can define the operational semantics over terms of the form $[\boldsymbol{\alpha}]D[E[v]]$, parenthesized as $[\boldsymbol{\alpha}](D[E[v]])$:

$$\begin{aligned}
[\boldsymbol{\alpha}]D[E[(\lambda^{\mathbf{s}} x^{\mathbf{t}}.v_{\mathbf{r}}) \mathbin{\triangleleft_{\mathbf{r}}} V_{\mathbf{t}}]] &\mapsto [\boldsymbol{\alpha}]D[E[v_{\mathbf{r}}\{V_{\mathbf{t}}/x^{\mathbf{t}}\}]] \\
[\boldsymbol{\alpha}]D[E[(\Lambda^{\mathbf{s}} a^{\mathbf{t}}.v_{\mathbf{r}}) \mathbin{\triangleleft_{\mathbf{r}}} A_{\mathbf{t}}]] &\mapsto [\boldsymbol{\alpha}]D[E[v_{\mathbf{r}}\{A_{\mathbf{t}}/a^{\mathbf{t}}\}]] \\
[\boldsymbol{\alpha}]D[E[v_{\mathbf{s}} \mathbin{\triangleleft_{\mathbf{r}}} v_{\mathbf{t}}]] &\mapsto [\boldsymbol{\alpha}]D[E[\mathbf{let}\, x^{\mathbf{s}} = v_{\mathbf{s}} \,\mathbf{in}\, \mathbf{let}\, y^{\mathbf{t}} = v_{\mathbf{t}} \,\mathbf{in}\, x^{\mathbf{s}} \mathbin{\triangleleft_{\mathbf{r}}} y^{\mathbf{t}}]] \\
&\text{if} \quad \nexists V_{\mathbf{t}}, v_{\mathbf{t}} = V_{\mathbf{t}} \\
[\boldsymbol{\alpha}]D[E[\mathbf{let}\, \boldsymbol{x} = V \,\mathbf{in}\, v]] &\mapsto [\boldsymbol{\alpha}]D[E[v\{V/\boldsymbol{x}\}]] \\
[\boldsymbol{\alpha}]D[E[\mathbf{let}\, x^{\mathbf{lv}} = v_{\mathbf{lv}} \,\mathbf{in}\, v']] &\mapsto [\boldsymbol{\alpha}]D[\mathbf{let}\, x^{\mathbf{lv}} = v_{\mathbf{lv}} \,\mathbf{in}\, E[v']] \qquad (E \neq \square) \\
[\boldsymbol{\alpha}]D[E[\mu\boldsymbol{\beta}.[\boldsymbol{\delta}]v]] &\mapsto [\boldsymbol{\delta}]D[v\{[\boldsymbol{\alpha}]E/[\boldsymbol{\beta}]\square\}]
\end{aligned}$$

Note that, unlike with the operational semantics for the sequent calculus, every standard $\lambda\mu$ reduction is simulated by one *or more* general reductions, *i.e.,* the transitive closure of $\to$ denoted by $\twoheadrightarrow^{+}$.

Theorem 6. *If $c \mapsto c'$ then $c \twoheadrightarrow^{+} c'$.*

Proof. Every standard reduction except for the rules which commute an evaluation context with a **let** or have a μ-abstraction capture its evaluation context map to exactly one general reduction. Those two cases proceed by induction on the structure of their context. For the μ capturing rule, we have

$$\begin{aligned}
[\boldsymbol{\alpha}]D[E[\mu\boldsymbol{\beta}.[\boldsymbol{\delta}]v]] &\twoheadrightarrow [\boldsymbol{\alpha}]D[\mu\boldsymbol{\beta}.[\boldsymbol{\delta}](v\{[\boldsymbol{\beta}]E/[\boldsymbol{\beta}]\square\})] \\
&\twoheadrightarrow [\boldsymbol{\alpha}]\mu\boldsymbol{\beta}.[\boldsymbol{\delta}]D[v\{[\boldsymbol{\beta}]E/[\boldsymbol{\beta}]\square\}] \\
&\to [\boldsymbol{\delta}]D[v\{[\boldsymbol{\alpha}]E/[\boldsymbol{\beta}]\square\}]
\end{aligned}$$

The case for commuting an evaluation context with a **let** is similar, and must make at least one reduction because of the assumption that the evaluation context is not empty.

Again, to state type safety we need to say what the expected results of the $\lambda\mu$ operational semantics are. Corresponding to the same definition for the sequent calculus, a multi-discipline $\lambda\mu$ command c is *done* exactly when $c = [\boldsymbol{\alpha}](D[\boldsymbol{x}])$ and $\boldsymbol{x}$ is not bound by D.

Theorem 7 (Progress). *Given a multi-discipline $\lambda\mu$ command $c : (x : \boldsymbol{a}, \dots \vdash_{\Theta} \alpha : \boldsymbol{b}, \dots)$, either c is done or $c \mapsto c'$ for some c'.*

Proof. As in Theorem 4, consider the commands with no standard reduction:

- A returned variable, $[\boldsymbol{\alpha}]D[\boldsymbol{x}] \not\mapsto$, which is done.
- A returned λ-abstraction, $[\boldsymbol{\alpha}]D[\lambda^{\mathbf{s}}\boldsymbol{x}.v]$ or $[\boldsymbol{\alpha}]D[\Lambda^{\mathbf{s}}\boldsymbol{a}.v]$, which is ruled out by assumption since typing these commands require $\boldsymbol{\alpha} : A \xrightarrow{\mathbf{s}} B$ or $\boldsymbol{\alpha} : \forall^{\mathbf{s}}\boldsymbol{a}.B$ in the environment.
- A free-variable stuck redex, $[\boldsymbol{\alpha}]D[E[F[\boldsymbol{x}]]] \not\mapsto$, which is only stuck when $F = \square \mathbin{\triangleleft_{\mathbf{s}}} V$ or $F = \square \mathbin{\triangleleft_{\mathbf{s}}} A$ and thus requires $\boldsymbol{x} : A \xrightarrow{\mathbf{s}} B$ or $\boldsymbol{x} : \forall^{\mathbf{s}}\boldsymbol{a}.B$ in the environment.

– A well-disciplined but ill-typed redex, like either $[\boldsymbol{\alpha}]D[E[(\lambda^{\mathbf{s}}\boldsymbol{x}.v) \lhd A]]$ or $[\boldsymbol{\alpha}]D[E[(\lambda^{\mathbf{s}}x^{\mathbf{t}}.v_{\mathbf{r}'}) \lhd V_{\mathbf{t}'}]]$ where $\mathbf{t} \neq \mathbf{t}'$ or $\mathbf{r} \neq \mathbf{r}'$, which are rejected by the typing rules.

All other cases of commands can take a standard reduction.

Theorem 8 (Preservation). *In the multi-discipline $\lambda\mu$-calculus*

1) if $c : (\Gamma \vdash_\Theta \Delta)$ and $c \to c'$ then $c' : (\Gamma \vdash_\Theta \Delta)$, and
2) if $\Gamma \vdash_\Theta v : A \mid \Delta$ and $v \to v'$ then $\Gamma \vdash_\Theta v' : A \mid \Delta$.

Proof. By induction on the given typing derivations, analogous to Theorem 5.

D The Five Disciplines are Admissible

Property 1. The **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 $D[c']$ 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 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 that the four facts:

1. if $E \to e'$ then e' is a co-value,
2. if $V \to v'$ then v' is a value,
3. if $D[\langle x^{\mathbf{lv}} \| E_{\mathbf{lv}} \rangle]$ is a hygienic decomposition and $D[\langle x^{\mathbf{lv}} \| E_{\mathbf{lv}} \rangle] \to c$ then there exists $D', E'_{\mathbf{lv}}$ such that $c = D'[\langle x^{\mathbf{lv}} \| E'_{\mathbf{lv}} \rangle]$ is a hygienic decomposition, and
4. if $D[\langle V_{\mathbf{ln}} \| \alpha^{\mathbf{ln}} \rangle]$ is a hygienic decomposition and $D[\langle V_{\mathbf{ln}} \| \alpha^{\mathbf{ln}} \rangle] \to c$ then there exists $D', V'_{\mathbf{ln}}$ such that $c = D'[\langle V'_{\mathbf{ln}} \| \alpha^{\mathbf{ln}} \rangle]$ is a hygienic decomposition.

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

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

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

E Proof of Strong Normalization

Pre-Types Rephrasing the pre-type orders in terms of set operations, let $\mathcal{A} = (A^+, A^-)$ and $\mathcal{B} = (B^+, B^-)$ be pre-types, where A^+ and B^+ are both sets of terms and A^- and B^- are both sets of co-terms. The two pre-type order relations are:

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

Likewise, the associated binary union operations ($\sqcup$ and $\vee$) and intersection operations ($\sqcap$ and $\wedge$) are:

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

So the union and intersection operations of containment and sub-typing cover all four possible combination of union and intersections on the sets underlying pre-types.

Property 2. Contrapositive: If $\mathcal{A} \sqsubseteq \mathcal{B}$ then $\mathcal{B}^\perp \sqsubseteq \mathcal{A}^\perp$. *Double orthogonal introduction*: $\mathcal{A} \sqsubseteq \mathcal{A}^{\perp\perp}$. *Triple orthogonal elimination*: $\mathcal{A}^{\perp\perp\perp} = \mathcal{A}^\perp$.

Proof.
- (*contrapositive*): Suppose that $\mathcal{A} \sqsubseteq \mathcal{B}$ and $v, e \in \mathcal{B}^\perp$, meaning that $\langle v \| e' \rangle \in \perp\!\!\!\perp$ and $\langle v' \| e \rangle \in \perp\!\!\!\perp$ for all $v', e' \in \mathcal{B}$. Since $\mathcal{A} \sqsubseteq \mathcal{B}$, we also know that $v', e' \in \mathcal{B}$ for all $v', e' \in \mathcal{A}$ by assumption, so $\langle v \| e' \rangle \in \perp\!\!\!\perp$ and $\langle v' \| e \rangle \in \perp\!\!\!\perp$. Therefore, $\mathcal{B}^\perp \sqsubseteq \mathcal{A}^\perp$.
- (*DOI*): Suppose that $v, e \in \mathcal{A}$ and $v', e' \in \mathcal{A}^\perp$, so that $\langle v \| e' \rangle \in \perp\!\!\!\perp$ and $\langle v' \| e \rangle \in \perp\!\!\!\perp$ by definition. Therefore $v, e \in \mathcal{A}^{\perp\perp}$ as well, so $\mathcal{A} \sqsubseteq \mathcal{A}^{\perp\perp}$.
- (*TOE*): First, note that $\mathcal{A}^\perp \sqsubseteq \mathcal{A}^{\perp\perp\perp}$ by applying double orthogonal introduction to $\mathcal{A}^\perp$. Furthermore, $\mathcal{A} \sqsubseteq \mathcal{A}^{\perp\perp}$ by applying double orthogonal introduction to $\mathcal{A}$ and so $\mathcal{A}^{\perp\perp\perp} \sqsubseteq \mathcal{A}^\perp$ by contrapositive. Therefore $\mathcal{A}^{\perp\perp\perp} = \mathcal{A}^\perp$.

Property 3. Monotonicity: If $\mathcal{A} \leq \mathcal{B}$ then $\mathcal{A}^\perp \leq \mathcal{B}^\perp$.

Proof. Suppose that $\mathcal{A} \leq \mathcal{B}$, $v \in \mathcal{A}^\perp$ and $e \in \mathcal{B}^\perp$, meaning that $\langle v \| e' \rangle \in \perp\!\!\!\perp$ and $\langle v' \| e \rangle \in \perp\!\!\!\perp$ for all $e' \in \mathcal{A}$ and $v' \in \mathcal{B}$. Since $\mathcal{A} \leq \mathcal{B}$ we know that for all $e' \in \mathcal{B}$, we have $e' \in \mathcal{A}$ so $\langle v \| e' \rangle \in \perp\!\!\!\perp$. Dually, we know that for all $v' \in \mathcal{A}$, we have $v' \in \mathcal{B}$ so $\langle v' \| e \rangle \in \perp\!\!\!\perp$. Therefore, $v \in \mathcal{B}^\perp$, $e \in \mathcal{A}^\perp$, and thus $\mathcal{A}^\perp \leq \mathcal{B}^\perp$.

