Thesis Proposal:
The logical basis of evaluation order

Noam Zeilberger
Carnegie Mellon University

May 25, 2007

Abstract

Most type systems are agnostic regarding the evaluation strategy for the underlying languages, with the value restriction for ML which is absent in Haskell as a notable exception. As type systems become more precise, however, detailed properties of the underlying operational semantics may become visible because properties captured by the types may be sound under one strategy but not the other. To give an example, intersection types distinguish between call-by-name and call-by-value functions because the subtyping rule $(A \to B) \cap (A \to C) \leq A \to (B \cap C)$ is valid for the former but not the latter in the presence of effects.

I propose to develop a unified, proof-theoretic approach to analyzing the interaction of types with evaluation order, based on the notion of *polarity*. Polarity was discovered and developed through linear logic, but I seek a fresh origin in Dummett's program of justifying the logical laws through alternative "meaning-theories," essentially hypotheses as to whether the *verification* or *use* of a proposition has a canonical form. In my preliminary work, I showed how a careful judgmental analysis of Dummett's ideas may be used to define a system of proofs and refutations, with a Curry-Howard interpretation as a *single* programming language in which the duality between call-by-value and call-by-name is realized as one of *types*. After extending its type system with (both positive and negative) union and intersection operators and a *derived* subtyping relationship, I found that many operationally-sensitive typing phenomena (e.g., alternative CBV/CBN subtyping distributivity principles, value and "covalue" restrictions) could be logically reconstructed. Here I give the technical details of this work, and present a plan for addressing open questions and extensions.

Thesis Committee: Frank Pfenning
Peter Lee
Robert Harper
Paul-André Melliès, Université Paris 7

1 Introduction

A famous essay by John Reynolds centers on the dangers of specifying the semantics of a language by means of a definitional interpreter, when the meaning of the defining language is itself potentially unclear [Rey72]. In a functional programming language, the result of function application can depend on whether evaluation order is call-by-value or call-by-name (due to the presence of non-termination and other side-effects), and Reynolds observes that a direct style interpreter leaves this choice only *implicit* in the evaluation order of the defining language—thus carrying little explanatory power, particularly in the case of a "meta-circular interpreter." He then goes on to give a careful account of how to make evaluation order *explicit* by writing the interpreter in continuation-passing style. Although the field of programming language semantics has given rise to many other styles of language definition, perhaps one of the morals to derive from "Definitional interpreters for higher-order programming languages" is the virtue of being fully explicit about evaluation order.

Yet a language definition includes not only the semantics of expressions, but also their syntax. Does the lesson of Reynolds' essay extend to the design of *type systems* as well? At a trivial level, it must. Implementations of popular functional languages such as ML (by default call-by-value) and Haskell (by default call-by-name, or rather its close relative call-by-*need*) include support, albeit limited, for mixing evaluation strategies—and obviously this support must have some syntactic basis, at the very least some string telling the compiler to employ one or the other strategy. On the other hand, this is exactly the sort of minimal answer the "Definitional interpreters" paper warns against, since it leaves the type distinction between call-by-value and call-by-name at the level of the "meta," i.e., wholly reliant upon a compiler's interpretation. Instead we should follow Reynolds' example and ask for something stronger: a type-theoretic *explication* of evaluation order.

This is a somewhat shadowy goal, so let us shed some light on the path by first examining the case of the archetypal functional language, the simply-typed lambda calculus, which is actually a bit *misleading.* The Curry-Howard correspondence observes that the simply-typed lambda calculus is just a reformulation of intuitionistic natural deduction. Since we know that the simply-typed lambda calculus admits soundly both call-by-name and call-by-value evaluation strategies, naively we are drawn to the conclusion that evaluation order must be a non-logical notion, outside the domain of the Curry-Howard correspondence. How could logic distinguish between call-by-value and call-by-name, when they correspond to the same system of logical deduction?

But at least one reason to doubt this pseudo-tautology is that the coincidence between type systems for languages with different evaluation strategies breaks down once these type systems become more expressive. Already this is the case with ML's type system. Although polymorphism—which corresponds logically to second-order quantification—is one of its distinctive and useful features, it has a troubled history: the original (standard) polymorphism introduction rule was unsound due to the presence of effects such as mutable storage

and callcc [HL91], and prompted various workarounds, the simplest being the *value restriction* [Wri95, MTHM97]. It might be easy to dismiss this as an anomaly peculiar to polymorphism and ML, but recent studies of intersection and union types in operational settings (in the context of so-called refinement type systems [PD01, DP04]) have produced many more examples of what could be called *operationally-sensitive typing phenomena*. The usual intersection introduction rule, for instance, is unsound in an effectful call-by-value language (requiring a value restriction), as is the standard subtyping distributivity law $(A \to B) \cap (A \to C) \leq A \to (B \cap C)$. These *are* sound under call-by-name evaluation (even in the presence of effects), but on the other hand there unions pose similar problems (e.g., $(A \to C) \cap (B \to C) \leq (A \cup B) \to C$ becomes unsound).

Hence type systems for languages with differing evaluation strategies really are distinguishable. But is there a systematic explanation for that difference? And does it have a logical basis in the Curry-Howard correspondence? The goal of my research is to answer this question—my thesis is that operationally-sensitive typing phenomena *can* be explained by the logical notion of *polarity*.

Polarity is an idea that comes out of linear logic, a classification of the connectives which opposes the *positive* $\otimes$ and $\oplus$ against the *negative* $\&$ and $\parr$. It was first discovered by Andreoli, applied to eliminate some of the crippling (but apparently needless) non-determinism that arises in doing proof search on the linear sequent calculus, through a restrictive yet still complete *focusing* strategy [And92]. However, my work seeks an origin for polarity that is not bound to linearity or the sequent calculus. In fact, I propose that polarity may be traced back to Michael Dummett's examination (in the 1976 William James Lectures [Dum91]) of the *justification* of the logical laws through alternative "meaning-theories," wherein the meanings of the connectives are fixed either by their introduction or their elimination rules. I believe that this idea is at the core of polarity, and can be elegantly formalized through the *judgmental method* [ML96, PD01]. The deductive systems constructed by this approach are interesting from a purely logical point-of-view—for example, they can compose intuitionistic and *co*-intuitionistic reasoning principles, and they have modular proofs of cut-elimination—but they also induce, through the Curry-Howard correspondence, programming languages in which evaluation order is fully explicit. Thus they are appropriate settings in which to study (and hopefully settle) questions about operationally-sensitive typing phenomena.

The next section begins with a very brief summary of Dummett's analysis of logical rules, providing the philosophical background for my account of polarity. A formalization of this analysis is then presented under the simplification that the only sort of higher-order reasoning is inference towards contradiction (corresponding to negation) rather than arbitrary inference (corresponding to implication). In Section 3, I present the Curry-Howard interpretation of this deductive system, a language that combines continuation-passing style with pattern-matching, and in which evaluation order is reflected at the level of types. Section 4 culminates my preliminary work, by extending this language with intersection and union types and a *derived* notion of subtyping that validates the above observations. Then in Section 5, I describe some of the open

problems and extensions I hope to address so as to solidify the argument of my thesis, and give a rough estimate of the time I expect to spend on them. Finally, in Section 6 I conclude with a review of related work.

2 The logic of proofs and refutations

2.1 Dummett's duelling definitions

In the 1976 William James Lectures [Dum91], Michael Dummett considered the possibility of *justifying* the logical laws. Rather than adopting the formalist position that the laws need no justification, or the holistic position that they can only be considered in toto, Dummett proposed that the logical laws can be considered individually and justified through an analysis of the meanings of the logical connectives. But not merely through an ordinary mathematical semantics—Dummett argued that such a meaning-theoretic analysis could be achieved through proof theory alone, that is, the meanings of the connectives could be fixed by (a subset of) the logical laws themselves.

Which laws determine the meanings of the connectives? Fundamentally, there are at least two rival interpretations. Quoting [Gen35, p. 80], Dummett motivates the first of these:

> Gerhard Gentzen, who, by inventing both natural deduction and the sequent calculus, first taught us how logic should be formalised, gave a hint how to do this [i.e., justify the logical laws], remarking without elaboration that "an introduction rule gives, so to say, a definition of the constant in question," by which he meant that it fixes its meaning, and that the elimination rule is, in the final analysis, no more than a consequence of this definition. [p. 251]

Seeing the introduction rules as fixing the meanings of the logical constants corresponds to what Dummett calls a "verificationist meaning-theory": the meaning of a logical constant is determined by the canonical ways of verifying the truth of a proposition with that constant as principal connective. The introduction rules are therefore self-justifying (they simply repeat the definition of a connective) while any other proposed logical rule must be validated by an "upwards justification procedure": a rule is valid if any *canonical proof* of its conclusion is already included in a *canonical proof* of its premises. The rationale behind this justification procedure is that if a proposition is true then it must have a canonical proof (what Dummett calls the "Fundamental Assumption")—and making this precise of course requires giving a formal definition of "canonical," which Dummett attempts to do. But for now, we will simply illustrate with examples. Recall the standard introduction rules for conjunction and disjunction:

$$\frac{A\ true \quad B\ true}{A \times B\ true} \qquad \frac{A\ true}{A + B\ true} \qquad \frac{B\ true}{A + B\ true}$$

These rules are now taken to define the connectives, asserting that *a canonical proof of* $A \times B$ *true is a canonical proof of* A *true together with a canonical*

proof of B true, while *a canonical proof of $A + B$ true is a canonical proof of A true or a canonical proof of B true.* Then the standard elimination rules for conjunction

$$\frac{A \times B \ true}{A \ true} \qquad \frac{A \times B \ true}{B \ true}$$

are justified, because a canonical proof of the conclusion A *true* (resp. B *true*) must already be part of a canonical proof of the premise $A \times B$ *true*. Diagrammatically, this argument may be represented as a pair of reductions on derivations:

$$\frac{\dfrac{\vdots}{A \ true} \quad \dfrac{\vdots}{B \ true}}{\dfrac{A \times B \ true}{A \ true}} \quad \Rightarrow \quad \begin{array}{c}\vdots\\ A \ true\end{array} \qquad\qquad \frac{\dfrac{\vdots}{A \ true} \quad \dfrac{\vdots}{B \ true}}{\dfrac{A \times B \ true}{B \ true}} \quad \Rightarrow \quad \begin{array}{c}\vdots\\ B \ true\end{array}$$

Moreover, this procedure can be applied to arbitrary logical inferences, not only the standard elimination rules. For example the "large-step" elimination rule

$$\frac{A \times (B_1 + B_2) \ true \qquad \begin{array}{c}[A \ true][B_1 \ true]\\ C \ true\end{array} \qquad \begin{array}{c}[A \ true][B_2 \ true]\\ C \ true\end{array}}{C \ true}$$

can be justified by the pair of transformations (for $i \in \{1, 2\}$) mapping

$$\frac{\dfrac{\vdots}{A \ true} \quad \dfrac{\begin{array}{c}\vdots\\ B_i \ true\end{array}}{B_1 + B_2 \ true}}{A \times (B_1 + B_2) \ true} \qquad \begin{array}{c}[A \ true][B_1 \ true]\\ \vdots\\ C \ true\end{array} \qquad \begin{array}{c}[A \ true][B_2 \ true]\\ \vdots\\ C \ true\end{array}}{C \ true}$$

to

$$\begin{array}{c}\dfrac{\vdots}{A \ true} \quad \dfrac{\vdots}{B_i \ true}\\ \vdots\\ C \ true\end{array}$$

Dummett stresses the significance of this generality. It means, for example, that while the verification of a proposition always takes a canonical form, its use (that is, the way one derives consequences from the proposition) need not be prescribed.