Property 4. The unions ($\sqcup$, $\vee$) and intersections ($\sqcap$, $\wedge$) on pre-types are all: (1) associative, (2) commutative, (3) idempotent (*i.e.,* $\mathcal{A} \sqcup \mathcal{A} = \mathcal{A}$), (4) and monotonic in both arguments with respect to both the containment ($\sqsubseteq$) and sub-typing ($\leq$) orders on pre-types.

Proof. By the same properties of the union and intersection operations on the sets underlying pre-types.

Top Reduction

Lemma 1 (Nondeterministic Top Expansion). *If $\mathcal{A}$ is closed under reduction and for all $v, e \in \mathcal{A}$ and c, $\langle v \| e \rangle \rightsquigarrow c$ implies $c \in \perp\!\!\!\perp$, then $\mathcal{A}$ is safe.*

Proof. To show that $\mathcal{A} \sqsubseteq \mathcal{A}^{\perp}$, we must show that $v, e \in \mathcal{A}$ implies $\langle v \| e \rangle \in \perp\!\!\!\perp$, so let $v, e \in \mathcal{A}$. Note that $\langle v \| e \rangle \in \perp\!\!\!\perp$ if and only if $\langle v \| e \rangle \to c$ implies $c \in \perp\!\!\!\perp$. Also note that both v and e are strongly normalizing because they belong to a pre-type, and we denote the number of reductions in their longest reduction sequence by $|v|$ and $|e|$. We can thus demonstrate $\langle v \| e \rangle \in \perp\!\!\!\perp$ by induction on $|v| + |e|$ and cases on the possible reductions of $\langle v \| e \rangle$:

- if $\langle v \| e \rangle \rightsquigarrow c$ then $c \in \perp\!\!\!\perp$ by assumption,
- if $\langle v \| e \rangle \to \langle v' \| e \rangle$ because $v \to v'$ then $v' \in \mathcal{A}$ because $\mathcal{A}$ is closed under reduction and $|v| > |v'|$ so $\langle v' \| e \rangle \in \perp\!\!\!\perp$ by the inductive hypothesis, and
- if $\langle v \| e \rangle \to \langle v \| e' \rangle$ because $e \to e'$ then $e' \in \mathcal{A}$ because $\mathcal{A}$ is closed under reduction and $|e| > |e'|$ so $\langle v \| e' \rangle \in \perp\!\!\!\perp$ by the inductive hypothesis.

Since there are no other possible reductions, it follows that $\langle v \| e \rangle \in \perp\!\!\!\perp$.

Lemma 8 (Top Commutation). *For any stable $\mathbf{r}$, if $v_{\mathbf{r}} \twoheadrightarrow v'_{\mathbf{r}}$, $e_{\mathbf{r}} \twoheadrightarrow e'_{\mathbf{r}}$, and $\langle v_{\mathbf{r}} \| e_{\mathbf{r}} \rangle \rightsquigarrow_p c$ for $p \in \{+, -, 0\}$ then $\langle v'_{\mathbf{r}} \| e'_{\mathbf{r}} \rangle \rightsquigarrow_p^{=} c'$ and $c \twoheadrightarrow c'$ for some c'.*

Proof. The statement of this lemma is a summarization of the following facts about the top and general reduction theories:

1. If $\langle v \| e \rangle \rightsquigarrow_0 c$ then
   (a) $v \to v'$ implies $\langle v' \| e \rangle \rightsquigarrow_0 c' \twoheadleftarrow c$ for some c', and
   (b) $e \to e'$ implies $\langle v \| e' \rangle \rightsquigarrow_0 c' \twoheadleftarrow c$ for some c'.
2. If $\langle v \| e \rangle \rightsquigarrow_+ c$ then
   (a) $v \to v'$ implies $\langle v' \| e \rangle \rightsquigarrow_+^{=} c' \twoheadleftarrow c$ for some c', and
   (b) $e \to e'$ implies $\langle v \| e' \rangle \rightsquigarrow_+ c' \twoheadleftarrow c$ for some c'.
3. If $\langle v \| e \rangle \rightsquigarrow_- c$ then
   (a) $v \to v'$ implies $\langle v' \| e \rangle \rightsquigarrow_- c' \twoheadleftarrow c$ for some c', and
   (b) $e \to e'$ implies $\langle v \| e' \rangle \rightsquigarrow_-^{=} c' \twoheadleftarrow c$ for some c'.

Each of these facts can be checked by cases on the possible reductions of the term or co-term. The generalization to the assumption that $v \twoheadrightarrow v'$ and $e \twoheadrightarrow e'$ follows from the above facts by induction on the reflexive-transitive closure of those two reduction sequences. The cases for 1 follow because there are no critical pairs between neutral top reductions and non-top reductions, and the cases for 2(b), and 3(a) follow because stability of $\mathbf{r}$ means (co-)values are closed under reduction. For 2(a) and 3(b), the remaining interesting cases are:

- $\tilde{\mu} x^{\mathbf{r}}.\langle x^{\mathbf{r}} \| e_{\mathbf{r}} \rangle \to_{\eta_{\tilde{\mu}}} e_{\mathbf{r}}$: The only possible negative top reduction is identical:

$$\langle V_{\mathbf{r}} \| \tilde{\mu} x^{\mathbf{r}}.\langle x^{\mathbf{r}} \| e_{\mathbf{r}} \rangle \rangle \rightsquigarrow_- \langle V_{\mathbf{r}} \| e_{\mathbf{r}} \rangle$$

- $\mu \alpha^{\mathbf{r}}.\langle v_{\mathbf{r}} \| \alpha^{\mathbf{r}} \rangle \to_{\eta_\mu} v_{\mathbf{r}}$: Analogous to the $\eta_{\tilde{\mu}}$ case by duality.

- $\tilde{\mu}x^{\mathbf{r}}.c \to \tilde{\mu}x^{\mathbf{r}}.c'$ because $c \to c'$: The only possible top reduction is

$$\langle V_{\mathbf{r}} \| \tilde{\mu}x^{\mathbf{r}}.c\rangle \rightsquigarrow_{-} c\{V_{\mathbf{r}}/x^{\mathbf{r}}\}$$

  and since reduction commutes with substitution, $c\{V_{\mathbf{r}}/x^{\mathbf{r}}\} \to c'\{V_{\mathbf{r}}/x^{\mathbf{r}}\}$.
- $\mu\alpha^{\mathbf{r}}.c \to \mu\alpha^{\mathbf{r}}.c'$ because $c \to c'$: Analogous to the previous case by duality.
- $v_{\mathbf{t}} \textcircled{r}\, e \to_{\varsigma\to} \tilde{\mu}x^{\mathbf{r}}.\langle v_{\mathbf{t}} \| \tilde{\mu}y^{\mathbf{t}}.\langle x^{\mathbf{r}} \| y^{\mathbf{t}} \textcircled{r}\, e\rangle\rangle$: The only possible top reduction is identical.
- $V \textcircled{r}\, e_{\mathbf{t}} \to_{\varsigma\to} \tilde{\mu}x^{\mathbf{r}}.\langle \mu\beta^{\mathbf{t}}.\langle x^{\mathbf{r}} \| V \textcircled{r}\, \beta^{\mathbf{t}}\rangle \| e_{\mathbf{t}}\rangle$: The only possible top reduction is identical.
- $v_{\mathbf{t}} \textcircled{r}\, e \to v'_{\mathbf{t}} \textcircled{r}\, e$ because $v_{\mathbf{t}} \to v'_{\mathbf{t}}$: If neither $v_{\mathbf{t}}$ nor $v'_{\mathbf{t}}$ are values of $\mathbf{t}$ then the only possible negative top reduction is

$$\langle V_{\mathbf{r}} \| v_{\mathbf{t}} \textcircled{r}\, e\rangle \rightsquigarrow_{-} \langle V_{\mathbf{r}} \| \tilde{\mu}x^{\mathbf{r}}.\langle v_{\mathbf{t}} \| \tilde{\mu}y^{\mathbf{t}}.\langle x^{\mathbf{r}} \| y^{\mathbf{t}} \textcircled{r}\, e\rangle\rangle\rangle$$

  which commutes in one step on both sides. If both $v_{\mathbf{t}}$ and $v'_{\mathbf{t}}$ are $\mathbf{t}$ values then the only possible negative top reduction is

$$\langle V_{\mathbf{r}} \| v_{\mathbf{t}} \textcircled{r}\, e\rangle \rightsquigarrow_{-} \langle V_{\mathbf{r}} \| \tilde{\mu}x^{\mathbf{r}}.\langle \mu\boldsymbol{\beta}.\langle x^{\mathbf{r}} \| v_{\mathbf{t}} \textcircled{r}\, \boldsymbol{\beta}\rangle \| e\rangle\rangle$$

  which also commutes in one step on both sides. If $v_{\mathbf{t}}$ is not a $\mathbf{t}$ value but $v'_{\mathbf{t}} = V'_{\mathbf{t}}$ is then the only possible negative top reduction is still

$$\langle V_{\mathbf{r}} \| v_{\mathbf{t}} \textcircled{r}\, e\rangle \rightsquigarrow_{-} \langle V_{\mathbf{r}} \| \tilde{\mu}x^{\mathbf{r}}.\langle v_{\mathbf{t}} \| \tilde{\mu}y^{\mathbf{t}}.\langle x^{\mathbf{r}} \| y^{\mathbf{t}} \textcircled{r}\, e\rangle\rangle\rangle$$

  which commutes as follows:

$$\begin{aligned}\langle V_{\mathbf{r}} \| V'_{\mathbf{t}} \textcircled{r}\, e\rangle &\leftarrow_{\eta_{\tilde{\mu}}} \langle V_{\mathbf{r}} \| \tilde{\mu}x^{\mathbf{r}}.\langle x^{\mathbf{r}} \| V'_{\mathbf{t}} \textcircled{r}\, e\rangle\rangle \\ &\leftarrow_{\tilde{\mu}} \langle V_{\mathbf{r}} \| \tilde{\mu}x^{\mathbf{r}}.\langle V'_{\mathbf{t}} \| \tilde{\mu}y^{\mathbf{t}}.\langle x^{\mathbf{r}} \| y^{\mathbf{t}} \textcircled{r}\, e\rangle\rangle\rangle \\ &\leftarrow \langle V_{\mathbf{r}} \| \tilde{\mu}x^{\mathbf{r}}.\langle v_{\mathbf{t}} \| \tilde{\mu}y^{\mathbf{t}}.\langle x^{\mathbf{r}} \| y^{\mathbf{t}} \textcircled{r}\, e\rangle\rangle\rangle\end{aligned}$$

- $v \textcircled{r}\, e_{\mathbf{t}} \to v \textcircled{r}\, e'_{\mathbf{t}}$ because $e_{\mathbf{t}} \to e'_{\mathbf{t}}$: similar to the previous case.
- The cases for $A \textcircled{r}\, e$ are analogous to the cases for $v \textcircled{r}\, e$.

Lemma 2 (Deterministic top expansion). *If* $\mathbf{r}$ *is stable,* $v, e \in \mathcal{SN}_{\mathbf{r}}$*, either* $v \in \mathcal{DN}_{\mathbf{r}}$ *or* $e \in \mathcal{DN}_{\mathbf{r}}$*, and* $\langle v \| e\rangle \rightsquigarrow c \in ⫫$ *then* $\langle v \| e\rangle \in ⫫$*.*

Proof. The fact that $\langle v \| e\rangle \in ⫫$ follows from Lemma 1 by building a suitable pre-type containing v and e, namely, let $\mathcal{A}$ be

$$v' \in \mathcal{A} \iff v \twoheadrightarrow v' \qquad\qquad e' \in \mathcal{A} \iff e \twoheadrightarrow e'$$

$\mathcal{A}$ is closed under reduction by definition. Furthermore, for any $v', e' \in \mathcal{A}$, there must be a $c' \in ⫫$ such that $\langle v' \| e'\rangle \rightsquigarrow^{=} c' \twoheadleftarrow c \in ⫫$ by commutation (Lemma 8) and the fact that $⫫$ is closed under reduction as well. And if $\langle v' \| e'\rangle \rightsquigarrow c''$ then $c'' \in ⫫$ either because $\langle v' \| e'\rangle = c' \in ⫫$ or because $\langle v' \| e'\rangle \rightsquigarrow c' \in ⫫$ and either assumption $v \in \mathcal{DN}_{\mathbf{s}}$ or $e \in \mathcal{DN}_{\mathbf{s}}$. Therefore, $\mathcal{A} \sqsubseteq \mathcal{A}^{\perp}$, so $\langle v \| e\rangle \in ⫫$.

Reducibility Candidates

Corollary 2. *For any stable discipline* $\mathbf{r}$ *and pre-type* $\mathcal{A}$ *of* $\mathbf{r}$*,* $\mathcal{A}^s$ *is closed under reduction.*

Proof. This follows from commuting top and general reduction with Lemma 8 and the fact that $⫫$ is closed under reduction.

Lemma 3 (Reducibility fixed-point). *For any pre-type* $\mathcal{A}$ *of an admissible discipline* $\mathbf{r}$*,* $\mathcal{A}$ *is a reducibility candidate of* $\mathbf{r}$ *if and only if* $\mathcal{A} = \mathcal{A}^s$*.*

Proof. First, suppose that $\mathcal{A} \in CR_{\mathbf{r}}$, and note that $\mathcal{A}^{\perp} \sqsubseteq \mathcal{A}^s$ by definition, so $\mathcal{A}^s \sqsubseteq \mathcal{A} \sqsubseteq \mathcal{A}^{\perp} \sqsubseteq \mathcal{A}^s$ and thus $\mathcal{A}^s = \mathcal{A}$. Second, suppose that $\mathcal{A} = \mathcal{A}^s$. To show $\mathcal{A} \in CR_{\mathbf{r}}$ it suffices to show that $\mathcal{A} \sqsubseteq \mathcal{A}^{\perp}$, which follows from nondeterministic top expansion (Lemma 1). In particular, $\mathcal{A} = \mathcal{A}^s$ is closed under reduction (Corollary 2) because $\mathbf{r}$ is stable, and for all $v, e \in \mathcal{A}^s$:

- if $\langle v \| e \rangle \rightsquigarrow_{+} c$ then e *must* be a co-value, so $c \in ⫫$ because $e \in \mathcal{A}^s = \mathcal{A}$,
- if $\langle v \| e \rangle \rightsquigarrow_{-} c$ then v *must* be a value, so $c \in ⫫$ because $v \in \mathcal{A}^s = \mathcal{A}$, and
- if $\langle v \| e \rangle \rightsquigarrow_{0} c$ then *both* v is a value and e is a co-value because of the focalization property of $\mathbf{r}$, so $c \in ⫫$ because of both the above two reasons.

Fixed-Point Solutions To invoke the Knaster-Tarsky fixed point theorem in the construction of pre-types, we need to know that pre-types form a latice (actually, two separate lattices based on the two different orderings), and that the operations used in the construction are monotonic with respect to the lattice ordering.

Lemma 9 (Pre-type lattice). *Each of the* $\leq$ *and* $\sqsubseteq$ *orderings on the set of pre-types in a discipline form a complete lattice; that is, they have all joins and meets.*

Proof. The set of subsets of a set is a complete lattice ordered by $\subseteq$ with the usual $\bigcup$ and $\bigcap$ operations. Further, the dual of a complete lattice is itself a complete lattice, and the product of two complete lattices is a complete lattice. The case of $\sqsubseteq$ is the product of the two subset lattices; the case of $\leq$ is the product of the two subset lattices where one is dualized. Note that the largest and smallest pre-types of $\mathbf{r}$ with respect to containment are $\mathcal{SN}_{\mathbf{r}}$ and the empty pre-type $\emptyset = (\{\}, \{\})$, respectively, whereas the largest and smallest pre-types of $\mathbf{r}$ with respect to sub-typing are $(\{v \mid v \in \mathcal{SN}_{\mathbf{r}}\}, \{\})$ and $(\{\}, \{e \mid e \in \mathcal{SN}_{\mathbf{r}}\})$, respectively.

Property 5 (Subtype monotonicity). For any pre-types $\mathcal{A}$ and $\mathcal{B}$ of $\mathbf{r}$, if $\mathcal{A} \leq \mathcal{B}$ then $\mathcal{A}^s \leq \mathcal{B}^s$.

Proof. The monotonicity of saturation $(-^s)$ follows analogously to the monotonicity of orthogonality $(-^{\perp})$ from Property 3.

Lemma 4 (Fixed-point construction). *For every discipline* $\mathbf{r}$*, there is a function* $\mathcal{F}_{\mathbf{r}}(-)$ *such that for any pre-type* $\mathcal{C}$ *of* $\mathbf{r}$*,* $\mathcal{F}_{\mathbf{r}}(\mathcal{C}) = \mathcal{C} \sqcup \mathcal{F}_{\mathbf{r}}(\mathcal{C})^s$*.*

Proof. For any $\mathcal{C}$ of $\mathbf{r}$, there must be an $\mathcal{A}$ of $\mathbf{r}$ such that $\mathcal{A} = \mathcal{C} \sqcup \mathcal{A}^s$ by the Knaster-Tarski theorem since the sub-type ordering of pre-types forms a lattice (Lemma 9) and the function $\mathcal{C} \sqcup (-^s)$ is monotonic with respect to the sub-type ordering (Properties 4 and 5). In fact, there is a full sub-type lattice of such $\mathcal{A}$s, so define $\mathcal{F}_{\mathbf{r}}(\mathcal{C})$ to be the largest one with respect to the sub-type order.

Lemma 5 (Fixed-point validity). *If* $\mathcal{C} \sqsubseteq \mathcal{C}^{\perp dv}$ *then* $\mathcal{F}_{\mathbf{r}}(\mathcal{C}) = \mathcal{F}_{\mathbf{r}}(\mathcal{C})^s$*.*

Proof. First, note that for any pre-type $\mathcal{A}$ of $\mathbf{s}$, $\mathcal{A}^{\perp} \sqsubseteq \mathcal{A}^s$. Since $\mathcal{F}(\mathcal{C}) = \mathcal{C} \sqcup \mathcal{F}(\mathcal{C})^s$ by definition, it suffices to show that $\mathcal{C} \sqsubseteq \mathcal{F}(\mathcal{C})^{\perp}$. Suppose $V \in \mathcal{C}$ and $e \in \mathcal{F}(\mathcal{C}) = \mathcal{C} \sqcup \mathcal{F}(\mathcal{C})^s$, so we must show that $\langle V \| e \rangle \in \perp\!\!\!\perp$. If $e \in \mathcal{C}$ then $\langle V \| e \rangle \in \perp\!\!\!\perp$ because $V, e \in \mathcal{C} \sqsubseteq \mathcal{C}^{\perp}$. Otherwise, $e \in \mathcal{F}(\mathcal{C})^s$ so we know that $\langle V \| e \rangle \rightsquigarrow^{=}_{-,0} c \in \perp\!\!\!\perp$. The case where $\langle V \| e \rangle = c \in \perp\!\!\!\perp$ is immediate, and the case where $\langle V \| e \rangle \rightsquigarrow_{-,0} c \in \perp\!\!\!\perp$ follows from Lemma 2 because $V \in \mathcal{C} \sqsubseteq \mathcal{DN}_{\mathbf{s}}$. Therefore, for any $V \in \mathcal{C}$ and $e \in \mathcal{F}(\mathcal{C})$, $\langle V \| e \rangle \in \perp\!\!\!\perp$. Dually, for any $v \in \mathcal{F}(\mathcal{C})$ and $E \in \mathcal{C}$, $\langle v \| E \rangle \in \perp\!\!\!\perp$, so $\mathcal{C} \sqsubseteq \mathcal{F}(\mathcal{C})^{\perp}$.

The Meaning of Types

Lemma 6. *For any set* O *of deterministically-normalizing* $\mathbf{r}$*-co-values,* $\mathit{Neg}(O)^{dv} \sqsubseteq \mathit{Neg}(O)^{dv\perp dv} = \mathit{Neg}(O)^{dv\perp dv\perp dv}$*.*

Proof. First, for any $V, E \in \mathit{Neg}(O)^{dv}$, we have $\langle V \| E \rangle \in \perp\!\!\!\perp$ by definition so $V, E \in \mathit{Neg}(O)^{dv}$ implies $V, E \in \mathit{Neg}(O)^{dv\perp}$ meaning that $\mathit{Neg}(O)^{dv} \sqsubseteq \mathit{Neg}(O)^{dv\perp dv}$ by monotonicity and idempotence. Second, note that since $e \in O$ if and only if $E \in \mathit{Neg}(O)^{dv}$ by assumption, $V \in \mathit{Neg}(O)^{dv}$ if and only if $V \in \mathit{Neg}(O)^{dv\perp dv}$. It follows that $V, E \in V \in \mathit{Neg}(O)^{dv\perp dv}$ implies $V, E \in V \in \mathit{Neg}(O)^{dv\perp dv\perp dv}$, *i.e.,* $\mathit{Neg}(O)^{dv\perp dv} \sqsubseteq \mathit{Neg}(O)^{dv\perp dv\perp dv}$. Finally, we have $\mathit{Neg}(O)^{dv\perp dv\perp dv} \sqsubseteq \mathit{Neg}(O)^{dv\perp dv}$ by contrapositive and monotonicity of the previous $\mathit{Neg}(O)^{dv} \sqsubseteq \mathit{Neg}(O)^{dv\perp dv}$, so $\mathit{Neg}(O)^{dv} \sqsubseteq \mathit{Neg}(O)^{dv\perp dv} = \mathit{Neg}(O)^{dv\perp dv\perp dv}$.

Adequacy A (co-)term is strongly normalizing if and only if it is strongly normalizing when made into a command with a (co-)variable. While seemingly obvious, this is an important sanity check since some potential reductions could invalidate that property by allowing reductions only on commands which should really be reductions on the (co-)term themselves. For example, if the ς rules were defined to work on commands, like they are in top reduction, then this property would not necessarily hold.

Lemma 10. *1.* $v_{\mathbf{r}}$ *is strongly normalizing if and only if* $\langle v_{\mathbf{r}} \| \alpha^{\mathbf{r}} \rangle$ *is.*
2. $e_{\mathbf{r}}$ *is strongly normalizing if and only if* $\langle x_{\mathbf{r}} \| e^{\mathbf{r}} \rangle$ *is.*

Proof. 1. Since $v_{\mathbf{r}}$ is a sub-term of $\langle v_{\mathbf{r}} \| \alpha^{\mathbf{r}} \rangle$, strong normalization of $\langle v_{\mathbf{r}} \| \alpha^{\mathbf{r}} \rangle$ implies strong normalization of $v_{\mathbf{r}}$.

Going the other way, we show that every reduction of $\langle v_{\mathbf{r}} \| \alpha^{\mathbf{r}} \rangle$, except for possibly one top-level μ reduction, can be traced by $v_{\mathbf{r}}$ as well. We proceed to show that $\langle v_{\mathbf{r}} \| \alpha^{\mathbf{r}} \rangle$ is strongly normalizing because all of its reducts are by well-founded induction on the longest reduction sequence starting from $v_{\mathbf{r}}$, denoted by $|v_{\mathbf{r}}|$:

- Suppose $v_{\mathbf{r}} = \mu\beta^{\mathbf{r}}.c$, so that we have the top-level $\mu_{\mathbf{r}}$ reduction:

$$\langle \mu\beta^{\mathbf{r}}.c \| \alpha^{\mathbf{r}} \rangle \to_{\mu} c\{\alpha^{\mathbf{r}}/\beta^{\mathbf{r}}\}$$

  Furthermore, we know $\mu\alpha^{\mathbf{r}}.c\{\alpha^{\mathbf{r}}/\beta^{\mathbf{r}}\}$ is strongly normalizing since it is $\alpha^{\mathbf{r}}$-equivalent to the strongly normalizing $\mu\beta^{\mathbf{r}}.c$, which means that $c\{\alpha^{\mathbf{r}}/\beta^{\mathbf{r}}\}$ is also strongly normalizing since it is a sub-command of $\mu\alpha^{\mathbf{r}}.c\{\alpha^{\mathbf{r}}/\beta^{\mathbf{r}}\}$.
- Suppose we have some other reduction internal to $v_{\mathbf{r}}$, so that:

$$\langle v_{\mathbf{r}} \| \alpha^{\mathbf{r}} \rangle \to \langle v'_{\mathbf{r}} \| \alpha^{\mathbf{r}} \rangle$$

  because $v_{\mathbf{r}} \to v'_{\mathbf{r}}$. Then we know that $|v'_{\mathbf{r}}| < |v_{\mathbf{r}}|$, so by the inductive hypothesis, we get that $\langle v'_{\mathbf{r}} \| \alpha^{\mathbf{r}} \rangle$ is strongly normalizing.