The verificationist meaning-theory and the upwards justification procedure are a way of vindicating Gentzen's conceptual prioritization of the logical rules. But Dummett argues that the notion that the introduction rules determine the meanings of the logical constants "has no more force than the converse

suggestion, that they are fixed by the elimination rules," going on to write, "The underlying idea is that the content of a statement is what you can *do* with it if you accept it.... This is, of course, the guiding idea of a pragmatist meaning-theory" [p. 280].

If the elimination rules are taken as primitive, then the other rules may now be justified by a dual, "downwards justification procedure": a rule is valid if any *canonically-obtained consequence* of its conclusion is already a *canonically-obtained consequence* of its premises. Again we forgo giving here the necessary definition of "canonically-obtained," but demonstrate with some examples. Taking the standard elimination rules above as now defining conjunction, we can justify the standard introduction rule by remarking that any use of $A \times B$ *true* must begin by projecting one of the two components and thus becomes a use of A *true* or B *true*—which are both premises of the introduction rule. Diagrammatically, this argument amounts to the following:

$$\dfrac{\dfrac{A\ \mathit{true} \quad B\ \mathit{true}}{A \times B\ \mathit{true}}}{A\ \mathit{true}} \quad \vdots \quad \Rightarrow \quad A\ \mathit{true} \quad \vdots \qquad \dfrac{\dfrac{A\ \mathit{true} \quad B\ \mathit{true}}{A \times B\ \mathit{true}}}{B\ \mathit{true}} \quad \vdots \quad \Rightarrow \quad B\ \mathit{true} \quad \vdots$$

And similarly, we can use this procedure to justify arbitrary logical inferences, such as the large-step introduction rules

$$\dfrac{A\ \mathit{true} \quad B_1\ \mathit{true}}{A \times (B_1 + B_2)\ \mathit{true}} \qquad \dfrac{A\ \mathit{true} \quad B_2\ \mathit{true}}{A \times (B_1 + B_2)\ \mathit{true}}$$

Here the intuitive argument for the validity of either rule is that any use of $A \times (B_1 + B_2)$ *true* is either a use of A *true* (a premise of the rule), or else a use of $B_1 + B_2$ *true*—but to draw a conclusion from $B_1 + B_2$ *true* one must be able to draw it from both B_1 *true* and B_2 *true* (one of these being a premise of the rule). Again, the generality of the downwards justification procedure means that while the uses of a proposition now always take a canonical form, its method of verification can be open-ended.

Can these two, equally legitimate ways of understanding the connectives be related? Dummett first considers the requirement of *harmony,* namely that a set of introduction and elimination rules can be validated under either interpretation, by both upwards and downwards justification procedures. However, as he puts it, harmony is an "excessively modest demand." The upwards justification procedure tells us only that the elimination rules are not too strong (given evidence for the premises we can construct evidence for the conclusion), while the downwards justification procedure tells us that the introduction rules are not too weak (any consequence drawn from the conclusion can be drawn from the premises). But are the elimination rules too weak, or the introduction rules too strong? Dummett proposes a test called *stability*—we omit his precise formulation here, but instead describe a simpler but similarly-motivated criterion taken from [PD01]. Suppose we start from a verificationist interpretation, and we wish to show that the elimination rules for a connective are not too weak,

i.e., that any consequence of a proposition with that principal connective can be obtained by first applying the elimination rules. Assuming the cut principle, this is equivalent to asserting that we can use the elimination rules to reconstruct the proposition. For example, we show that the elimination rules for conjunction are not too weak with the following transformation:[1]

$$\begin{array}{c} \vdots \\ A \times B \ \mathit{true} \end{array} \quad \Rightarrow \quad \dfrac{\dfrac{\begin{array}{c}\vdots\\ A \times B \ \mathit{true}\end{array}}{A \ \mathit{true}} \quad \dfrac{\begin{array}{c}\vdots\\ A \times B \ \mathit{true}\end{array}}{B \ \mathit{true}}}{A \times B \ \mathit{true}}$$

Of course, we can also start from a pragmatist interpretation and apply the dual transformation to check that the introduction rules are not too strong.

Now, Dummett observes (and considers it a virtue) that the standard natural deduction rules for intuitionistic logic may be validated by all of these procedures—unlike, for example, classical natural deduction (and in particular, the rules for negation). But is that really the last word? In the following sections, we will show how to formalize and *extend* Dummett's program.

Starting from the standard introduction rules, we will use the full force of a verificationist interpretation to derive a more powerful proof system for intuitionistic logic (in particular, with large-step elimination rules). Then by applying a pragmatist interpretation to the standard elimination rules, we will obtain a proof system that is actually *co*-intuitionistic. Finally, we will show how to combine these two systems, and relate them back to classical logic. Throughout this section, we will rely heavily on the judgmental method, as introduced by Martin-Löf and extensively developed by Pfenning and his collaborators [ML96, PD01]. Specifically, the formalism we develop will make essential use of categorical distinctions between the sort of reasoning involved in canonical proofs and in the inferences they justify, as well as between canonical consequences and the inferences *they* justify. At least in part this explains why we are able to go beyond Dummett's analysis.

2.2 Dual systems of proofs and refutations

We begin by examining the verificationist approach. The idea is to formalize the notion of canonical proof, concurrently with the notion of justified inference. *Concurrently,* because a canonical proof of an implication $A \to B$ is a justified inference from A to B. As a simplification, however, rather than considering arbitrary inferences, we will restrict our attention to inferences towards contradiction—that is, *refutations*—and thereby analyze only canonical proofs of negations $\neg A$ rather than of arbitrary implications (but this treatment *should* generalize: see Section 5).

Let us start by defining the judgment $A \ \mathit{true}$, with the meaning that there is a canonical proof of A. Repeating the definitions in the previous section, a

[1] In [PD01], this transformation is called a *local expansion,* used in combination with the upwards justification procedure, which is called a *local reduction.*

canonical proof of a conjunction is a pair of canonical proofs of the conjuncts, and a canonical proof of a disjunction is a proof of either disjunct:

$$\frac{A\ true \quad B\ true}{A \times B\ true} \qquad \frac{A\ true}{A + B\ true} \quad \frac{B\ true}{A + B\ true}$$

A canonical proof of a negation is a justified refutation, for which we use the judgment $A\ untrue$:

$$\frac{A\ untrue}{\neg A\ true}$$

But what is a justified refutation? Abstractly, it is an inference from the assumption that A is true towards a contradiction, justified on the grounds that if A is true then it must have a canonical proof (Dummett's Fundamental Assumption). In other words, a justified refutation of A consists of a method for turning any canonical proof of A into a contradiction.

Now, each of the above sets of introduction rules for canonical proofs comes with a corresponding inversion principle. For example, given a canonical proof of $A + B$, we can recover either a canonical proof of A or a canonical proof of B. A perspicuous way of keeping track of these inversion principles is by means of a relation $\Phi \Rightarrow A\ true$, between the judgment $A\ true$ and a list of judgments Φ containing only refutations $B\ untrue$ and atomic truths $X\ true$. We can view Φ as a minimal list of components needed to construct a canonical proof of A, and the set $(A\ true)^{-1} = \{\Phi \,|\, \Phi \Rightarrow A\ true\}$ as providing a decomposition of the hypothesis $A\ true$. Thus for example, we have $(X + (Y \times Z)\ true)^{-1} = \{(X\ true), (Y\ true, Z\ true)\}$, since a canonical proof of $X + (Y \times Z)$ is either a canonical proof of X (which cannot be further decomposed), or a canonical proof of Y together with a canonical proof of Z. We have $(\neg A \times \neg B\ true)^{-1} = \{(A\ untrue, B\ untrue)\}$, since a canonical proof of $\neg A \times \neg B$ is a refutation of A together with a refutation of B. By unpacking the above introduction rules, we can give the following axiomatization of $\Phi \Rightarrow A\ true$:

$$\frac{\Phi_1 \Rightarrow A\ true \quad \Phi_2 \Rightarrow B\ true}{\Phi_1, \Phi_2 \Rightarrow A \times B\ true} \qquad \frac{\Phi \Rightarrow A\ true}{\Phi \Rightarrow A + B\ true} \quad \frac{\Phi \Rightarrow B\ true}{\Phi \Rightarrow A + B\ true}$$

$$\frac{}{X\ true \Rightarrow X\ true} \qquad \frac{}{A\ untrue \Rightarrow \neg A\ true}$$

where Φ_1, Φ_2 is the usual concatenation of contexts.

We can now state the procedure for justified refutation. To refute A, we must be able to derive a contradiction from the hypotheses Φ, for any Φ such that $\Phi \Rightarrow A\ true$. And to derive a contradiction from Φ (written $\Phi \vdash contra$), we must find some hypothesis $B\ untrue \in \Phi$, and prove that B is true. Since the proof of $B\ true$ may itself require hypotheses in Φ (e.g., consider how one refutes $X \times \neg X$), we actually need to generalize to hypothetical judgments $\Phi \vdash A\ true$ and $\Phi \vdash A\ untrue$. This is done in Figure 1, which summarizes the foregoing verificationist analysis. Again, the three hypothetical judgments may be glossed as follows:

Formulas $A, B ::= X \mid 1 \mid A \times B \mid 0 \mid A + B \mid \neg A$
Contexts $\Phi ::= \cdot \mid \Phi, X\ true \mid \Phi, A\ untrue$

$\boxed{\Phi \vdash A\ true}$

$$\frac{X\ true \in \Phi}{\Phi \vdash X\ true} \gg X \qquad \frac{\Phi \vdash A\ untrue}{\Phi \vdash \neg A\ true} \gg \neg$$

$$\frac{}{\Phi \vdash 1\ true} \gg 1 \qquad \frac{\Phi \vdash A\ true \quad \Phi \vdash B\ true}{\Phi \vdash A \times B\ true} \gg \times$$

(no rule for 0) $\qquad \dfrac{\Phi \vdash A\ true}{\Phi \vdash A + B\ true} \quad \dfrac{\Phi \vdash B\ true}{\Phi \vdash A + B\ true} \gg +$

$\boxed{\Phi \vdash A\ untrue}$

$$\frac{\forall \Phi' \in (A\ true)^{-1} \quad \Phi, \Phi' \vdash contra}{\Phi \vdash A\ untrue}\ blur\ll$$

$\boxed{\Phi \vdash contra}$

$$\frac{\Phi \vdash A\ true \quad A\ untrue \in \Phi}{\Phi \vdash contra}\ focus\gg$$

. . . . . .