Since every reduct of $\langle v_{\mathbf{r}} \| \alpha^{\mathbf{r}} \rangle$ is strongly normalizing, then $\langle v_{\mathbf{r}} \| \alpha^{\mathbf{r}} \rangle$ is also strongly normalizing.

2. Analogous to the above by duality.

Consequently, the largest pre-type $\mathcal{SN}_{\mathbf{r}}$ (with respect to containment) can be rephrased in terms of orthogonality and the pre-type $Var_{\mathbf{r}}$ of (co-)variables. This ensures that any pre-type $\mathcal{A}^{\perp} \sqsubseteq \mathcal{A}$ in $\mathbf{r}$ must contain (co-)variables, since $\mathcal{A} \sqsubseteq \mathcal{SN}_{\mathbf{r}}$ by definition and thus $\mathcal{SN}_{\mathbf{r}}^{\perp} \sqsubseteq \mathcal{A}^{\perp} \sqsubseteq \mathcal{A}$, which we need to derive strong normalization from soundness of the logical relation as is standard [8].

Corollary 3. *$\mathcal{SN}_{\mathbf{r}} = Var_{\mathbf{r}}^{\perp}$. Furthermore, if $\mathcal{A}^{\perp} \sqsubseteq \mathcal{A}$ then $Var_{\mathbf{r}} \sqsubseteq \mathcal{A}$ for any $\mathcal{A}$ of $\mathcal{R}$.*

Proof. The fact that $\mathcal{SN}_{\mathbf{r}} = Var_{\mathbf{r}}^{\perp}$ follows immediately from Lemma 10. Also, $Var_{\mathbf{r}} \sqsubseteq \mathcal{A}$ because $Var_{\mathbf{r}} \sqsubseteq Var_{\mathbf{r}}^{\perp\perp} = \mathcal{SN}_{\mathbf{r}}^{\perp} \sqsubseteq \mathcal{A}^{\perp} \sqsubseteq \mathcal{A} \sqsubseteq \mathcal{SN}_{\mathbf{r}}$ by double orthogonal introduction and contrapositive.

We now prove the meaning of the left and right activation, implication, and universal quantification rules. Each of the (co-)terms in the conclusion of the rules belong to the meaning of the stated type via saturation by demonstrating the following two facts (1) it is strongly normalizing on its own, and (2) it induces a top reduction to a strongly normalizing command for every other (co-)value in the meaning of the type.

Lemma 11. *For any reducibility candidate $\mathcal{A}$,*

- *if $c\{E/\boldsymbol{\alpha}\} \in \perp\!\!\!\perp$ for all $E \in \mathcal{A}$ then $\mu\boldsymbol{\alpha}.c \in \mathcal{A}$, and*
- *if $c\{V/\boldsymbol{x}\} \in \perp\!\!\!\perp$ for all $V \in \mathcal{A}$ then $\tilde{\mu}\boldsymbol{x}.c \in \mathcal{A}$.*

Proof. – First, note that $\boldsymbol{\alpha} \in \mathcal{A}$ by Corollary 3 so $c \in \perp\!\!\!\perp$ and thus $\mu\boldsymbol{\alpha}.c$ is strongly normalizing. Next, observe that for all $E \in \mathcal{A}$ we have the top reduction

$$\langle \mu\boldsymbol{\alpha}.c \| E\rangle \rightsquigarrow_{+} c\{E/\boldsymbol{\alpha}\} \in \perp\!\!\!\perp$$

so $\mu\boldsymbol{\alpha}.c \in \mathcal{A}^s \sqsubseteq \mathcal{A}$.

– Analogous to the previous case by duality.

Lemma 12. *For any admissible discipline* $\mathbf{r}$ *and reducibility candidates* $\mathcal{A}$, $\mathcal{B}$*,*

– *if* $v \in \mathcal{A}$ *and* $e \in \mathcal{B}$ *then* $v \textcircled{r} e \in \mathcal{A} \xrightarrow{\mathbf{r}} \mathcal{B}$*, and*
– *if* $c\{V/\boldsymbol{x}, E/\boldsymbol{\alpha}\} \in \perp\!\!\!\perp$ *for all* $V \in \mathcal{A}$ *and* $E \in \mathcal{B}$ *then* $\mu(\boldsymbol{x} \textcircled{r} \boldsymbol{\alpha}).c \in \mathcal{A} \xrightarrow{\mathbf{r}} \mathcal{B}$.

Proof. – First, note that for all $V \in \mathcal{A}$ and $E \in \mathcal{B}$, $V \textcircled{r} E \in \mathcal{A} \xrightarrow{\mathbf{r}} \mathcal{B}$ by Lemma 6 since $V \textcircled{r} E$ must be a deterministically-normalizing co-value of the focalizing discipline $\mathbf{r}$. Next, for all $V \in \mathcal{A}$, $e \in \mathcal{B}$ where e is not a co-value, and $V' \in \mathcal{A} \xrightarrow{\mathbf{r}} \mathcal{B}$, observe that we have the top reduction

$$\langle V' \| V \textcircled{r} e\rangle \rightsquigarrow_{-} \langle V' \| \tilde{\mu}\boldsymbol{x}.\langle \mu\boldsymbol{\beta}.\langle \boldsymbol{x} \| V \textcircled{r} \boldsymbol{\beta}\rangle \| e\rangle\rangle$$

where $\mu\boldsymbol{\beta}.\langle \boldsymbol{x} \| V \textcircled{r} \boldsymbol{\beta}\rangle \in \mathcal{B}$ and thus $\tilde{\mu}\boldsymbol{x}.\langle \mu\boldsymbol{\beta}.\langle \boldsymbol{x} \| V \textcircled{r} \boldsymbol{\beta}\rangle \| e\rangle \in \mathcal{A} \xrightarrow{\mathbf{r}} \mathcal{B}$ by Lemma 11, so $\langle V' \| \tilde{\mu}\boldsymbol{x}.\langle \mu\boldsymbol{\beta}.\langle \boldsymbol{x} \| V \textcircled{r} \boldsymbol{\beta}\rangle \| e\rangle\rangle \in \perp\!\!\!\perp$ forcing $V \textcircled{r} e \in (\mathcal{A} \xrightarrow{\mathbf{r}} \mathcal{B})^s = \mathcal{A} \xrightarrow{\mathbf{r}} \mathcal{B}$. Finally, for all $v \in \mathcal{A}$ where v is not a value, $e \in \mathcal{B}$, and $V' \in \mathcal{A} \xrightarrow{\mathbf{r}} \mathcal{B}$, observe that we have the top reduction

$$\langle V' \| v \textcircled{r} e\rangle \rightsquigarrow_{-} \langle V' \| \tilde{\mu}\boldsymbol{x}.\langle v \| \tilde{\mu}\boldsymbol{y}.\langle \boldsymbol{x} \| \boldsymbol{y} \textcircled{r} e\rangle\rangle\rangle$$

where $\tilde{\mu}\boldsymbol{y}.\langle \boldsymbol{x} \| \boldsymbol{y} \textcircled{r} e\rangle \in \mathcal{A}$ and thus $\tilde{\mu}\boldsymbol{x}.\langle v \| \tilde{\mu}\boldsymbol{y}.\langle \boldsymbol{x} \| \boldsymbol{y} \textcircled{r} e\rangle\rangle \in \mathcal{A} \xrightarrow{\mathbf{r}} \mathcal{B}$ by Lemma 11 so $\langle V' \| \tilde{\mu}\boldsymbol{x}.\langle v \| \tilde{\mu}\boldsymbol{y}.\langle \boldsymbol{x} \| \boldsymbol{y} \textcircled{r} e\rangle\rangle\rangle \in \perp\!\!\!\perp$ forcing $v \textcircled{r} e \in (\mathcal{A} \xrightarrow{\mathbf{r}} \mathcal{B})^s = \mathcal{A} \xrightarrow{\mathbf{r}} \mathcal{B}$.

– Let $O = \{V \textcircled{r} E \mid V \in \mathcal{A}, E \in \mathcal{B}\}$. First, note that $\boldsymbol{x} \in \mathcal{A}$ and $\boldsymbol{\alpha} \in \mathcal{B}$ by Corollary 3 so that $c \in \perp\!\!\!\perp$ and thus $\mu(\boldsymbol{x} \textcircled{r} \boldsymbol{\alpha}).c$ is strongly normalizing. Next, observe that for all $V \in \mathcal{A}$ and $E \in \mathcal{B}$ we have the top reduction

$$\langle \mu(\boldsymbol{x} \textcircled{r} \boldsymbol{\alpha}).c \| V \textcircled{r} E\rangle \rightsquigarrow_{0} c\{V/\boldsymbol{x}, E/\boldsymbol{\alpha}\} \in \perp\!\!\!\perp$$

and that $\mu(\boldsymbol{x} \textcircled{r} \boldsymbol{\alpha}).c$ is a deterministically-normalizing value of the discipline $\mathbf{r}$ so that $\langle \mu(\boldsymbol{x} \textcircled{r} \boldsymbol{\alpha}).c \| E'\rangle \in \perp\!\!\!\perp$ for all $E' \in O$ by deterministic top expansion (Lemma 2). Therefore, $\mu(\boldsymbol{x} \textcircled{r} \boldsymbol{\alpha}).c \in \mathit{Neg}(O) \sqsubseteq \mathcal{A} \xrightarrow{\mathbf{r}} \mathcal{B}$ by Lemma 6.

Lemma 13. *For any admissible discipline* $\mathbf{r}$ *and* $K \subseteq CR_{\mathbf{t}}$*,*

– *if* $e \in \mathcal{B} \in K$ *then* $A_{\mathbf{t}} \textcircled{r} e \in \forall^{\mathbf{r}}\mathbf{t}.K$*, and*
– *if* $c\{A_{\mathbf{t}}/a^{\mathbf{t}}, E/\boldsymbol{\alpha}\}$ *for all* $A_{\mathbf{t}}$ *and* $E \in \mathcal{B} \in K$ *then* $\mu(a^{\mathbf{t}} \textcircled{s} \boldsymbol{\alpha}).c \in \forall^{\mathbf{r}}\mathbf{t}.K$.

Proof. – First, note that for all types $A_{\mathbf{t}}$ and $E \in \mathcal{B} \in K$, $A_{\mathbf{t}} \textcircled{r} E \in \forall^{\mathbf{r}}\mathbf{t}.K$ by Lemma 6 since $A_{\mathbf{t}} \textcircled{r} E$ must be a deterministically-normalizing co-value of the focalizing discipline $\mathbf{r}$. Next, for all $A_{\mathbf{t}}$, $e \in \mathcal{B} \in K$ where e is not a co-value, and $V \in \forall^{\mathbf{r}}\mathbf{t}.K$, observe that we have the top reduction

$$\langle V \| A_{\mathbf{t}} \textcircled{r} e\rangle \rightsquigarrow_{-} \langle V \| \tilde{\mu}\boldsymbol{x}.\langle \mu\boldsymbol{\beta}.\langle \boldsymbol{x} \| A_{\mathbf{t}} \textcircled{r} \boldsymbol{\beta}\rangle \| e\rangle\rangle$$

where $\mu\boldsymbol{\beta}.\langle \boldsymbol{x} \| A_{\mathbf{t}} \textcircled{r} \boldsymbol{\beta}\rangle \in \mathcal{B}$ and thus $\tilde{\mu}\boldsymbol{x}.\langle \mu\boldsymbol{\beta}.\langle \boldsymbol{x} \| A_{\mathbf{t}} \textcircled{r} \boldsymbol{\beta}\rangle \| e\rangle \in \forall^{\mathbf{r}}\mathbf{t}.K$ by Lemma 11, so $\langle V' \| \tilde{\mu}\boldsymbol{x}.\langle \mu\boldsymbol{\beta}.\langle \boldsymbol{x} \| A_{\mathbf{t}} \textcircled{r} \boldsymbol{\beta}\rangle \| e\rangle\rangle \in \perp\!\!\!\perp$ forcing $A_{\mathbf{t}} \textcircled{r} e \in \forall^{\mathbf{r}}\mathbf{t}.K^{s} = \forall^{\mathbf{r}}\mathbf{t}.K$.