$\boxed{\Phi \Rightarrow A\ true}$

$$\frac{}{X\ true \Rightarrow X\ true} \qquad \frac{}{A\ untrue \Rightarrow \neg A\ true}$$

$$\frac{}{\cdot \Rightarrow 1\ true} \qquad \frac{\Phi_1 \Rightarrow A\ true \quad \Phi_2 \Rightarrow B\ true}{\Phi_1, \Phi_2 \Rightarrow A \times B\ true}$$

(no rule for 0) $\qquad \dfrac{\Phi \Rightarrow A\ true}{\Phi \Rightarrow A + B\ true} \quad \dfrac{\Phi \Rightarrow B\ true}{\Phi \Rightarrow A + B\ true}$

$$(A\ true)^{-1} = \{\Phi \mid \Phi \Rightarrow A\ true\}$$

Figure 1: Canonical proofs and justified refutations

$\Phi \vdash A\ \mathit{true}$	A has a canonical proof (assuming Φ)
$\Phi \vdash A\ \mathit{untrue}$	A has a justified refutation (assuming Φ)
$\Phi \vdash \mathit{contra}$	Φ is contradictory

It is also worth observing that in terms of provability, the system of deduction defined here is intuitionistic.

There is clearly a close connection between $\Phi \vdash A\ \mathit{true}$, which asserts that A has a canonical proof, and $\Phi \Rightarrow A\ \mathit{true}$, which merely "inverts" that judgment. Indeed, this connection is what makes our method of refutation *justified.* Formally, we can describe the connection with the following converse properties:

Property (Inversion-reduction). *If* $\Phi \vdash A\ \mathit{true}$ *then* $\exists \Phi' \in (A\ \mathit{true})^{-1}$ *such that* $\Phi \vdash \Phi'$ *(meaning* $\Phi \vdash J$ *for all* $J \in \Phi'$*).*

Property (Inversion-expansion). *If* $\Phi' \in (A\ \mathit{true})^{-1}$ *and* $\Phi \vdash \Phi'$ *then* $\Phi \vdash A\ \mathit{true}$*.*

It is trivial to check both properties by induction on A. Moreover, we contend that these are exactly the properties needed to justify the rule for refutation: inversion-reduction guarantees that it is not too strong, while inversion-expansion that it is not too weak. We demonstrate this by proving that the system defined by Figure 1 satisfies "identity" and "cut" principles. We observe, first, that $\Phi \Rightarrow A\ \mathit{true}$ satisfies a subformula property: the hypotheses in Φ mention only subformulas of A. Moreover, we note that weakening is admissible for each of the hypothetical judgments $\Phi \vdash A\ \mathit{true}$, $\Phi \vdash A\ \mathit{untrue}$, and $\Phi \vdash \mathit{contra}$.

Principle (Identity). *If* $A\ \mathit{untrue} \in \Phi$ *then* $\Phi \vdash A\ \mathit{untrue}$*.*

Principle (Cut). *If* $\Phi \vdash A\ \mathit{true}$ *and* $\Phi \vdash A\ \mathit{untrue}$ *then* $\Phi \vdash \mathit{contra}$*.*

Proof (of identity). We reduce this to the identity principle on subformulas of A, proving it simultaneously with a "context identity" principle:

Principle (Context identity). *For all* Φ*,* $\Phi \vdash \Phi$*.*

Now assume $A\ \mathit{untrue} \in \Phi$. The derivation of $\Phi \vdash A\ \mathit{untrue}$ begins as follows:[2]

$$\dfrac{\forall \Phi' \in (A\ \mathit{true})^{-1} \qquad \dfrac{\Phi, \Phi' \vdash A\ \mathit{true} \quad A\ \mathit{untrue} \in \Phi}{\Phi, \Phi' \vdash \mathit{contra}}\ \mathit{focus}\gg}{\Phi \vdash A\ \mathit{untrue}}\ \mathit{blur}\ll$$

So that we must show $\Phi, \Phi' \vdash A\ \mathit{true}$ for all $\Phi' \in (A\ \mathit{true})^{-1}$. But $\Phi' \vdash \Phi'$ holds by the context identity principle, which implies $\Phi' \vdash A\ \mathit{true}$ by inversion-expansion, and finally $\Phi, \Phi' \vdash A\ \mathit{true}$ by admissible weakening.

By definition, context identity $\Phi \vdash \Phi$ reduces to the identity principle on hypotheses $B\ \mathit{untrue}$ in Φ (it is trivial for atomic truth hypotheses). Since the $\Phi' \in (A\ \mathit{true})^{-1}$ used above contain only subformulas of A, the induction is well-founded. □

[2]The funny names labelling the rules will eventually be explained.

Proof (of cut). Likewise we reduce to the cut principle on subformulas of A, proving cut simultaneously with a substitution principle:

Principle (Substitution). *If* $\Phi \vdash \Phi'$ *and* $\Phi, \Phi' \vdash J$ *then* $\Phi \vdash J$

Now assume $\Phi \vdash A\ \mathit{true}$ and $\Phi \vdash A\ \mathit{untrue}$. By applying inversion-reduction on the first premise we find a $\Phi' \in (A\ \mathit{true})^{-1}$ such that $\Phi \vdash \Phi'$. Since the second premise was derived using $\mathit{blur}\ll$, we have $\Phi, \Phi' \vdash \mathit{contra}$, and $\Phi \vdash \mathit{contra}$ follows by substitution.

The proof of substitution uses a secondary induction on the derivation of $\Phi, \Phi' \vdash J$, but appeals back to the cut principle in the following case:

$$\frac{\Phi, \Phi' \vdash B\ \mathit{true} \quad B\ \mathit{untrue} \in \Phi'}{\Phi, \Phi' \vdash \mathit{contra}}\ \mathit{focus}\gg$$

Here $\Phi \vdash B\ \mathit{true}$ by the i.h., and $\Phi \vdash B\ \mathit{untrue}$ by the hypothesis that $\Phi \vdash \Phi'$; hence we obtain $\Phi \vdash \mathit{contra}$ by the cut principle. Again the induction is well-founded because above we applied substitution with $\Phi' \in (A\ \mathit{true})^{-1}$, which contains only subformulas of A. □

(We should remark, incidentally, that while admissibility of identity and cut are standard theorems for deductive systems (usually for sequent calculi), *both* of the proofs exhibited above are entirely modular, in the sense that interactions between different connectives need not be considered. In particular, the proof of cut admissibility avoids the usual quadratic explosion of cases, relying only upon the inversion-reduction property.)

With these basic facts established, let us now see how to obtain a "pragmatist" interpretation, fixing the meanings of the logical constants by their elimination rules and formalizing the notion of "canonically-obtained consequence." Again, rather than considering arbitrary consequences, we will confine ourselves to a distinguished one—and the elimination rules thereby become introduction rules for *canonical refutations.*

Let us illustrate first with disjunction. The usual elimination rule states that to show some conclusion from a disjunction, one must show it from both disjuncts. As a special case, then, to *refute* a proposition $A + B$ one must refute both A and B:

$$\frac{A\ \mathit{false} \quad B\ \mathit{false}}{A + B\ \mathit{false}}$$

By the same line of reasoning, the two elimination rules for conjunction become two falsehood-introduction rules:

$$\frac{A\ \mathit{false}}{A \times B\ \mathit{false}} \qquad \frac{B\ \mathit{false}}{A \times B\ \mathit{false}}$$

These rules define conjunction by the slogan, "A refutation of $A \times B$ is a refutation of A or a refutation of B." Now, while both *rules* are certainly sound if we interpret falsehood as intuitionistic falsehood (i.e., as *untrue*), this "co-disjunction property" obtained as an inversion principle is *not* intuitionistically

Formulas $A, B ::= X \mid 1 \mid A \times B \mid 0 \mid A + B \mid \neg A$
Contexts $\Phi ::= \cdot \mid \Phi, X\ \mathit{false} \mid \Phi, A\ \mathit{unfalse}$

$\boxed{\Phi \vdash A\ \mathit{false}}$

$$\frac{X\ \mathit{false} \in \Phi}{\Phi \vdash X\ \mathit{false}}\ X\ll \qquad \frac{\Phi \vdash A\ \mathit{unfalse}}{\Phi \vdash \neg A\ \mathit{false}}\ \neg\ll$$

$$\text{(no rule for 1)} \qquad \frac{\Phi \vdash A\ \mathit{false}}{\Phi \vdash A \times B\ \mathit{false}}\ \frac{\Phi \vdash B\ \mathit{false}}{\Phi \vdash A \times B\ \mathit{false}}\ \times\ll$$

$$\frac{}{\Phi \vdash 0\ \mathit{false}}\ 0\ll \qquad \frac{\Phi \vdash A\ \mathit{false} \quad \Phi \vdash B\ \mathit{false}}{\Phi \vdash A + B\ \mathit{false}}\ +\ll$$

$\boxed{\Phi \vdash A\ \mathit{unfalse}}$

$$\frac{\forall \Phi' \in (A\ \mathit{false})^{-1} \quad \Phi, \Phi' \vdash \mathit{contra}}{\Phi \vdash A\ \mathit{unfalse}} \gg \mathit{blur}$$

$\boxed{\Phi \vdash \mathit{contra}}$

$$\frac{A\ \mathit{unfalse} \in \Phi \quad \Phi \vdash A\ \mathit{false}}{\Phi \vdash \mathit{contra}} \ll \mathit{focus}$$

. . . . . .

$\boxed{\Phi \Rightarrow A\ \mathit{false}}$

$$\frac{}{X\ \mathit{false} \Rightarrow X\ \mathit{false}} \qquad \frac{}{A\ \mathit{unfalse} \Rightarrow \neg A\ \mathit{false}}$$

$$\text{(no rule for 1)} \qquad \frac{\Phi \Rightarrow A\ \mathit{false}}{\Phi \Rightarrow A \times B\ \mathit{false}}\ \frac{\Phi \Rightarrow B\ \mathit{false}}{\Phi \Rightarrow A \times B\ \mathit{false}}$$

$$\frac{}{\cdot \Rightarrow 0\ \mathit{false}} \qquad \frac{\Phi_1 \Rightarrow A\ \mathit{false} \quad \Phi_2 \Rightarrow B\ \mathit{false}}{\Phi_1, \Phi_2 \Rightarrow A + B\ \mathit{false}}$$

$$(A\ \mathit{false})^{-1} = \{\Phi \mid \Phi \Rightarrow A\ \mathit{false}\}$$

Figure 2: Canonical refutations and justified proofs-by-contradiction

legitimate: consider the law of contradiction $A \times \neg A$ *untrue*. Instead what we have here is the stronger notion of "constructible falsity" [Nel49], which in turn may be *deconstructed* to justify a co-intuitionistic notion of truth: a proposition is co-intuitionistically true just in case its falsehood entails a contradiction.

Figure 2 presents a system of canonical refutations and justified proofs-by-contradiction, again by making use of three judgments:

$\Phi \vdash A\ \mathit{false}$	A has a canonical refutation (assuming Φ)
$\Phi \vdash A\ \mathit{unfalse}$	A has a justified proof-by-contradiction (assuming Φ)
$\Phi \vdash \mathit{contra}$	Φ is contradictory

If we define the dualization operator $(-)^\circ$ on formulas by

$$X^\circ = X \qquad 1^\circ = 0 \qquad 0^\circ = 1$$

$$(A \times B)^\circ = A^\circ + B^\circ \quad (A + B)^\circ = A^\circ \times B^\circ \quad (\neg A)^\circ = \neg A^\circ$$

and extend it to judgments and contexts with

$$(A\ \mathit{true})^\circ = A^\circ\ \mathit{false} \quad \mathit{contra}^\circ = \mathit{contra} \quad (A\ \mathit{untrue})^\circ = A^\circ\ \mathit{unfalse}$$
$$(A\ \mathit{false})^\circ = A^\circ\ \mathit{true} \quad (A\ \mathit{unfalse})^\circ = A^\circ\ \mathit{untrue}$$
$$(\cdot)^\circ = (\cdot) \qquad (J, \Phi)^\circ = (J^\circ, \Phi^\circ)$$

then the following observation is obvious:

Observation (Principle of Duality). $\Phi \vdash J$ *iff* $\Phi^\circ \vdash J^\circ$

In particular, the pragmatist system must admit identity and cut principles.

Principle (Identity). *If* $A\ \mathit{unfalse} \in \Phi$ *then* $\Phi \vdash A\ \mathit{unfalse}$.

Principle (Cut). *If* $\Phi \vdash A\ \mathit{unfalse}$ *and* $\Phi \vdash A\ \mathit{false}$ *then* $\Phi \vdash \mathit{contra}$.

2.3 The unity of duality

Given the formal duality between verificationist and pragmatist interpretations, it seems that a choice of one over the other has, to use Dummett's phrase, "no more force than the converse suggestion." On the other hand, the two interpretations are *different—A true* cannot be equated with *A unfalse*, nor *A untrue* with *A false*.[3] Some notable counterexamples:

$$\vdash A + \neg A\ \mathit{unfalse} \quad \text{but} \quad \nvdash A + \neg A\ \mathit{true}$$
$$\vdash A \times \neg A\ \mathit{untrue} \quad \text{but} \quad \nvdash A \times \neg A\ \mathit{false}$$

But to say that the two interpretations of the logical constants are different is another way of saying that they define different logical constants. In other words, much like linear logic recognizes two conjunctions $\otimes$ and $\&$ with very different properties, we should distinguish "verificationist conjunction" from "pragmatist conjunction," and so forth with the other connectives.

Indeed, so long as we realize that this is not suddenly imposing linearity constraints, it is convenient to distinguish them using the *notation* of linear logic. Thus we write $\otimes$ and $\oplus$ for verificationist conjunction and disjunction, $\&$ and $\parr$ for pragmatist. With a bit of foreshadowing, we write $\overset{v}{\neg}$ and $\overset{n}{\neg}$ for the respective forms of negation. The verificationist and pragmatist approaches correspond, respectively, to *positive* and *negative polarity*. As per the technique of polarized linear logic, we can define two independent classes of formulas:

$$\begin{array}{lll} P, Q & ::= & X \mid 1 \mid P \otimes Q \mid 0 \mid P \oplus Q \mid \overset{v}{\neg} P \\ M, N & ::= & \overline{X} \mid \top \mid M \& N \mid \bot \mid M \parr N \mid \overset{n}{\neg} M \end{array}$$

[3] Or to put it another way, taking this *unjustified* step would make the system collapse to classical logic.

Figures 1 and 2 should now be suitably reinterpreted with, respectively, Ps and Qs, or Ms and Ns, in place of As and Bs. The reader may keep in mind the following mnemonic: the verificationist connectives are defined positively, i.e., in terms of truth, while the pragmatist connectives are defined negatively, i.e., in terms of falsehood.

This syntactic separation makes it easy to see that there is no harm in *combining* the two interpretations, i.e., allowing contexts to contain a mix of hypotheses X *true*, $\overline{X}$ *false*, P *untrue*, N *unfalse*, and considering the union of the rules in Figures 1 and 2. However, it is also not clear that this gains us anything—indeed, the syntactic separation ensures that there are no possible interactions between the two sets of rules! But to break this impasse we can (following [Gir01]) add a pair of mediating connectives:

$$\begin{array}{lll} P, Q & ::= & \cdots \mid \downarrow N \\ M, N & ::= & \cdots \mid \uparrow P \end{array}$$

With these operators (Girard's "shift" operators), we may now interpret (the more permissive) co-intuitionistic truth as a *modality* of intuitionistic truth

$$\frac{\Phi \vdash N \ \mathit{unfalse}}{\Phi \vdash \downarrow N \ \mathit{true}} \gg\downarrow \qquad N \ \mathit{unfalse} \Rightarrow \downarrow N \ \mathit{true}$$

and intuitionistic falsehood as a modality of co-intuitionistic falsehood

$$\frac{\Phi \vdash P \ \mathit{untrue}}{\Phi \vdash \uparrow P \ \mathit{false}} \uparrow\ll \qquad P \ \mathit{untrue} \Rightarrow \uparrow P \ \mathit{false}$$

The two fragments now can interact in interesting ways. For example, we can show (with the initial premise by the identity principle)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P \ \mathit{untrue}, {}^{v}\neg P \mathit{untrue} \vdash P \ \mathit{untrue}}{P \ \mathit{untrue}, {}^{v}\neg P \ \mathit{untrue} \vdash {}^{v}\neg P \ \mathit{true}} \gg{}^{v}\neg}{P \ \mathit{untrue}, {}^{v}\neg P \ \mathit{untrue} \vdash \mathit{contra}} \mathit{focus}\gg}{\vdash \uparrow P \otimes \uparrow {}^{v}\neg P \ \mathit{unfalse}} \gg \mathit{blur}}{\vdash \downarrow(\uparrow P \otimes \uparrow {}^{v}\neg P) \ \mathit{true}} \gg\downarrow$$

By a more complicated derivation, we can show $\downarrow\uparrow(P \oplus {}^{v}\neg P)$ *true*. In fact, one can show $\downarrow\uparrow P$ *true* whenever P *untrue* $\vdash$ *contra*, and $\uparrow\downarrow N$ *false* whenever N *unfalse* $\vdash$ *contra*, so that the composition of the shift operators is essentially double-negation.[4] The reader is invited to play with the shift operators and try to gain some intuition for their logical content—but their meaning will become

[4]Note that we can now also define another, *involutive* negation $(-)^{\perp}$, such that $N^{\perp}$ *true* iff N *false* and $P^{\perp}$ *false* iff P *true*, essentially internalizing the dualization operator. Then ${}^{v}\neg P$ could be decomposed as $\downarrow(P^{\perp})$ (as is done in [Gir01, §9.2]) and ${}^{n}\neg N$ as $\uparrow(N^{\perp})$. While $(-)^{\perp}$ is interesting as a tool of analysis, its operational value is at present unclear to me, and it will be left out of the language defined in the sequel. However, see the discussion in Sections 5 and 6.

much more mundane once we develop the Curry-Howard interpretation. There, we will see that they indicate the presence of control effects: $\downarrow N$ is the type of a *suspended* expression of type N, while $\uparrow P$ the type of a *captured* continuation of type P.

We hold off on defining the programming language for just a while longer, in order to explain the relationship of this unified logic of intuitionistic and co-intuitionistic reasoning to Andreoli's notion of "focusing" strategies for sequent calculi.

2.4 A focused look at classical sequent calculus

Consider the following transformation taking a context of hypotheses Φ to a pair of multisets of formulas $(\Gamma_\Phi; \Delta_\Phi)$:

- If X *true* $\in \Phi$ then $X \in \Gamma_\Phi$; if N *unfalse* $\in \Phi$ then $N \in \Gamma_\Phi$
- If $\overline{X}$ *false* $\in \Phi$ then $\overline{X} \in \Delta_\Phi$; if P *untrue* $\in \Phi$ then $P \in \Delta_\Phi$

For example, corresponding to $\Phi = (X\ \mathit{true}, X \oplus {}^{v}\neg X\ \mathit{untrue})$ we have $\Gamma_\Phi = (X)$ and $\Delta_\Phi = (X \oplus {}^{v}\neg X)$. Note that we retain the syntactic distinction between positive and negative formulas. Now, we locate five kinds of sequents—four styles of *focused* sequents (which have a single polarized formula "in focus" on either the left or right) as well as ordinary, *unfocused* sequents:

$$\Gamma \to \Delta \gg P \qquad P \ll \Gamma \to \Delta$$
$$\Gamma \to \Delta \gg N \qquad N \ll \Gamma \to \Delta$$
$$\Gamma \to \Delta$$

where Γ contains (arbitrary) negative formulas and (only) atomic positive formulas, while Δ contains (arbitrary) positive formulas and (only) atomic negative formulas. Each of our hypothetical judgments transforms (bijectively) into one of these five sequents:

$$\begin{array}{rcl}
\Phi \vdash P\ \mathit{true} & \iff & \Gamma_\Phi \to \Delta_\Phi \gg P \\
\Phi \vdash P\ \mathit{untrue} & \iff & P \ll \Gamma_\Phi \to \Delta_\Phi \\
\Phi \vdash N\ \mathit{unfalse} & \iff & \Gamma_\Phi \to \Delta_\Phi \gg N \\
\Phi \vdash N\ \mathit{false} & \iff & N \ll \Gamma_\Phi \to \Delta_\Phi \\
\Phi \vdash \mathit{contra} & \iff & \Gamma_\Phi \to \Delta_\Phi
\end{array}$$

Now, if we look at Figures 1 and 2 through the lens of this transformation, we see the reason behind the rule names. For example the truth-introduction rules $\gg op$ introduce the connective op in right-focus, as in

$$\frac{\Gamma \to \Delta \gg P \quad \Gamma \to \Delta \gg Q}{\Gamma \to \Delta \gg P \otimes Q} \gg\otimes \qquad \frac{P \ll \Gamma \to \Delta}{\Gamma \to \Delta \gg {}^{v}\neg P} \gg{}^{v}\neg$$

whereas the falsehood-introduction rules $op \ll$ introduce op in left-focus. The *blur* rules, read as proof-search directives from bottom to top, "blur" a focused goal sequent into a set of unfocused goals, while the *focus* rules focus on a particular formula in an unfocused goal:

$$\frac{\forall(\Gamma',\Delta') \in (P)^{-1} \quad \Gamma',\Gamma \to \Delta,\Delta'}{P \ll \Gamma \to \Delta}\ \mathit{blur}{\ll}$$

$$\frac{\forall(\Gamma',\Delta') \in (N)^{-1} \quad \Gamma',\Gamma \to \Delta,\Delta'}{\Gamma \to \Delta \gg N}\ {\gg}\mathit{blur}$$

$$\frac{\Gamma \to \Delta, P \gg P}{\Gamma \to \Delta, P}\ \mathit{focus}{\gg} \qquad \frac{N \ll N, \Gamma \to \Delta}{N, \Gamma \to \Delta}\ {\ll}\mathit{focus}$$

The identity and cut principles become:

Principle (Identity). *(pos.)* $P \ll \Gamma \to \Delta, P$*; (neg.)* $N, \Gamma \to \Delta \gg N$

Principle (Cut). *(pos.) if* $\Gamma \to \Delta \gg P$ *and* $P \ll \Gamma \to \Delta$ *then* $\Gamma \to \Delta$*; (neg.) if* $\Gamma \to \Delta \gg N$ *and* $N \ll \Gamma \to \Delta$ *then* $\Gamma \to \Delta$