– Let $O = \{A_{\mathbf{t}} \textcircled{r} E \mid \mathcal{B} \in K, E \in \mathcal{B}\}$. First, note that $\boldsymbol{\alpha} \in \mathcal{B}$ by Corollary 3 so that $c \in \perp\!\!\!\perp$ and thus $\mu(a^{\mathbf{t}} \textcircled{r} \boldsymbol{\alpha}).c$ is strongly normalizing. Next, observe that for all $A_{\mathbf{t}}$ and $E \in \mathcal{B} \in K$ we have the top reduction

$$\langle \mu(a^{\mathbf{t}} \textcircled{r} \boldsymbol{\alpha}).c \| A_{\mathbf{t}} \textcircled{r} E\rangle \rightsquigarrow_{0} c\{A_{\mathbf{t}}/a^{\mathbf{t}}, E/\boldsymbol{\alpha}\} \in \perp\!\!\!\perp$$

and that $\mu(a^{\mathbf{t}} \textcircled{r} \boldsymbol{\alpha}).c$ is a deterministically-normalizing value of the discipline $\mathbf{r}$ so that $\langle \mu(a^{\mathbf{t}} \textcircled{r} \boldsymbol{\alpha}).c \| E'\rangle \in \perp\!\!\!\perp$ for all $E' \in O$ by deterministic top expansion (Lemma 2). Therefore, $\mu(a^{\mathbf{t}} \textcircled{r} \boldsymbol{\alpha}).c \in \mathit{Neg}(O) \sqsubseteq \forall^{\mathbf{r}}\mathbf{t}.K$ by Lemma 6.

Lemma 14. *Assuming all disciplines are admissible, if* $FV(A_{\mathbf{r}}) \subseteq \Theta$ *and* $\phi \in [\![\Theta]\!]$ *then* $[\![A_{\mathbf{r}}]\!]\phi \in \mathit{CR}_{\mathbf{r}}$.

Proof. By induction on the syntax of $A_{\mathbf{r}}$:

– $a^{\mathbf{r}}$: $\phi(a) \in \mathit{CR}_{\mathbf{r}}$ by definition of $\phi \in [\![\Theta]\!]$.
– $A \xrightarrow{\mathbf{r}} B$: by the inductive hypothesis and Lemmas 3, 5 and 6 since focalization means that $V \textcircled{r} E$ is always a deterministically-normalizing co-value.
– $\forall^{\mathbf{r}} a^{\mathbf{t}}.B$: by the inductive hypothesis and Lemmas 3, 5 and 6 since focalization means that $A \textcircled{r} E$ is always a deterministically-normalizing co-value.

Theorem 1 (Adequacy). *(1)* $c : (\Gamma \vdash_{\Theta} \Delta)$ *implies* $c : (\Gamma \vDash_{\Theta} \Delta)$*, (2)* $\Gamma \vdash_{\Theta} v : A \mid \Delta$ *implies* $\Gamma \vDash_{\Theta} v : A \mid \Delta$*, and (3)* $\Gamma \mid e : A \vdash_{\Theta} \Delta$ *implies* $\Gamma \mid e : A \vDash_{\Theta} \Delta$*.*

Proof. By mutual induction on the derivations of $c : (\Gamma \vdash_{\Theta} \Delta)$, $\Gamma \vdash_{\Theta} v : A \mid \Delta$, and $\Gamma \mid e : A \vdash_{\Theta} \Delta$:

– *Cut*: follows from the inductive hypothesis by Lemma 14 and the fact that $\mathcal{A} \sqsubseteq \mathcal{A}^{\perp\perp}$ for every reducibility candidate $\mathcal{A}$ by Definition 6.
– *Var* and *Co-Var*: follows by the definition of $[\![\Gamma \vdash_{\Theta} \Delta]\!]$.
– *Act* and *Co-Act*: follows from the inductive hypothesis and Lemma 11.
– $\rightarrow L$ and $\rightarrow R$: follows from the inductive hypothesis and Lemma 12.
– $\forall L$ and $\forall R$: follows from the inductive hypothesis and Lemma 13.

Corollary 1 (Strong normalization). *Typed commands, terms, and co-terms are strongly normalizing.*

Proof. By adequacy (Theorem 1), we get that $c : (\Gamma \vDash_\Theta \Delta)$, $\Gamma \vDash_\Theta v : A \mid \Delta$, and $\Gamma \mid e : A \vDash_\Theta \Delta$ which means that for all $\phi \in [\![\Theta]\!]$ and $\rho \in [\![\Gamma \vdash_\Theta \Delta]\!]\phi$, $c\{\rho\} \in \perp\!\!\!\perp$ and $v\{\rho\}, e\{\rho\} \in [\![A]\!]\phi$. We can witness particular $\phi_\emptyset \in [\![\Theta]\!]$ that sends every type variable to the reducibility candidate $\mathcal{F}_{\mathbf{r}}(\emptyset)$ (Lemma 5) of the appropriate $\mathbf{r}$, and we can witness that the identity substitution $\rho_{id} \in [\![\Gamma \vdash_\Theta \Delta]\!]\phi$ because type variables (of the appropriate $\mathbf{r}$) are substitutable types and (co-)variables (of the appropriate $\mathbf{r}$) are substitutable (co-)values in the meaning of every type (Corollary 3). Therefore, $c\{\rho_{id}\} = c \in \perp\!\!\!\perp$, $v\{\rho_{id}\} = v \in [\![A]\!]\phi$, and $e\{\rho_{id}\} = e \in [\![A]\!]\phi$ meaning that c, v, and e are strongly normalizing.

F Proof that Biorthogonals are Fixed Points

Lemma 7. *For any admissible discipline* $\mathbf{r}$ *where* $\mathcal{SN}_{\mathbf{r}} = \mathcal{DN}_{\mathbf{r}}$ *and pre-type* $\mathcal{A}$ *of* $\mathbf{r}$*,* $\mathcal{A}$ *is the* unique *reducibility candidate containing* $\mathcal{A}^v$ *if and only if* $\mathcal{A} = \mathcal{A}^{v\perp}$*.*

Proof. It is always the case that every reducibility candidate is a fixed point of $-^{v\perp}$. We have $\mathcal{A}^{\perp} \sqsubseteq \mathcal{A}^{v\perp} \sqsubseteq \mathcal{A}^s$ for any $\mathcal{A}$ by definition so when $\mathcal{A}$ is a reducibility candidate we also have $\mathcal{A} = \mathcal{A}^{v\perp}$ because $\mathcal{A}^{v\perp} \sqsubseteq \mathcal{A}^s \sqsubseteq \mathcal{A} \sqsubseteq \mathcal{A}^{\perp} \sqsubseteq \mathcal{A}^{v\perp}$.

What remains to show is that every fixed point of $-^{v\perp}$ is reducibility candidate of $\mathbf{r}$ when $\mathcal{SN}_{\mathbf{r}} = \mathcal{DN}_{\mathbf{r}}$, which follows from Lemma 3 and the fact that for every pre-type $\mathcal{A}$ of $\mathbf{r}$, $\mathcal{A}^s = \mathcal{A}^{v\perp}$. This holds because $\mathcal{SN}_{\mathbf{r}} = \mathcal{DN}_{\mathbf{r}}$ so deterministic top expansion applies to everything in $\mathcal{SN}_{\mathbf{r}}$. In particular, let $v \in \mathcal{A}^s$ and $E \in \mathcal{A}^v$, so that $\langle v \| E\rangle \rightsquigarrow_{+,0} c \in \perp\!\!\!\perp$. Either $\langle v \| E\rangle = c \in \perp\!\!\!\perp$, or $\langle v \| E\rangle \rightsquigarrow c \in \perp\!\!\!\perp$ so $\langle v \| E\rangle \in \perp\!\!\!\perp$ by deterministic top expansion because $v, E \in \mathcal{DN}_{\mathbf{r}}$. Dually, $\langle V \| e\rangle \in \perp\!\!\!\perp$ for all $e \in \mathcal{A}^s$ and $V \in \mathcal{A}^v$, so $\mathcal{A}^s \sqsubseteq \mathcal{A}^{v\perp} \sqsubseteq \mathcal{A}^s$ and thus $\mathcal{A}^s = \mathcal{A}^{v\perp}$. Therefore, all pre-types of $\mathbf{r}$ that are fixed points of $-^{v\perp}$ are exactly the fixed points of $-^s$ which in turn are the reducibility candidates of $\mathbf{r}$ (Lemma 3).

For the uniqueness of these reducibility candidates, suppose that we have some $\mathcal{A}, \mathcal{B} \in CR_{\mathbf{r}}$ such that $A^v \sqsubseteq \mathcal{B}^v$. It follows that $\mathcal{B} = \mathcal{A}^{v\perp} \sqsubseteq \mathcal{B}^{v\perp} = \mathcal{A}$ by contrapositive and the above fact, and also $\mathcal{A} = \mathcal{A}^{\perp} \sqsubseteq \mathcal{B}^{\perp} = \mathcal{B}$ by contrapositive of $\mathcal{B} \sqsubseteq \mathcal{A}$, so $\mathcal{A} = \mathcal{B}$.

Theorem 2. *Let* $\mathbf{r}$ *be any admissible discipline with deterministic top reduction (including* $\mathbf{v}$*,* $\mathbf{n}$*,* $\mathbf{lv}$*, and* $\mathbf{ln}$*, among others),* C *be a set of* $\mathbf{r}$*-values, and* O *be a set of* $\mathbf{r}$*-co-values. Both* $Pos(C)^{v\perp v\perp}$ *and* $Neg(O)^{v\perp v\perp}$ *are reducibility candidates of* $\mathbf{r}$*. Furthermore,* $Pos(C)^{\perp} \in CR_{\mathbf{v}}$ *if* $\mathbf{r} = \mathbf{v}$ *and* $Neg(O)^{\perp} \in CR_{\mathbf{n}}$ *if* $\mathbf{r} = \mathbf{n}$*.*

Proof. We begin by observing the fact that $\mathcal{SN}_{\mathbf{r}} = \mathcal{DN}_{\mathbf{r}}$ because the top reductions of $\mathbf{r}$ are all deterministic: for all $v_{\mathbf{r}}, e_{\mathbf{r}} \in \mathcal{SN}_{\mathbf{r}}$, if $\langle v_{\mathbf{r}} \| e_{\mathbf{r}}\rangle \rightsquigarrow c$ and $\langle v_{\mathbf{r}} \| e_{\mathbf{r}}\rangle \rightsquigarrow c'$ then either $c = c' \in \perp\!\!\!\perp$ or $c = c' \notin \perp\!\!\!\perp$. As a consequence, deterministic top expansion (Lemma 2) applies to any strongly-normalizing (co-)terms of $\mathbf{r}$: if $v, e \in \mathcal{SN}_{\mathbf{r}}$ and $\langle v \| e\rangle \rightsquigarrow c \in \perp\!\!\!\perp$ then $\langle v \| e\rangle \in \perp\!\!\!\perp$.

Next, observe that $Pos(C)^v \sqsubseteq Pos(C)^{v\perp v} = Pos(C)^{v\perp v\perp v}$ and $Neg(O)^v \sqsubseteq Neg(O)^{v\perp v} = Neg(O)^{v\perp v\perp v}$ analogously to Lemma 6. It follows that $Pos(C)^{v\perp v\perp} = Pos(C)^{v\perp v\perp v\perp}$ and $Neg(O)^{v\perp v\perp} = Neg(O)^{v\perp v\perp v\perp}$ are both fixed points of $-^{v\perp}$, and therefore both reducibility candidates of $\mathbf{r}$ by Lemma 7.

In the special case where $\mathbf{r} = \mathbf{n}$ and every term is a value, observe that $v \in \mathit{Neg}(O)$ if and only if $v \in \mathit{Neg}(O)^{\perp}$ by definition and if and only if $v \in \mathit{Neg}(O)^{\perp v}$ because $\mathbf{n}$ considers any v to be a value. Furthermore, if $e \in \mathit{Neg}(O)$ then $e \in \mathit{Neg}(O)^{\perp}$ by double orthogonal introduction and $e \in \mathit{Neg}(O)^{\perp v}$ because O contains only co-values. It follows that $\mathit{Neg}(O)^{\perp}$ is a fixed point of $-^{v\perp}$. First, suppose that $v, e \in \mathit{Neg}(O)^{\perp}$ and $v', e' \in \mathit{Neg}(O)^{\perp v}$. Both $\langle v \| e' \rangle \in \perp\!\!\!\perp$ and $\langle v' \| e \rangle \in \perp\!\!\!\perp$ because $v, v' \in \mathit{Neg}(O)$ so $\mathit{Neg}(O)^{\perp} \sqsubseteq \mathit{Neg}(O)^{\perp v \perp}$. Second, suppose that $v, e \in \mathit{Neg}(O)^{\perp v \perp}$ and $v', e' \in \mathit{Neg}(O)$. Both $\langle v \| e' \rangle \in \perp\!\!\!\perp$ and $\langle v' \| e \rangle \in \perp\!\!\!\perp$ because $v', e' \in \mathit{Neg}(O)^{\perp v}$ so $\mathit{Neg}(O)^{\perp v \perp} \sqsubseteq \mathit{Neg}(O)^{\perp}$. Therefore, $\mathit{Neg}(O)^{\perp} = \mathit{Neg}(O)^{\perp v \perp}$. Dually, in the special case where $\mathbf{r} = \mathbf{v}$ and every co-term is a co-value, $\mathit{Pos}(C)^{\perp} = \mathit{Pos}(C)^{\perp v \perp}$.

G Proof of $\boldsymbol{\lambda\mu}$ Strong Normalization

Here we prove that the multi-discipline $\lambda\mu$-calculus is strongly normalization by translation into the corresponding instance of the sequent calculus from Section 4. The main addition is that we identify a special case of the μ rule which is *linear*, which denote by the name $\underline{\mu}$, given as follows:

$$\langle \mu\boldsymbol{\alpha}.C[\boldsymbol{\alpha}] \| E \rangle \to_{\underline{\mu}} C[E] \qquad\qquad (\boldsymbol{\alpha} \notin FV(C))$$

where C is an arbitrary context such that the left- and right-hand sides of the above rule are well-disciplined commands. We write $\to_{\underline{\mu}}$ for the compatible closure of the above rule, $\twoheadrightarrow_{\underline{\mu}}$ for the reflexive-transitive closure of $\to_{\underline{\mu}}$, and $=_{\underline{\mu}}$ for the reflexive-transitive-symmetric closure of $\to_{\underline{\mu}}$. We will also write $\underline{\mu\boldsymbol{\alpha}.c}$ for a μ-abstraction in which $\boldsymbol{\alpha}$ appears *exactly once* in c—that is, c has the form $C[\boldsymbol{\alpha}]$ where $\boldsymbol{\alpha}$ is not free in C—to accentuate the fact that the underlined μ-abstraction can be subject to $\underline{\mu}$-reduction.

The purpose of pointing out the special $\underline{\mu}$ case of the more general μ rule is that $\underline{\mu}$-reduction is particularly well-behaved.

Lemma 15. *$\to_{\underline{\mu}\eta_\mu}$ reduction is confluent and strongly normalizing (even on untyped commands and (co-)terms). It follows that the $=_{\underline{\mu}\eta_\mu}$ relation is the same as the $\twoheadrightarrow_{\underline{\mu}\eta_\mu}\twoheadleftarrow_{\underline{\mu}\eta_\mu}$ relation.*

Proof. The confluence of $\underline{\mu}\eta_\mu$-reduction follows because the only critical pair is trivial (it converges in zero steps). Strong normalization follows from the fact that the $\underline{\mu}\eta_\mu$ rules are both left and right linear, so every reduction eliminates exactly one μ-abstraction since no duplication can occur.

Next, notice how $\underline{\mu}\eta_\mu$-reductions can never delete any other more serious reductions, which we denote by $c \to_! c'$ if and only if $c \to c'$ and $c \not\to_{\underline{\mu}\eta_\mu} c'$, and similarly for (co-)terms.

Lemma 16 (Commutation). *If $c_1 \leftarrow_{\underline{\mu}\eta_\mu} c \to_! c_2$ then $c_1 \to_! c' \twoheadleftarrow_{\underline{\mu}\eta_\mu} c_2$ for some c', and similarly for (co-)terms. Consequently, if $c_1 =_{\underline{\mu}\eta_\mu} c \to_! c_2$ then $c_1 \twoheadrightarrow_{\underline{\mu}\eta_\mu} \to_! c' \twoheadleftarrow_{\underline{\mu}\eta_\mu} c_2$ for some c', and similarly for (co-)terms.*

Proof. For the first fact about $\underline{\mu}\eta_\mu$-reduction, notice that $\underline{\mu}\eta_\mu$-reduction does not erase any sub-expression, but other general reductions can duplicate or erase the $\underline{\mu}\eta_\mu$ reduction. And we know that c_1 is not the same as c_2 by the definition of $\rightarrow_!$. The second fact about $\underline{\mu}\eta_\mu$-equivalence follows from the previous one and Lemma 15.

We now define the translation from the multi-discipline $\lambda\mu$-calculus into the **n**, **v**, and **lv** instance of the parametric, multi-discipline sequent calculus, which is analogous to the standard compositional translation from natural deduction to sequent calculus, as follows:

$$
\begin{aligned}
[\![\boldsymbol{x}]\!] &= \boldsymbol{x} \\
[\![\lambda^{\mathbf{s}}\boldsymbol{x}.v_{\mathbf{r}}]\!] &= \mu(\boldsymbol{x} \circledS \alpha^{\mathbf{r}}).\langle [\![v]\!] \| \alpha^{\mathbf{r}} \rangle \\
[\![\Lambda^{\mathbf{s}}\boldsymbol{a}.v_{\mathbf{r}}]\!] &= \mu(\boldsymbol{a} \circledS \alpha^{\mathbf{r}}).\langle [\![v]\!] \| \alpha^{\mathbf{r}} \rangle \\
[\![v_{\mathbf{s}} \lhd_{\mathbf{r}} v']\!] &= \underline{\mu}\alpha^{\mathbf{r}}.\langle [\![v_{\mathbf{s}}]\!] \| [\![v']\!] \circledS \alpha^{\mathbf{r}} \rangle \\
[\![v_{\mathbf{s}} \lhd_{\mathbf{r}} A]\!] &= \underline{\mu}\alpha^{\mathbf{r}}.\langle [\![v_{\mathbf{s}}]\!] \| A \circledS \alpha^{\mathbf{r}} \rangle \\
[\![\mathbf{let}\, \boldsymbol{x} = v' \,\mathbf{in}\, v_{\mathbf{r}}]\!] &= \underline{\mu}\alpha^{\mathbf{r}}.\langle [\![v']\!] \| \tilde{\mu}\boldsymbol{x}.\langle [\![v_{\mathbf{r}}]\!] \| \alpha^{\mathbf{r}} \rangle \rangle \\
[\![\mu\boldsymbol{\alpha}.c]\!] &= \mu\boldsymbol{\alpha}.[\![c]\!] \\
[\![[\boldsymbol{\alpha}]v]\!] &= \langle v \| \boldsymbol{\alpha} \rangle
\end{aligned}
$$

Note how, except for the μ-abstractions that originally came from the $\lambda\mu$-term, all the extra μs generated by the translation are linear $\underline{\mu}$-abstractions. We will also define the following translation of $\lambda\mu$ frame (F), evaluation (E), and delayed (D) contexts into sequent calculus **s**-co-values and contexts:

$$
\begin{aligned}
[\![\Box \lhd_{\mathbf{r}} V]\!]^{\mathbf{s}}_{E_{\mathbf{r}}} &= [\![V]\!] \circledS E_{\mathbf{r}} \\
[\![\Box \lhd_{\mathbf{r}} A]\!]^{\mathbf{s}}_{E_{\mathbf{r}}} &= A \circledS E_{\mathbf{r}} \\
[\![\mathbf{let}\, x^{\mathbf{v}} = \Box \,\mathbf{in}\, v_{\mathbf{r}}]\!]^{\mathbf{v}}_{E_{\mathbf{r}}} &= \mu x^{\mathbf{v}}.\langle [\![v_{\mathbf{r}}]\!] \| E_{\mathbf{r}} \rangle \\
[\![\mathbf{let}\, x^{\mathbf{lv}} = \Box \,\mathbf{in}\, D[E'[x^{\mathbf{lv}}]]_{\mathbf{r}}]\!]^{\mathbf{lv}}_{E_{\mathbf{r}}} &= \mu x^{\mathbf{lv}}.[\![D]\!][\langle x^{\mathbf{lv}} \| [\![E']\!]_{E_{\mathbf{r}}} \rangle] \\
[\![\Box]\!]^{\mathbf{r}}_{E_{\mathbf{r}}} &= E_{\mathbf{r}} \\
[\![F[E']]\!]^{\mathbf{s}}_{E_{\mathbf{r}}} &= [\![E']\!]^{\mathbf{s}}_{[\![F]\!]^{\mathbf{t}}_{E_{\mathbf{r}}}} && (E'[x^{\mathbf{s}}] \in \mathit{Term}_{\mathbf{t}}) \\
[\![\Box]\!] &= \Box \\
[\![\mathbf{let}\, x^{\mathbf{lv}} = v_{\mathbf{lv}} \,\mathbf{in}\, D]\!] &= \langle [\![v_{\mathbf{lv}}]\!] \| \tilde{\mu}x^{\mathbf{lv}}.[\![D]\!] \rangle
\end{aligned}
$$

First off, translation preserves typing from the $\lambda\mu$-calculus to the sequent calculus.

Lemma 17. *1) If $\Gamma \vdash_\Theta v : A \mid \Delta$ in $\lambda\mu$ then $\Gamma \vdash_\Theta [\![v]\!] : A \mid \Delta$ in the sequent calculus.*
2) If $c : (\Gamma \vdash_\Theta \Delta)$ in $\lambda\mu$ then $[\![c]\!] : (\Gamma \vdash_\Theta \Delta)$ in the sequent calculus.

Proof. By induction on the given typing derivation.

Translation also preserves the notion of value and co-value (*i.e.,* evaluation context) from the $\lambda\mu$-calculus into the sequent calculus. Additionally, $\underline{\mu}$-reduction is enough to push frame, evaluation, and delayed contexts out of the way.

Lemma 18. *1) $V_{\mathbf{s}}$ is a $\lambda\mu$ $\mathbf{s}$-value if and only if $[\![V_{\mathbf{s}}]\!]$ is a sequent calculus $\mathbf{s}$-value.*
2) If F is a $\lambda\mu$ frame context such that $F[x^{\mathbf{s}}]$ is an $\mathbf{r}$-term then $[\![F]\!]^{\mathbf{s}}_{E_{\mathbf{r}}}$ is a sequent calculus $\mathbf{s}$-co-value.
3) If E is a $\lambda\mu$ evaluation context such that $E[x^{\mathbf{s}}]$ is an $\mathbf{r}$-term then $[\![E]\!]^{\mathbf{s}}_{E'_{\mathbf{r}}}$ is a sequent calculus $\mathbf{r}$-co-value.
4) If D is a $\lambda\mu$ delayed context then $[\![D]\!]$ is a sequent calculus delayed context.
5) $[\![F[v_{\mathbf{s}}]_{\mathbf{r}}]\!] \twoheadrightarrow_{\underline{\mu}} \underline{\mu}\alpha^{\mathbf{r}}.\langle [\![v_{\mathbf{s}}]\!] \| [\![F]\!]^{\mathbf{s}}_{\alpha^{\mathbf{r}}}\rangle$.
6) $\langle [\![E'[v_{\mathbf{s}}]_{\mathbf{r}}]\!] \| E_{\mathbf{r}}\rangle \twoheadrightarrow_{\underline{\mu}} \langle [\![v_{\mathbf{s}}]\!] \| [\![E']\!]^{\mathbf{s}}_{E_{\mathbf{r}}}\rangle$.
7) $\langle [\![D[v_{\mathbf{r}}]]\!] \| E_{\mathbf{r}}\rangle \twoheadrightarrow_{\underline{\mu}} [\![D]\!][\langle [\![v_{\mathbf{r}}]\!] \| E_{\mathbf{r}}\rangle]$.