Consider as an example the derivation of $X \oplus {}^{v}\neg X$ *untrue* $\vdash$ *contra*. Via this notational transformation the derivation becomes the following proof of $\cdot \to X \oplus {}^{v}\neg X$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{X \to X \oplus {}^{v}\neg X \gg X}\ {\gg}X}{X \to X \oplus {}^{v}\neg X \gg X \oplus {}^{v}\neg X}\ {\gg}\oplus}{X \to X \oplus {}^{v}\neg X}\ \mathit{focus}{\gg}}{X \ll \cdot \to X \oplus {}^{v}\neg X}\ \mathit{blur}{\ll}}{\cdot \to X \oplus {}^{v}\neg X \gg {}^{v}\neg X}\ {\gg}{}^{v}\neg}{\cdot \to X \oplus {}^{v}\neg X \gg X \oplus {}^{v}\neg X}\ {\gg}\oplus}{\cdot \to X \oplus {}^{v}\neg X}\ \mathit{focus}{\gg}$$

And now if we "forget" about polarity (i.e., replace $\oplus$ and ${}^{v}\neg$ by $+$ and $\neg$) and erase the symbols "$\gg$" and "$\ll$":

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{X \to X + \neg X, X}}{X \to X + \neg X, X + \neg X}}{X \to X + \neg X}}{X \to X + \neg X}}{\cdot \to X + \neg X, \neg X}}{\cdot \to X + \neg X, X + \neg X}}{\cdot \to X + \neg X}$$

The derivation has become more or less an ordinary classical sequent calculus proof, where truth-introduction rules have been replaced by ordinary right rules, *focus*$\gg$ by right-contraction, and *blur*$\ll$ by a sequence (in this case empty) of left rules. If we continue carrying out this transformation, we will find that falsehood-introduction rules map to sequent calculus left rules, $\ll$*focus* to left-contraction, $\gg$*blur* to a sequence of right rules. The punchline here is that

while we derived Figures 1 and 2 judgmentally as refinements of intuitionistic and co-intuitionistic natural deduction, based on dual verificationist/pragmatist readings of the connectives, it is also possible to view them as a *focusing* strategy for the classical sequent calculus.

For a full background on focusing, the reader is referred to Andreoli's tutorial [And01]. The technique was originally invented as a way of guiding proof search in linear logic [And92], based on the observation that the order of application of rules need not be entirely unconstrained, but instead can be made to alternate between very narrow focused and unfocused stages. Starting from an unfocused sequent, the first action must be to choose some formula to focus on. Classifying the linear connectives $\otimes, \oplus, 1, 0$ as positive, $\&, \parr, \top, \bot$ as negative,[5] the chosen formula is either one on the right side of the sequent with positive outermost connective, or one on the left side with negative outermost connective.[6] Proof search must continue by analyzing the focused formula, up to the point of a polarity mismatch (i.e., a positive connective in left-focus, or a negative connective in right-focus). At that point the formula is decomposed in one step by applying a sequence of invertible rules, converting the goal into a set of unfocused sequents.

From the point of view of proof search, the crucial fact is that this strategy is complete: a linear logic sequent has an ordinary proof just in case it has a focusing proof. Since there are far fewer focusing proofs, the efficiency of proof search is greatly improved. Yet, our rational reconstruction of focusing argues that it is much more than optimization.

For classical logic, this effect is particularly striking. Whereas the polarity of each linear connective (save negation) is forced by its resource-awareness, the classical connectives are fundamentally ambivalent. As demonstrated by different authors, many of the degeneracies of classical logic—non-confluence, lack of a non-trivial theory of type isomorphisms—may be traced back to this ambivalence [Gir91, DJS97, Lau05]. On the other hand, from the standpoint of classical provability, nothing is lost by the focusing discipline, in the sense that we can prove (see Section 3.4) a completeness theorem:

$$|\Gamma| \to^c |\Delta| \text{ implies } \Gamma \to \Delta$$

where $\to^c$ stands for classical provability, and $|-|$ is the operator which forgets polarity. Moreover, it should be intuitively clear from the preceding judgmental analysis that focusing completeness is essentially a refined statement of the completeness of alternative double-negation translations, and accordingly we can expect that the focusing proof has a better-defined, more *explicit* computational content than the classical sequent calculus proof. In fact, we will find that focusing proofs are even more explicit than ordinary proofs of intuitionistic

[5] Andreoli uses the classification "synchronous"/"asynchronous," making the dichotomy "verificationist ≈ positive ≈ synchronous ≈ call-by-value" vs. "pragmatist ≈ negative ≈ asynchronous ≈ call-by-name". This surfeit of names for the same thing is certainly confusing, but on the other hand perhaps encouraging.

[6] The innovation of keeping positive and negative *formulas* syntactically distinct but composable via the shift operators was introduced in [Gir01].

$$
\begin{array}{lll}
\text{Positive types} & P, Q & ::= X \mid 1 \mid P \otimes Q \mid 0 \mid P \oplus Q \mid {}^{v}\neg P \mid \downarrow N \\
\text{Negative types} & M, N & ::= \overline{X} \mid \top \mid M \,\&\, N \mid \bot \mid M \parr N \mid {}^{n}\neg M \mid \uparrow P
\end{array}
$$

$$
\text{Contexts} \qquad \Phi \quad ::= \cdot \mid \Phi, x \overset{\text{val}}{:} X \mid \Phi, \overline{u} \overset{\text{cnt}}{:} P \mid \Phi, u \overset{\text{exp}}{:} N \mid \Phi, \overline{x} \overset{\text{cov}}{:} \overline{X}
$$

Judgments

$\Phi \vdash V \overset{\text{val}}{:} P$	value V has type P
$\Phi \vdash K \overset{\text{cnt}}{:} P$	continuation K has type P
$\Phi \vdash E \overset{\text{exp}}{:} N$	expression E has type N
$\Phi \vdash C \overset{\text{cov}}{:} N$	covalue C has type N
$\Phi \vdash S \overset{\text{stm}}{:} \cdot$	statement S is well-typed

Figure 3: **CU** types, contexts and typing judgments

logic, in the sense that *evaluation order* is completely determined by the polarities of the connectives. We now explore this phenomenon, turning to the task of giving polarized logic a Curry-Howard interpretation.

3 The language of values and continuations (and their duals)

Based on the forgoing logical analysis, in this section we define a programming language called the calculus of unity,[7] or **CU**, which seamlessly combines strict and lazy evaluation. The verificationist or positive connectives become constructors for strict types, the pragmatist or negative connectives lazy type constructors, and the five logical judgments (intuitionistic/co-intuitionistic truth/falsehood and contradiction) typing judgments for five distinct programming constructs (values, continuations, expressions, *covalues,* and statements), as exhibited in Figure 3.

We begin by defining a canonical fragment of **CU**, translating each logical rule mechanically into a typing rule. The rules for justified refutation and proof-by-contradiction now become rules for type-checking continuations and expressions defined by *pattern-matching.* This canonical fragment is sufficient for resolving many interesting questions about typing (in particular the question of subtyping for refinement types, as we will see in Section 4), but it is insufficient as a theoretical framework for studying *computation,* because for one, every term is already fully evaluated. Thus we *internalize* the principles of identity and cut as typing rules, and give a reduction relation on cuts as the operational semantics of **CU**. But this is still not enough, because (by the consistency of logic!) there are no closed programs to execute. Following [Gir01], we solve this problem by adding a single closed statement representing successful termination. This may appear exotic from a logical perspective (it internalizes inconsistency), but from

[7] With apologies to [Gir93].

an operational standpoint, all that is going on is we are specifying an observable result. We then finish this section with a simple statement and proof of type safety as a special case of cut-elimination on "closed sequents."

3.1 Canonical CU

Figure 4 gives the positive fragment of **CU**, including call-by-value continuations, strict products and sums. The value typing judgment $\Phi \vdash V \stackrel{\text{val}}{:} P$ is simply an annotation of the truth judgment $\Phi \vdash P$ *true* of Figure 1 with proof terms. It should also be mostly recognizable, for example the rule

$$\frac{\Phi \vdash V_1 \stackrel{\text{val}}{:} P \quad \Phi \vdash V_2 \stackrel{\text{val}}{:} Q}{\Phi \vdash \langle V_1, V_2 \rangle \stackrel{\text{val}}{:} P \otimes Q} \gg\otimes$$

which constructs a strict product out of a pair of values. The reader can without much danger take our use of the word "value" in its usual programming languages sense, and these typing rules thus impose a "value restriction" of sorts—but in Section 4.2 we will show how to derive rules for the more general case. Slightly less standard (but still straightforward) is the rule

$$\frac{\Phi \vdash K \stackrel{\text{cnt}}{:} P}{\Phi \vdash \text{con}(K) \stackrel{\text{val}}{:} \underline{v}\neg P} \gg^{\underline{v}}_{\neg}$$

which constructs a value out of a continuation, deferring to the continuation typing judgment $\Phi \vdash K \stackrel{\text{cnt}}{:} P$.

The main novelty of the positive fragment is in the method for checking continuations, encoded as the rule *blur*$\ll$:

$$\frac{\forall v \in \|P\| \, \forall \Phi' \in (v \stackrel{\text{val}}{:} P)^{-1} \quad \Phi, \Phi' \vdash K(v) \stackrel{\text{stm}}{:} \cdot}{\Phi \vdash K \stackrel{\text{cnt}}{:} P}$$

A continuation K may be thought of as a partial map from patterns to statements, and $K(v)$ represents the statement K would execute given a value matching pattern v. We write $K = \{v_1 \mapsto S_1 \mid \cdots \mid v_n \mapsto S_n\}$ to describe a continuation defined on finitely many patterns (so $K(v) = S_i$ if $v = v_i$ modulo renaming of variables, and $K(v)$ is undefined otherwise), but abstractly continuations need not satisfy this finiteness restriction. Operationally, when K receives a value V matching pattern v, it will execute $K(v)$ (assuming $K(v)$ is defined), after substituting the components of V for the variables bound by v. Intuitively, then, the premise of the continuation checking rule says that K can handle any value of type P: for all patterns v matching values of type P, for any $\Phi' \in (v \stackrel{\text{val}}{:} P)^{-1}$ decomposing the pattern hypothesis, $K(v)$ is defined and well-typed under the additional assumptions Φ'. (In the usual terminology of pattern-matching, we can see the premise that $K(v)$ be defined as requiring that K is *exhaustive*. The fact that K is a function guarantees it is *non-redundant*.)