Proof. Facts 1-4 follow by induction on the syntax of values and frame, evaluation, and delayed contexts. Facts 5-6 follow from facts 1-4 by induction on the syntax of frame, evaluation, and delayed contexts. The most interesting case is for the call-by-need frame context $\mathbf{let}\, x^{\mathbf{lv}} = \square \,\mathbf{in}\, D[E[x^{\mathbf{lv}}]]$ which proceeds as follows:

$$\begin{aligned}
[\![\mathbf{let}\, x^{\mathbf{lv}} = v_{\mathbf{lv}} \,\mathbf{in}\, D[E[x^{\mathbf{lv}}]]]\!] &= \underline{\mu}\boldsymbol{\alpha}.\langle [\![v_{\mathbf{lv}}]\!] \| \tilde{\mu}x^{\mathbf{lv}}.\langle [\![D[E[x^{\mathbf{lv}}]]]\!] \| \boldsymbol{\alpha}\rangle\rangle \\
&\twoheadrightarrow_{\underline{\mu}} \underline{\mu}\boldsymbol{\alpha}.\langle [\![v_{\mathbf{lv}}]\!] \| \tilde{\mu}x^{\mathbf{lv}}.[\![D]\!][\langle [\![E[x^{\mathbf{lv}}]]\!] \| \boldsymbol{\alpha}\rangle]\rangle \\
&\twoheadrightarrow_{\underline{\mu}} \underline{\mu}\boldsymbol{\alpha}.\langle [\![v_{\mathbf{lv}}]\!] \| \tilde{\mu}x^{\mathbf{lv}}.[\![D]\!][\langle x^{\mathbf{lv}} \| [\![E]\!]^{\mathbf{lv}}_{\boldsymbol{\alpha}}\rangle]\rangle \\
&= \underline{\mu}\boldsymbol{\alpha}.\langle [\![v_{\mathbf{lv}}]\!] \| [\![\mathbf{let}\, x^{\mathbf{lv}} = \square \,\mathbf{in}\, D[E[x^{\mathbf{lv}}]]]\!]^{\mathbf{lv}}_{\boldsymbol{\alpha}}\rangle
\end{aligned}$$

We also have a standard substitution lemma that commutes translation with the three different kinds of substitution (variable substitution, co-variable substitution, and structural substitution) of the $\lambda\mu$-calculus.

Lemma 19. *For M ranging over both $\lambda\mu$ commands and terms,*

1) $[\![M\{V_{\mathbf{s}}/x^{\mathbf{s}}\}]\!] = [\![M]\!]\{[\![V_{\mathbf{s}}]\!]/x^{\mathbf{s}}\}$,
2) $[\![M\{\beta^{\mathbf{s}}/\alpha^{\mathbf{s}}\}]\!] = [\![M]\!]\{\beta^{\mathbf{s}}/\alpha^{\mathbf{s}}\}$, and
3) $[\![M\{[\beta^{\mathbf{r}}]E/[\alpha^{\mathbf{s}}]\}]\!] \twoheadrightarrow_{\underline{\mu}} [\![M]\!]\{[\![E]\!]^{\mathbf{s}}_{\beta^{\mathbf{r}}}/\alpha^{\mathbf{s}}\}$.

Proof. Facts (1) and (2) are immediate by induction on the syntax of $\lambda\mu$ commands and terms since $[\![x^{\mathbf{s}}]\!] = x^{\mathbf{s}}$ and $[\![\,[\alpha^{\mathbf{s}}]v_{\mathbf{s}}]\!] = \langle [\![v_{\mathbf{s}}]\!] \| \alpha^{\mathbf{s}}\rangle$. Fact (3) is a little more interesting since some $\underline{\mu}$-reductions are involved in the base case, which reduces as follows:

$$\begin{aligned}
[\![([\alpha^{\mathbf{s}}]v_{\mathbf{s}})\{[\beta^{\mathbf{r}}]E/[\alpha^{\mathbf{s}}]\}]\!] &= \llbracket [\beta^{\mathbf{r}}]\,(E[v_{\mathbf{s}}\{[\beta^{\mathbf{r}}]E/[\alpha^{\mathbf{s}}]\}])]\!] \\
&= \langle [\![E[v_{\mathbf{s}}\{[\beta^{\mathbf{r}}]E/[\alpha^{\mathbf{s}}]\}]]\!] \| \beta^{\mathbf{r}}\rangle \\
&\twoheadrightarrow_{\underline{\mu}} \langle [\![v_{\mathbf{s}}\{[\beta^{\mathbf{r}}]E/[\alpha^{\mathbf{s}}]\}]\!] \| [\![E]\!]^{\mathbf{s}}_{\beta^{\mathbf{r}}}\rangle && (\textit{Lemma } 18) \\
&\twoheadrightarrow_{\underline{\mu}} \langle [\![v_{\mathbf{s}}]\!]\{[\![E]\!]^{\mathbf{s}}_{\beta^{\mathbf{r}}}/\alpha^{\mathbf{s}}\} \| [\![E]\!]^{\mathbf{s}}_{\beta^{\mathbf{r}}}\rangle && (IH) \\
&= \langle [\![v_{\mathbf{s}}]\!] \| \alpha^{\mathbf{s}}\rangle\{[\![E]\!]^{\mathbf{s}}_{\beta^{\mathbf{r}}}/\alpha^{\mathbf{s}}\} \\
&= [\![\,[\alpha^{\mathbf{s}}]v_{\mathbf{s}}]\!]\{[\![E]\!]^{\mathbf{s}}_{\beta^{\mathbf{r}}}/\alpha^{\mathbf{s}}\}
\end{aligned}$$

Analogous to the $\underline{\eta}_\mu$ reductions in the sequent calculus, we call out the $\lambda\mu$ reductions that are *administrative* in nature, denoted by $\to_{ad}$ and defined as follows:

$$
\begin{aligned}
F[\mathbf{let}\,\boldsymbol{x} = v'\,\mathbf{in}\,v] &\to_{ad} \mathbf{let}\,\boldsymbol{x} = v'\,\mathbf{in}\,F[v] \\
\mathbf{let}\,\boldsymbol{x} = v'\,\mathbf{in}\,\mu\boldsymbol{\alpha}.[\boldsymbol{\beta}]v &\to_{ad} \mu\boldsymbol{\alpha}.[\boldsymbol{\beta}](\mathbf{let}\,\boldsymbol{x} = v'\,\mathbf{in}\,v) \\
[\boldsymbol{\beta}](\mu\boldsymbol{\alpha}.C[\boldsymbol{\alpha}]) &\to_{ad} C[\boldsymbol{\beta}] && (\boldsymbol{\alpha} \notin FV(C)) \\
F[\mu\boldsymbol{\alpha}.C[[\boldsymbol{\alpha}]v]] &\to_{ad} \mu\boldsymbol{\beta}.C[[\boldsymbol{\beta}]F[v]] && (\boldsymbol{\alpha} \notin FV(C)) \\
\mu\boldsymbol{\alpha}.[\boldsymbol{\alpha}]v &\to_{ad} v && (\boldsymbol{\alpha} \notin v)
\end{aligned}
$$

where the Cs that appear in the rules stand for arbitrary contexts such that the overall command and term are well-formed and well-disciplined. A *serious* $\lambda\mu$ reduction, denoted by $\to_!$, is any non-administrative reduction: $v \to_! v'$ if and only if $v \to v'$ and $v \not\to_{ad} v'$, and similarly for commands.

Lemma 20. *Administrative $\lambda\mu$ reductions are strongly normalizing even for untyped commands and terms.*

Proof. The main reason is that administrative reduction can never duplicate any expression, so they all actually reduce something in the term or command; either eliminating a μ, eliminating a frame context around a **let** or a μ by pushing it inward, or eliminating a **let** around a μ by pushing it inward. The "inward pushing" actions cannot go on forever by induction on the syntax of the term.

We can now show that translation maps every $\lambda\mu$ reduction to exactly one sequent calculus reduction, up to $\underline{\mu}\eta_\mu$-equality.

Lemma 21. *1) If $v \to_{ad} v'$ then $[\![v]\!] =_{\underline{\mu}\eta_\mu} [\![v']\!]$ and similarly for commands.*
2) If $v \to_! v'$ then $[\![v]\!] =_{\underline{\mu}\eta_\mu}\to_!=_{\underline{\mu}\eta_\mu} [\![v']\!]$ and similarly for commands.

Proof. The translation function is compositional, so we only need to check the reduction axioms themselves. Also by compositionality, we get for free a translation of general $\lambda\mu$ contexts C such that $[\![C[v]]\!] = [\![C]\!][[\![v]\!]]$ and $[\![C[c]]\!] = [\![C]\!][[\![c]\!]]$. The cases for the administrative $\lambda\mu$ reductions are:

– $F[\mathbf{let}\,\boldsymbol{x} = v'\,\mathbf{in}\,v_{\mathbf{s}}] \to_{ad} \mathbf{let}\,\boldsymbol{x} = v'\,\mathbf{in}\,F[v_{\mathbf{s}}]$:

$$
\begin{aligned}
[\![F[\mathbf{let}\,\boldsymbol{x} = v'\,\mathbf{in}\,v_{\mathbf{s}}]]\!] &\twoheadrightarrow_{\underline{\mu}} \underline{\mu}\boldsymbol{\alpha}.\langle [\![\mathbf{let}\,\boldsymbol{x} = v'\,\mathbf{in}\,v_{\mathbf{s}}]\!] \| [\![F]\!]^{\mathbf{s}}_{\boldsymbol{\alpha}}\rangle && (\textit{Lemma } 18) \\
&= \underline{\mu}\boldsymbol{\alpha}.\langle \underline{\mu}\beta^{\mathbf{s}}.\langle [\![v']\!] \| \tilde{\mu}\boldsymbol{x}.\langle [\![v_{\mathbf{s}}]\!] \| \beta^{\mathbf{s}}\rangle\rangle \| [\![F]\!]^{\mathbf{s}}_{\boldsymbol{\alpha}}\rangle \\
&\to_{\underline{\mu}} \underline{\mu}\boldsymbol{\alpha}.\langle [\![v']\!] \| \tilde{\mu}\boldsymbol{x}.\langle [\![v_{\mathbf{s}}]\!] \| [\![F]\!]^{\mathbf{s}}_{\boldsymbol{\alpha}}\rangle\rangle \\
&\twoheadleftarrow_{\underline{\mu}} \underline{\mu}\boldsymbol{\alpha}.\langle [\![v']\!] \| \tilde{\mu}\boldsymbol{x}.\langle [\![F[v_{\mathbf{s}}]]\!] \| \boldsymbol{\alpha}\rangle\rangle \\
&= [\![\mathbf{let}\,\boldsymbol{x} = v'\,\mathbf{in}\,F[v_{\mathbf{s}}]]\!]
\end{aligned}
$$

– $\mathbf{let}\,\boldsymbol{x} = v\,\mathbf{in}\,\mu\boldsymbol{\alpha}.[\boldsymbol{\beta}]v' \to_{ad} \mu\boldsymbol{\alpha}.[\boldsymbol{\beta}](\mathbf{let}\,\boldsymbol{x} = v\,\mathbf{in}\,v')$:

$$
[\![\mathbf{let}\,\boldsymbol{x} = v\,\mathbf{in}\,\mu\boldsymbol{\alpha}.[\boldsymbol{\beta}]v']\!] = \underline{\mu}\boldsymbol{\alpha}.\langle [\![v]\!] \| \tilde{\mu}\boldsymbol{x}.\langle \mu\boldsymbol{\alpha}.\langle [\![v']\!] \| \boldsymbol{\beta}\rangle \| \boldsymbol{\alpha}\rangle\rangle
$$

$$\begin{aligned}
&\to_{\underline{\mu}} \mu\boldsymbol{\alpha}.\langle \llbracket v\rrbracket \| \tilde{\mu}\boldsymbol{x}.\langle \llbracket v'\rrbracket \| \boldsymbol{\beta}\rangle\rangle \\
&\leftarrow_{\underline{\mu}} \mu\boldsymbol{\alpha}.\langle \underline{\mu}\boldsymbol{\beta}.\langle \llbracket v\rrbracket \| \tilde{\mu}\boldsymbol{x}.\langle \llbracket v'\rrbracket \| \boldsymbol{\beta}\rangle\rangle \| \boldsymbol{\beta}\rangle \\
&= \llbracket \mu\boldsymbol{\alpha}.[\boldsymbol{\beta}](\mathbf{let}\, \boldsymbol{x} = v \,\mathbf{in}\, v')\rrbracket
\end{aligned}$$