Observe that in the world defined by Figure 4, $(v \stackrel{\text{val}}{:} P)^{-1}$ is always a singleton, i.e., a pattern hypothesis can always be decomposed uniquely into types for

$$
\begin{array}{lll}
\text{Values} & V & ::= x \mid \langle\rangle \mid \langle V_1, V_2\rangle \mid \mathrm{inl}(V) \mid \mathrm{inr}(V) \mid \mathrm{con}(K) \\
\text{Value patterns} & v & ::= x \mid \langle\rangle \mid \langle v_1, v_2\rangle \mid \mathrm{inl}(v) \mid \mathrm{inr}(v) \mid \mathrm{con}(\overline{u}) \\
\text{Continuations} & K & ::= \{v_1 \mapsto S_1 \mid \cdots \mid v_n \mapsto S_n\} \\
\text{Statements} & S & ::= V \rhd \overline{u}
\end{array}
$$

$$\boxed{\Phi \vdash V \overset{\mathrm{val}}{:} P}$$

$$\frac{x \overset{\mathrm{val}}{:} X \in \Phi}{\Phi \vdash x \overset{\mathrm{val}}{:} X} \gg X \qquad \frac{\Phi \vdash K \overset{\mathrm{cnt}}{:} P}{\Phi \vdash \mathrm{con}(K) \overset{\mathrm{val}}{:} \overset{v}{\neg} P} \gg^{\overset{v}{\neg}}$$

$$\frac{}{\Phi \vdash \langle\rangle \overset{\mathrm{val}}{:} 1} \gg 1 \qquad \frac{\Phi \vdash V_1 \overset{\mathrm{val}}{:} P \quad \Phi \vdash V_2 \overset{\mathrm{val}}{:} Q}{\Phi \vdash \langle V_1, V_2\rangle \overset{\mathrm{val}}{:} P \otimes Q} \gg \otimes$$

$$\text{(no rule for 0)} \qquad \frac{\Phi \vdash V \overset{\mathrm{val}}{:} P}{\Phi \vdash \mathrm{inl}(V) \overset{\mathrm{val}}{:} P \oplus Q} \quad \frac{\Phi \vdash V \overset{\mathrm{val}}{:} Q}{\Phi \vdash \mathrm{inr}(V) \overset{\mathrm{val}}{:} P \oplus Q} \gg \oplus$$

$$\boxed{\Phi \vdash K \overset{\mathrm{cnt}}{:} P}$$

$$\frac{\forall v \in \|P\| \, \forall \Phi' \in (v \overset{\mathrm{val}}{:} P)^{-1} \quad \Phi, \Phi' \vdash K(v) \overset{\mathrm{stm}}{:} \cdot}{\Phi \vdash K \overset{\mathrm{cnt}}{:} P} \; \mathit{blur} \ll$$

$$\boxed{\Phi \vdash S \overset{\mathrm{stm}}{:} \cdot}$$

$$\frac{\Phi \vdash V \overset{\mathrm{val}}{:} P \quad \overline{u} \overset{\mathrm{cnt}}{:} P \in \Phi}{\Phi \vdash V \rhd \overline{u} \overset{\mathrm{stm}}{:} \cdot} \; \mathit{focus} \gg$$

. . . . . .

$$\boxed{\Phi \Rightarrow v \overset{\mathrm{val}}{:} A}$$

$$\frac{}{x \overset{\mathrm{val}}{:} X \Rightarrow x \overset{\mathrm{val}}{:} X} \qquad \frac{}{\overline{u} \overset{\mathrm{cnt}}{:} A \Rightarrow \mathrm{con}(\overline{u}) \overset{\mathrm{val}}{:} \overset{v}{\neg} A}$$

$$\frac{}{\cdot \Rightarrow \langle\rangle \overset{\mathrm{val}}{:} 1} \qquad \frac{\Phi_1 \Rightarrow v_1 \overset{\mathrm{val}}{:} A \quad \Phi_2 \Rightarrow v_2 \overset{\mathrm{val}}{:} B}{\Phi_1, \Phi_2 \Rightarrow \langle v_1, v_2\rangle \overset{\mathrm{val}}{:} A \otimes B}$$

$$\text{(no rule for 0)} \qquad \frac{\Phi \Rightarrow v \overset{\mathrm{val}}{:} A}{\Phi \Rightarrow \mathrm{inl}(v) \overset{\mathrm{val}}{:} A \oplus B} \quad \frac{\Phi \Rightarrow v \overset{\mathrm{val}}{:} B}{\Phi \Rightarrow \mathrm{inr}(v) \overset{\mathrm{val}}{:} A \oplus B}$$

$$\|P\| = \left\{ v \mid \exists\, \Phi.\Phi \Rightarrow v \overset{\mathrm{val}}{:} P \right\} \qquad (v \overset{\mathrm{val}}{:} P)^{-1} = \left\{ \Phi \mid \Phi \Rightarrow v \overset{\mathrm{val}}{:} P \right\}$$

Figure 4: Canonical **CU** (positive fragment)

its variable bindings (e.g., $(\langle \mathrm{inr}(\mathrm{con}(\overline{u})), x\rangle \overset{\mathrm{val}}{:} (P \oplus^{y} \neg Q) \otimes X)^{-1} = \{(\overline{u} \overset{\mathrm{cnt}}{:} Q, x \overset{\mathrm{val}}{:} X)\}$). Another way of putting this is that each context in $(P\ \mathit{true})^{-1}$ corresponds to a unique pattern. However, we will not rely on this fact—and indeed, it will cease to hold once we extend the type system with positive unions.

Finally, in the canonical fragment, we distinguish continuation variables $\overline{u}$ from continuations K, and use the former in statements $V \rhd \overline{u}$. Intuitively, this statement represents the action of throwing V to the continuation instantiated for $\overline{u}$. It is checked with rule $focus{\gg}$

$$\frac{\Phi \vdash V \overset{\mathrm{val}}{:} P \quad \overline{u} \overset{\mathrm{cnt}}{:} P \in \Phi}{\Phi \vdash V \rhd \overline{u} \overset{\mathrm{stm}}{:} \cdot}$$

which finds a type for $\overline{u}$ in the context, and verifies that the value has the same type. Again, it is a fact of the present system that if a type for $\overline{u}$ exists then it is unique—but again, we will eventually extend the type system (with positive intersections) so that this property fails.

We now turn to the negative fragment of **CU**, given in Figure 5. Of course, the negative constructs are simply dual to the negative ones... but perhaps deserving of their own explanation. Whereas positive types are defined by their value constructors, negative types are defined by their destructors, or, to put it more symmetrically, by their continuation constructors. We call these continuations in restricted form *covalues.* So for example, the covalue for a lazy pair is either of the form $\mathrm{fst}(-)$ or $\mathrm{snd}(-)$:

$$\frac{\Phi \vdash C \overset{\mathrm{cov}}{:} M}{\Phi \vdash \mathrm{fst}(C) \overset{\mathrm{cov}}{:} M \& N} \qquad \frac{\Phi \vdash C \overset{\mathrm{cov}}{:} N}{\Phi \vdash \mathrm{snd}(C) \overset{\mathrm{cov}}{:} M \& N}\ \&{\ll}$$

So to speak, this pair of rules forces the continuation to "choose" whether to project the first or the second component. Now, laziness comes from the fact that *expressions* can be defined by pattern-matching against these choices. We can think of the lazy pair expression $E = \{\mathrm{fst}(c) \mapsto S_1 \mid \mathrm{snd}(c) \mapsto S_2\}$ as "asking" its continuation to make a decision between the two branches S_1 and S_2. Since they are only conditionally executed, in spirit S_1 and S_2 are completely unrestricted. For instance, they could be non-terminating or effectful (although we have not yet described how to write down such statements).

As we saw for the corresponding logical systems, perhaps the most interesting behavior arises from the interaction of the positive and negative fragments. In Section 2.3, we interpreted the shift operators as modalities, embedding co-intuitionistic truth-by-contradiction into direct intuitionistic truth, and likewise intuitionistic falsehood-by-contradiction into direct co-intuitionistic falsehood. They now have a very concrete operational reading: a lazy expression $E \overset{\mathrm{exp}}{:} N$ can be *suspended* and treated as a value $\exp(E) \overset{\mathrm{val}}{:} {\downarrow}N$, while a continuation $K \overset{\mathrm{cnt}}{:} P$ can be *captured* and treated as a covalue $\mathrm{con}(K) \overset{\mathrm{cov}}{:} {\uparrow}P$. With these coercions (typing rules given in Figure 6) we can now directly mix the two fragments in a controlled manner.

Let us begin by considering callcc, which has long been associated with Curry-Howard interpretations of classical logic (cf. [Gri90]). Given a statement abstracting in a P-continuation, we can construct an expression of type ${\uparrow}P$:

Covalues	C	$::= \overline{x} \mid [] \mid [C_1, C_2] \mid \mathrm{fst}(C) \mid \mathrm{snd}(C) \mid \exp(E)$
Covalue patterns	c	$::= \overline{x} \mid [] \mid [c_1, c_2] \mid \mathrm{fst}(c) \mid \mathrm{snd}(c) \mid \exp(u)$
Expressions	E	$::= \{c_1 \mapsto S_1 \mid \cdots \mid c_n \mapsto S_n\}$
Statements	S	$::= u \lhd C$

$\boxed{\Phi \vdash C \overset{\mathrm{cnt}}{:} N}$

$$\frac{\overline{x} \overset{\mathrm{cov}}{:} \overline{X} \in \Phi}{\Phi \vdash \overline{x} \overset{\mathrm{cov}}{:} \overline{X}}\ X{\ll} \qquad \frac{\Phi \vdash E \overset{\mathrm{exp}}{:} N}{\Phi \vdash \exp(E) \overset{\mathrm{cov}}{:} \neg_n N}\ \neg_n{\ll}$$

$$\text{(no rule for } \top) \qquad \frac{\Phi \vdash C \overset{\mathrm{cov}}{:} M}{\Phi \vdash \mathrm{fst}(C) \overset{\mathrm{cov}}{:} M \& N} \quad \frac{\Phi \vdash C \overset{\mathrm{cov}}{:} N}{\Phi \vdash \mathrm{snd}(C) \overset{\mathrm{cov}}{:} M \& N}\ \&{\ll}$$

$$\frac{}{\Phi \vdash [] \overset{\mathrm{cov}}{:} \bot}\ \bot{\ll} \qquad \frac{\Phi \vdash C_1 \overset{\mathrm{cov}}{:} M \quad \Phi \vdash C_2 \overset{\mathrm{cov}}{:} N}{\Phi \vdash [C_1, C_2] \overset{\mathrm{cov}}{:} M \parr N}\ \parr{\ll}$$

$\boxed{\Phi \vdash E \overset{\mathrm{exp}}{:} N}$

$$\frac{\forall c \in \|N\|\ \forall \Phi' \in (c \overset{\mathrm{cov}}{:} N)^{-1} \quad \Phi, \Phi' \vdash E(c) \overset{\mathrm{stm}}{:} \cdot}{\Phi \vdash E \overset{\mathrm{exp}}{:} N}\ {\gg}\mathit{blur}$$

$\boxed{\Phi \vdash S \overset{\mathrm{stm}}{:} \cdot}$

$$\frac{u \overset{\mathrm{exp}}{:} N \in \Phi \quad \Phi \vdash C \overset{\mathrm{cov}}{:} N}{\Phi \vdash u \lhd C \overset{\mathrm{stm}}{:} \cdot}\ {\ll}\mathit{focus}$$

$\boxed{\Phi \Rightarrow c \overset{\mathrm{cov}}{:} A}$

$$\frac{}{\overline{x} \overset{\mathrm{cov}}{:} X \Rightarrow x \overset{\mathrm{cov}}{:} X} \qquad \frac{}{u \overset{\mathrm{exp}}{:} A \Rightarrow \exp(u) \overset{\mathrm{cov}}{:} \neg_n A}$$

$$\text{(no rule for } \top) \qquad \frac{\Phi \Rightarrow c \overset{\mathrm{cov}}{:} A}{\Phi \Rightarrow \mathrm{fst}(c) \overset{\mathrm{cov}}{:} A \& B} \quad \frac{\Phi \Rightarrow c \overset{\mathrm{cov}}{:} B}{\Phi \Rightarrow \mathrm{snd}(c) \overset{\mathrm{cov}}{:} A \& B}$$

$$\frac{}{\cdot \Rightarrow [] \overset{\mathrm{cov}}{:} 0} \qquad \frac{\Phi_1 \Rightarrow c_1 \overset{\mathrm{cov}}{:} A \quad \Phi_2 \Rightarrow c_2 \overset{\mathrm{cov}}{:} B}{\Phi_1, \Phi_2 \Rightarrow [c_1, c_2] \overset{\mathrm{cov}}{:} A \parr B}$$

$$\|N\| = \left\{c \mid \exists\, \Phi.\Phi \Rightarrow c \overset{\mathrm{cov}}{:} N\right\} \qquad (c \overset{\mathrm{cov}}{:} N)^{-1} = \left\{\Phi \mid \Phi \Rightarrow c \overset{\mathrm{cov}}{:} N\right\}$$

Figure 5: Canonical **CU** (negative fragment)

Value	V	$::= \cdots \mid \exp(E)$
Covalue	C	$::= \cdots \mid \mathrm{con}(K)$
Value pattern	v	$::= \cdots \mid \exp(u)$
Covalue pattern	c	$::= \cdots \mid \mathrm{con}(\overline{u})$

$$\frac{\Phi \vdash E \overset{\exp}{:} N}{\Phi \vdash \exp(E) \overset{\mathrm{val}}{:} \downarrow N} \gg\downarrow \qquad \frac{\Phi \vdash K \overset{\exp}{:} P}{\Phi \vdash \mathrm{con}(K) \overset{\mathrm{cov}}{:} \uparrow P} \uparrow\ll$$

$$u \overset{\exp}{:} N \Rightarrow \exp(u) \overset{\mathrm{val}}{:} \downarrow N \qquad \overline{u} \overset{\mathrm{cnt}}{:} P \Rightarrow \mathrm{con}(\overline{u}) : \uparrow P$$

Figure 6: Canonical **CU** (shift operators)

$$\frac{\Phi, \overline{u} \overset{\mathrm{cnt}}{:} P \vdash S \overset{\mathrm{stm}}{:} \cdot}{\Phi \vdash \{\mathrm{con}(\overline{u}) \mapsto S\} \overset{\exp}{:} \uparrow P} \gg blur$$

As will be formalized in the operational semantics, once this expression is thrown a captured continuation $\mathrm{con}(K)$, evaluation proceeds on $[K/\overline{u}]S$, so that the expression truly has the behavior of callcc. But here the shift explicitly marks the presence of a "control effect."

Now as we explained in Section 2.4, although the logic encoded by canonical **CU** is not classical in the traditional sense, being rather a careful combination of intuitionistic and co-intuitionistic reasoning, it may be seen as a *focusing* system for the classical sequent calculus. Suppose we had tried to include callcc directly in the language with a rule similar to that in [CH00]:

$$\frac{\Phi, \overline{u} \overset{\mathrm{cnt}}{:} P \vdash S \overset{\mathrm{stm}}{:} \cdot}{\Phi \vdash \mu\overline{u}.S : P}\ callcc??$$

Given a statement abstracting in a P-continuation, this rule constructs a (is it a "value"?..."expression"?...let's just say) term of type P. The fact that this construct does not quite fit our terminology should already be a warning—it is not a value in the ordinary sense because it is effectful, but nor is it an expression in our sense because it has positive type. Perhaps if we had included callcc we may have adopted a more "value"-neutral terminology—but this mismatch reflects a deeper logical problem. Ignoring proof terms, the rule's premise establishes that the assumption P *untrue* entails a contradiction—hence if we read the conclusion as P *true*, it destroys the interpretation of that judgment as representing *intuitionistic* truth, distinct from co-intuitionistic truth. Likewise if we rewrite the rule in sequent form (via the translation of Section 2.4):

$$\frac{\Gamma \to \Delta, P}{\Gamma \to \Delta \gg P}$$

Now it violates the focusing restriction: it illegally loses right-focus on a positive formula. Conceivably, we could read the conclusion as P *unfalse*, but this would just make the coercion from positive to negative formulas implicit.

In general, loosely speaking we can treat expressions of type $\uparrow P$ as "effectful computations computing a value of type P." We should emphasize that this is different from the standard monadic encoding, using the double-negation monad. Whereas the type system enforces that the *only* way to use an expression of type $\uparrow P$ is to pass it a captured continuation $\mathrm{con}(K) \stackrel{\mathrm{cov}}{:} \uparrow P$, one must explicitly maintain a monadic discipline on values of type $\overset{v}{\neg}\overset{v}{\neg} P$. We can make a similar observation for the mixing of strict and lazy sums and products. There is a well-known monadic encoding of lazy pairs using strict pairs: a lazy pair of suspended computations could be represented as $\overset{v}{\neg}\overset{v}{\neg} P \otimes \overset{v}{\neg}\overset{v}{\neg} Q$. But using the shift operators, we also have a more direct encoding as $\uparrow P \& \uparrow Q$. As first observed by Filinski, there is a dual phenomenon for sums [Fil89]. Note that what we call lazy sums (following Filinski's example) are *not* what most call-by-need languages (e.g., Haskell) call sums: upon pattern-matching, the latter are eagerly reduced down to a tag before computation can proceed. Filinski shows how to simulate this behavior, effectively by encoding sums as $\overset{n}{\neg}\overset{n}{\neg} M \parr \overset{n}{\neg}\overset{n}{\neg} N$ [Fil89, §2.5.3]. But the shift operators enable a more direct encoding as $\uparrow(\downarrow M \oplus \downarrow N)$. Intuitively, this is the type of a computation computing a tagged, lazy expression.

3.2 Computation in CU

The type system thus far presented is in perfect isomorphism with the deductive system of Section 2, in the sense that it was obtained simply by annotating the logical rules with proof terms. Let us therefore examine the two identity principles and two cut principles identified in Section 2.2:

- if P *untrue* $\in \Phi$ then $\Phi \vdash P$ *untrue*
- if N *unfalse* $\in \Phi$ then $\Phi \vdash N$ *unfalse*
- if $\Phi \vdash P$ *true* and $\Phi \vdash P$ *untrue* then $\Phi \vdash$ *contra*
- if $\Phi \vdash N$ *true* and $\Phi \vdash N$ *untrue* then $\Phi \vdash$ *contra*

In canonical **CU**, these identity principles are analogues of full η-expansion. For example, the identity $X \oplus Y$ *untrue* $\vdash X \oplus Y$ *untrue* is witnessed by the following proof-term:

$$\overline{u} \stackrel{\mathrm{cnt}}{:} X \oplus Y \vdash \{\mathrm{inl}(x) \mapsto \mathrm{inl}(x) \rhd \overline{u} \mid \mathrm{inr}(y) \mapsto \mathrm{inr}(y) \rhd \overline{u}\} \stackrel{\mathrm{cnt}}{:} X \oplus Y$$

The cut principles are analogues of full β-reduction. For example, a cut of $\Phi \vdash \mathrm{inr}(\mathrm{con}(K)) \stackrel{\mathrm{val}}{:} X \oplus \overset{v}{\neg} Q$ against $\Phi \vdash \{\mathrm{inl}(x) \mapsto S_1, \mathrm{inr}(\mathrm{con}(\overline{u})) \mapsto V' \rhd \overline{u}\} \stackrel{\mathrm{cnt}}{:} X \oplus \overset{v}{\neg} Q$ is reduced (assuming $\overline{u}$ is not free in V') to a cut of $\Phi \vdash V' \stackrel{\mathrm{val}}{:} Q$ against $\Phi \vdash K \stackrel{\mathrm{cnt}}{:} Q$, which is then further reduced based on the structure of V' and K, eventually yielding a canonical statement S such that $\Phi \vdash S \stackrel{\mathrm{stm}}{:} \cdot$.

Terms of canonical **CU** are therefore always "η-expanded, β-reduced." While this makes for an exceptionally simple type theory (see [WCPW02] for the benefits derived from this property in a logical framework), it is problematic from the point of view of a programming language. Forcing expansion means that programs may get much larger (consider the expansion of $\overline{u} \stackrel{\mathrm{cnt}}{:} (P_1 \oplus Q_1) \otimes \cdots \otimes (P_n \oplus Q_n)$

Continuations	K	$::= \cdots \mid \overline{u}$
Expressions	E	$::= \cdots \mid u$
Statements	S	$::= \cdots \mid V \rhd K \mid E \lhd C$

$$\frac{\overline{u} \overset{\text{cnt}}{:} P \in \Phi}{\Phi \vdash \overline{u} \overset{\text{cnt}}{:} P}\ id\ll \qquad \frac{\Phi \vdash V \overset{\text{val}}{:} P \quad \Phi \vdash K \overset{\text{cnt}}{:} P}{\Phi \vdash V \rhd K \overset{\text{stm}}{:} \cdot}\ cut\gg$$

$$\frac{u \overset{\text{exp}}{:} N \in \Phi}{\Phi \vdash u \overset{\text{exp}}{:} N}\ \gg id \qquad \frac{\Phi \vdash E \overset{\text{exp}}{:} N \quad \Phi \vdash C \overset{\text{cov}}{:} N}{\Phi \vdash E \lhd C \overset{\text{stm}}{:} \cdot}\ \ll cut$$

Figure 7: **CU** identity and cut

for example). Forcing reduction is even worse—it means that programs are already evaluated!

So, we must *internalize* the identity and cut principles within the type system—these rules are given in Figure 7. The two identity rules then allow more compact proof terms, while the process of eliminating the two cut rules turns into a notion of evaluation for **CU** statements. To explain the latter, let us start by explaining the form of pattern-matching and substitution used implicitly above.

Definition. *A* substitution σ *is a mapping from variables to terms, such that* σ *maps (co)value variables to (co)value variables, and continuation (expression) variables to continuations (expressions).*

Proposition (Factorization). *Any value* V *factors uniquely as* $V = \sigma(v)$*, for some pattern* v *and substitution* σ*. Likewise, any covalue* C *factors uniquely as* $C = \sigma(c)$ *for some* c *and* σ*. (Uniqueness is modulo renaming, since patterns bind variables.)*

One way of understanding factorization is that values and covalues are just trees with leaves labelled by (co)value variables, continuations, or expressions. The proposition says that any tree can be represented uniquely by its internal nodes (a pattern) and a list of labels for the leaves (a substitution). Using this factorization, we can state very simple rules for reducing cuts:

$$\begin{array}{lcll} \sigma(v) \rhd K & \mapsto & \sigma(K(v)) & \text{if } K(v) \text{ defined} \\ E \lhd \sigma(c) & \mapsto & \sigma(E(c)) & \text{if } E(c) \text{ defined} \end{array}$$

By iterating the reduction relation, we have a small-step operational semantics for closed **CU** statements.