– $[\boldsymbol{\beta}](\mu\boldsymbol{\alpha}.C[\boldsymbol{\alpha}]) \to_{ad} C[\boldsymbol{\beta}]$:

$$\begin{aligned}
\llbracket [\boldsymbol{\beta}](\mu\boldsymbol{\alpha}.C[\boldsymbol{\alpha}])\rrbracket &= \langle \underline{\mu}\boldsymbol{\alpha}.\llbracket C\rrbracket[\boldsymbol{\alpha}] \| \boldsymbol{\beta}\rangle \\
&\to_{\underline{\mu}} \llbracket C\rrbracket[\boldsymbol{\beta}] && (\mathit{Compositional}) \\
&= \llbracket C[\boldsymbol{\beta}]\rrbracket && (\mathit{Lemma}\ 19)
\end{aligned}$$

– $F[\mu\boldsymbol{\alpha}.C[[\boldsymbol{\alpha}]v]] \to_{ad} \mu\boldsymbol{\beta}.C[[\boldsymbol{\beta}]F[v]]$ where $\alpha \notin FV(C)$:

$$\begin{aligned}
\llbracket F[\mu\alpha^{\mathbf{s}}.C[[\boldsymbol{\alpha}]v]]\rrbracket &\twoheadrightarrow_{\underline{\mu}} \underline{\mu}\boldsymbol{\beta}.\langle \mu\alpha^{\mathbf{s}}.\llbracket C\rrbracket[\langle \llbracket v\rrbracket \| \boldsymbol{\alpha}\rangle] \| \llbracket F\rrbracket^{\mathbf{s}}_{\boldsymbol{\beta}}\rangle && (\mathit{Lemma}\ 18) \\
&\to_{\underline{\mu}} \underline{\mu}\boldsymbol{\beta}.\llbracket C\rrbracket[\langle \llbracket v\rrbracket \| \llbracket F\rrbracket^{\mathbf{s}}_{\boldsymbol{\beta}}\rangle] && (\mathit{Lemma}\ 18) \\
&\twoheadleftarrow_{\underline{\mu}} \mu\boldsymbol{\beta}.\llbracket C\rrbracket[\langle \llbracket F[v]\rrbracket \| \boldsymbol{\beta}\rangle] && (\mathit{Lemma}\ 18) \\
&= \llbracket \mu\boldsymbol{\beta}.C[[\boldsymbol{\beta}]F]\rrbracket && (\mathit{Compositional})
\end{aligned}$$

– $\mu\boldsymbol{\alpha}.[\boldsymbol{\alpha}]v \to_{ad} v$:

$$\llbracket \mu\boldsymbol{\alpha}.[\boldsymbol{\alpha}]v\rrbracket = \mu\boldsymbol{\alpha}.\langle \llbracket v\rrbracket \| \boldsymbol{\alpha}\rangle \to_{\eta_\mu} \llbracket v\rrbracket$$

The cases for the serious $\lambda\mu$ reductions are:

– $(\lambda^{\mathbf{s}}x^{\mathbf{t}}.v_{\mathbf{r}}) \triangleleft_{\mathbf{r}} V_{\mathbf{t}} \to_{!} v\{V_{\mathbf{t}}/x^{\mathbf{t}}\}$:

$$\begin{aligned}
\llbracket (\lambda^{\mathbf{s}}x^{\mathbf{t}}.v_{\mathbf{r}}) \triangleleft_{\mathbf{r}} V_{\mathbf{t}}\rrbracket &= \underline{\mu}\alpha^{\mathbf{r}}.\langle \mu(x^{\mathbf{t}} \circledS \beta^{\mathbf{r}}).\langle \llbracket v_{\mathbf{r}}\rrbracket \| \beta^{\mathbf{r}}\rangle \| \llbracket V_{\mathbf{t}}\rrbracket \circledS \alpha^{\mathbf{r}}\rangle \\
&\to_{\beta\to} \underline{\mu}\alpha^{\mathbf{r}}.\langle \llbracket v_{\mathbf{r}}\rrbracket \| \alpha^{\mathbf{r}}\rangle && (\mathit{Lemma}\ 18) \\
&\to_{\eta_\mu} \llbracket v_{\mathbf{r}}\rrbracket
\end{aligned}$$

– $(\Lambda^{\mathbf{s}}a^{\mathbf{t}}.v_{\mathbf{r}}) \triangleleft_{\mathbf{r}} A_{\mathbf{t}} \to_{!} v\{A_{\mathbf{t}}/a^{\mathbf{t}}\}$: similar to the previous case.
– $v_{\mathbf{s}} \triangleleft_{\mathbf{r}} v_{\mathbf{t}} \to_{!} \mathbf{let}\, x^{\mathbf{s}} = v_{\mathbf{s}} \,\mathbf{in}\,\mathbf{let}\, y^{\mathbf{t}} = v_{\mathbf{t}} \,\mathbf{in}\, x^{\mathbf{s}} \triangleleft_{\mathbf{r}} y^{\mathbf{t}}$ where $v_{\mathbf{t}}$ is not a value:

$$\begin{aligned}
\llbracket v_{\mathbf{s}} \triangleleft_{\mathbf{r}} v_{\mathbf{t}}\rrbracket &= \underline{\mu}\alpha^{\mathbf{r}}.\langle \llbracket v_{\mathbf{s}}\rrbracket \| \llbracket v_{\mathbf{t}}\rrbracket \circledS \alpha^{\mathbf{r}}\rangle \\
&\to_{\varsigma\to} \underline{\mu}\alpha^{\mathbf{r}}.\langle \llbracket v_{\mathbf{s}}\rrbracket \| \tilde{\mu}x^{\mathbf{s}}.\langle \llbracket v_{\mathbf{t}}\rrbracket \| \tilde{\mu}y^{\mathbf{t}}.\langle x^{\mathbf{s}} \| y^{\mathbf{t}} \circledS \alpha^{\mathbf{r}}\rangle\rangle\rangle && (\mathit{Lemma}\ 18) \\
&\twoheadleftarrow_{\underline{\mu}} \llbracket \mathbf{let}\, x^{\mathbf{s}} = v_{\mathbf{s}} \,\mathbf{in}\,\mathbf{let}\, y^{\mathbf{t}} = v_{\mathbf{t}} \,\mathbf{in}\, x^{\mathbf{s}} \triangleleft_{\mathbf{r}} y^{\mathbf{t}}\rrbracket
\end{aligned}$$

– $\mathbf{let}\, \boldsymbol{x} = V \,\mathbf{in}\, v_{\mathbf{r}} \to_{!} v\{V/\boldsymbol{x}\}$:

$$\begin{aligned}
\llbracket \mathbf{let}\, \boldsymbol{x} = V \,\mathbf{in}\, v_{\mathbf{r}}\rrbracket &= \underline{\mu}\alpha^{\mathbf{r}}.\langle \llbracket V\rrbracket \| \tilde{\mu}\boldsymbol{x}.\langle \llbracket v_{\mathbf{r}}\rrbracket \| \alpha^{\mathbf{r}}\rangle\rangle \\
&\to_{\tilde{\mu}} \underline{\mu}\alpha^{\mathbf{r}}.\langle \llbracket v_{\mathbf{r}}\rrbracket\{\llbracket V\rrbracket/\boldsymbol{x}\} \| \alpha^{\mathbf{r}}\rangle && (\mathit{Lemma}\ 18) \\
&= \underline{\mu}\alpha^{\mathbf{r}}.\langle \llbracket v_{\mathbf{r}}\{V/x\}\rrbracket \| \alpha^{\mathbf{r}}\rangle && (\mathit{Lemma}\ 19) \\
&\to_{\eta_\mu} \llbracket v_{\mathbf{r}}\{V/x\}\rrbracket
\end{aligned}$$

– $[\boldsymbol{\beta}](\mu\boldsymbol{\alpha}.c) \to_{!} c\{\boldsymbol{\beta}/\boldsymbol{\alpha}\}$:

$$\begin{aligned} [\![[\boldsymbol{\beta}](\mu\boldsymbol{\alpha}.c)]\!] &= \langle \mu\boldsymbol{\alpha}.[\![c]\!] \| \boldsymbol{\beta} \rangle \\ &\to_{\mu} [\![c]\!]\{\boldsymbol{\beta}/\boldsymbol{\alpha}\} \\ &= [\![c\{\boldsymbol{\beta}/\boldsymbol{\alpha}\}]\!] && (\textit{Lemma } 19) \end{aligned}$$

– $F[\mu\boldsymbol{\alpha}.c] \to_{!} \mu\boldsymbol{\beta}.c\{[\boldsymbol{\beta}]F/[\boldsymbol{\alpha}]\Box\}$:

$$\begin{aligned} [\![F[\mu\alpha^{\mathbf{s}}.c]]\!] &\twoheadrightarrow_{\underline{\mu}} \underline{\mu}\boldsymbol{\beta}.\langle \mu\alpha^{\mathbf{s}}.[\![c]\!] \| [\![F]\!]^{\mathbf{s}}_{\boldsymbol{\beta}} \rangle && (\textit{Lemma } 18) \\ &\to_{\mu} \mu\boldsymbol{\beta}.[\![c]\!]\{[\![F]\!]^{\mathbf{s}}_{\boldsymbol{\beta}}/\alpha^{\mathbf{s}}\} && (\textit{Lemma } 18) \\ &\twoheadleftarrow_{\underline{\mu}} \mu\boldsymbol{\beta}.[\![c\{[\boldsymbol{\beta}]F/[\alpha^{\mathbf{s}}]\Box\}]\!] && (\textit{Lemma } 19) \\ &= [\![\mu\boldsymbol{\beta}.c\{[\boldsymbol{\beta}]F/[\alpha^{\mathbf{s}}]\Box\}]\!] \end{aligned}$$

Theorem 3. *Typed multi-discipline $\lambda\mu$ is strongly normalizing.*

Proof. An infinite typed $\lambda\mu$ reduction translates to an infinite typed sequent calculus reduction, which is impossible due to Corollary 1 and Lemma 17. In more detail, we can demonstrate the translation of an infinite reduction sequence by a simulation argument, where the simulation relation between $\lambda\mu$ terms and commands with sequent calculus terms and commands is

$$v \sim v' \iff [\![v]\!] =_{\underline{\mu}\eta_{\mu}} v' \qquad\qquad c \sim c' \iff [\![c]\!] =_{\underline{\mu}\eta_{\mu}} c'$$

To get started, note that $v \sim [\![v]\!]$ and $c \sim [\![c]\!]$ by reflexivity. Next, observe the simulation is transitive under administrative reduction and closed under serious reduction by Lemma 21: if $v_1 \sim v_1'$, then $v_1 \to_{ad} v_2$ in $\lambda\mu$ implies $v_1 \sim v_1'$, and also $v_1 \to_{!} v_2$ in $\lambda\mu$ implies that there is a v_2' in the sequent calculus such that $v_2 \twoheadrightarrow^{+} v_2'$ and $v_2 \sim v_2'$ due to Lemma 16 (and similarly for commands). Since administrative $\lambda\mu$ reduction is strongly normalizing (Lemma 20), any infinite reduction in typed $\lambda\mu$ must have an infinite number of serious reductions, and thus by the previous two facts, the translation of that term or command would also have an infinite number of serious reductions in the typed sequent calculus. Therefore, there cannot be an infinite reduction in typed $\lambda\mu$.

More constructively, we could think of $[\![-]\!]$ as computing a well-founded measure on serious sequent calculus reductions, which by Lemma 21 is decreasing with each serious reduction in $\lambda\mu$.