But now we come to a paradox. In the language we have defined, every closed statement has the form $V \rhd \{v_1 \mapsto S_1 \mid \cdots \mid v_n \mapsto S_n\}$ or $\{c_1 \mapsto S_1 \mid \cdots \mid c_n \mapsto S_n\} \lhd C$, and thus either reduces to a new (closed) statement, or else gets stuck on pattern-matching failure. Repeating the argument, every closed statement either eventually gets stuck or else does not terminate. Yet certainly *well-typed*

Statements $S \quad ::= \cdots \mid \text{done} \mid \text{fail}$

$$\frac{}{\Phi \vdash \text{done} \overset{\text{stm}}{\vdots} \cdot}\ \textit{done} \qquad \text{(no rule for fail)}$$

Figure 8: **CU** termination and failure

closed statements should not get stuck—and it would be bizarre if they all failed to terminate.

The answer to this paradox is simple: we have not yet defined any closed well-typed statements! This is actually obvious when we consider that well-typed statements of **CU** (as we have described it so far) correspond to proofs (possibly using the admissible cut and identity principles) of contradiction through intuitionistic and co-intuitionistic reasoning. A *closed* well-typed statement would therefore be a proof of the inconsistency of logic. One might react to this logical barrier to programmer productivity by considering open reduction (e.g., by adding a "top-level continuation"), but an easier solution is to simply augment the language with a single, closed well-typed statement representing successful termination. This statement is named "done", and for symmetry we can also add a single, *ill*-typed statement "fail", representing pattern-matching failure:[8]

$$\begin{array}{lcll} \sigma(v) \rhd K & \mapsto & \text{fail} & \text{if } K(v) \text{ undefined} \\ E \lhd \sigma(c) & \mapsto & \text{fail} & \text{if } E(c) \text{ undefined} \end{array}$$

The typing rules for both statements are given in Figure 8. Of course, a real programming language would have a richer set of statements—but for our purposes these two are enough.

3.3 Type safety and cut-elimination

We now prove type safety and a "partial" cut-elimination theorem for the complete language defined thus far, i.e., the union of Figures 4–8. The proof is just a slight modification of the proof of cut-admissibility from Section 2.2—but first we need some notation.

We use $t : \tau$ to stand for an arbitrary conclusion $V \overset{\text{val}}{\vdots} P$, $K \overset{\text{cnt}}{\vdots} P$, $E \overset{\text{exp}}{\vdots} N$, $C \overset{\text{cov}}{\vdots} N$, or $S \overset{\text{stm}}{\vdots} \cdot$, and $l : \tau$ for an arbitrary hypothesis $x \overset{\text{val}}{\vdots} X$, $\overline{u} \overset{\text{cnt}}{\vdots} P$, $u \overset{\text{exp}}{\vdots} N$, or $\overline{x} \overset{\text{cov}}{\vdots} \overline{X}$.

Definition. *Let σ be a substitution with domain disjoint from Φ. We say that $\Phi \vdash \sigma : \Phi'$ if for all hypotheses $l : \tau \in \Phi'$, $\Phi \vdash \sigma(l) : \tau$.*

Now we can restate the inversion properties of Section 2.2 in terms of **CU**:

[8] A reader familiar with Ludics may recognize these as alternative spellings of "daimon" and "faith".

$$
\begin{array}{lcll}
\sigma(v) \rhd K & \mapsto & \sigma(K(v)) & \text{if } K(v) \text{ defined} \\
\sigma(v) \rhd K & \mapsto & \text{fail} & \text{if } K(v) \text{ undefined} \\
E \lhd \sigma(c) & \mapsto & \sigma(E(c)) & \text{if } E(c) \text{ defined} \\
E \lhd \sigma(c) & \mapsto & \text{fail} & \text{if } E(c) \text{ undefined}
\end{array}
$$

Figure 9: Small-step semantics (closed cut-elimination)

Property (Inversion). *(pos.)* $\Phi \vdash \sigma(v) \stackrel{val}{:} P$ *iff* $\exists \Phi' \in (v \stackrel{val}{:} P)^{-1}$ *such that* $\Phi \vdash \sigma : \Phi'$*; (neg.)* $\Phi \vdash \sigma(c) \stackrel{cov}{:} N$ *iff* $\exists \Phi' \in (c \stackrel{cov}{:} N)^{-1}$ *such that* $\Phi \vdash \sigma : \Phi'$.

Proof. Both directions immediate by induction on P (resp. N). □

For the proof of safety, in fact we only need the forward direction of inversion (reduction). We also need a substitution lemma.

Lemma (Substitution). *If* $\Phi, \Phi' \vdash t : \tau$ *and* $\Phi \vdash \sigma : \Phi'$ *then* $\Phi \vdash \sigma(t) : \tau$.

Proof. By induction on the derivation of $\Phi, \Phi' \vdash t : \tau$. There are essentially two ways to use a hypothesis in Φ': either by using it directly in one of the hypothesis rules ($\gg X$, $X \ll$, $id \ll$, or $\gg id$)—in this case substitution is trivial—or else by focusing on it, for example like so:

$$\frac{\Phi, \Phi' \vdash V \stackrel{\text{val}}{:} P \quad \overline{u} \stackrel{\text{cnt}}{:} P \in \Phi'}{\Phi, \Phi' \vdash V \rhd \overline{u} \stackrel{\text{stm}}{:} \cdot} \; focus\gg$$

By the i.h., $\Phi \vdash \sigma(V) \stackrel{\text{val}}{:} P$, and moreover $\Phi \vdash \sigma(\overline{u}) \stackrel{\text{cnt}}{:} P$ by the assumption that $\Phi \vdash \sigma : \Phi'$. Hence we can replace the focus with a cut:

$$\frac{\Phi \vdash \sigma(V) \stackrel{\text{val}}{:} P \quad \Phi \vdash \sigma(\overline{u}) \stackrel{\text{cnt}}{:} P}{\Phi \vdash \sigma(V) \rhd \sigma(\overline{u}) \stackrel{\text{stm}}{:} \cdot} \; cut\gg$$

□

These two facts are enough to prove type safety. For reference, the reduction rules described in the previous section are included together in Figure 9, defining a small-step operational semantics for the focusing calculus. Let us refer to throws $V \rhd K$ or $E \lhd C$, where K and E are not variables, as *serious cuts* (so that, e.g., $V \rhd \overline{u}$ is not a serious cut, even though a derivation of $V \rhd \overline{u} \stackrel{\text{stm}}{:} \cdot$ may end in $cut\gg$). Because of the reification of pattern-matching failure, it is immediate that the transition relation is total on serious cuts.

Proposition (Progress). *If S is a serious cut then there exists an S' such that $S \mapsto S'$.*

The transition relation is actually deterministic on well-typed cuts, but we do not need this fact. Indeed, later we will explicitly extend the language with non-determinism.

Lemma (Preservation). *If* $\Phi \vdash S \stackrel{stm}{:} \cdot$ *and* $S \mapsto S'$ *then* $\Phi \vdash S' \stackrel{stm}{:} \cdot$.

$$\frac{}{\text{done} \Downarrow \text{done}} \qquad \frac{S \mapsto S' \quad S' \Downarrow S^*}{S \Downarrow S^*} \qquad \frac{}{\text{fail} \Downarrow \text{fail}}$$

$$\frac{V \Downarrow V^*}{V \rhd \overline{u} \Downarrow V^* \rhd \overline{u}} \qquad \frac{C \Downarrow C^*}{u \lhd C \Downarrow u \lhd C^*}$$

$$\frac{}{x \Downarrow x} \qquad \frac{}{\langle\rangle \Downarrow \langle\rangle} \qquad \frac{V_1 \Downarrow V_1^* \quad V_2 \Downarrow V_2^*}{\langle V_1, V_2\rangle \Downarrow \langle V_1^*, V_2^*\rangle} \qquad \frac{V \Downarrow V^*}{\text{inl}(V) \Downarrow \text{inl}(V^*)} \qquad \frac{V \Downarrow V^*}{\text{inr}(V) \Downarrow \text{inr}(V^*)}$$

$$\frac{}{\overline{x} \Downarrow \overline{x}} \qquad \frac{}{[] \Downarrow []} \qquad \frac{C_1 \Downarrow C_1^* \quad C_2 \Downarrow C_2^*}{[C_1, C_2] \Downarrow [C_1^*, C_2^*]} \qquad \frac{C \Downarrow C^*}{\text{fst}(C) \Downarrow \text{fst}(C^*)} \qquad \frac{C \Downarrow C^*}{\text{snd}(C) \Downarrow \text{snd}(C^*)}$$

$$\frac{K \Downarrow K^*}{\text{con}(K) \Downarrow \text{con}(K^*)} \qquad \frac{E \Downarrow E^*}{\text{exp}(E) \Downarrow \text{exp}(E^*)}$$

$$\frac{S_i \Downarrow S_i^* \quad i = 1..n}{\{v_i \mapsto S_i\}_{i=1..n} \Downarrow \{v_i \mapsto S_i^*\}_{i=1..n}} \qquad \frac{}{\overline{u} \Downarrow \overline{u}} \qquad \frac{}{u \Downarrow u} \qquad \frac{S_i \Downarrow S_i^* \quad i = 1..n}{\{c_i \mapsto S_i\}_{i=1..n} \Downarrow \{c_i \mapsto S_i^*\}_{i=1..n}}$$

Figure 10: Big-step semantics (general cut-elimination)

Proof. S is a serious cut, either positive ($S = \sigma(v) \rhd K$) or negative ($S = E \lhd \sigma(c)$). Consider the positive case: we must show that $S' = \sigma(K(v))$ is well-typed in Φ. By hypothesis that S is well-typed, there exists a P such that $\Phi \vdash \sigma(v) \overset{\text{val}}{:} P$ and $\Phi \vdash K \overset{\text{cnt}}{:} P$. The former implies (by inversion-reduction) that there exists a $\Phi' \in (v \overset{\text{val}}{:} P)^{-1}$ such that $\Phi \vdash \sigma : \Phi'$. The latter implies that $\Phi, \Phi' \vdash K(v) \overset{\text{stm}}{:} \cdot$. Hence by the substitution lemma, $\Phi \vdash \sigma(K(v)) \overset{\text{stm}}{:} \cdot$. The negative case is symmetric. □

Corollary (Type safety). *If S is a closed well-typed statement, then either $S = \text{done}$ or else there is an S' such that $S \mapsto S'$, and moreover any such S' is well-typed.*

Proof. If $S \neq \text{done}$, we can apply progress (since all closed well-typed statements are serious cuts) to obtain $S \mapsto S'$, and then preservation for $S' \overset{\text{stm}}{:} \cdot$. □

Type safety is effectively a cut-elimination theorem for closed sequents—although only "partial" cut-elimination, as it does not explicitly guarantee that the cut-elimination procedure terminates, only that *if* it terminates it yields a cut-free proof. Generalizing this to the open case is not difficult, after we lift the small-step transition relation on serious cuts to a big-step relation on arbitrary proof terms, written $t \Downarrow t^*$ (Figure 10).

Theorem (Partial cut-elimination). *If $\Phi \vdash t : \tau$ and $t \Downarrow t^*$ then $\Phi \vdash t^* : \tau$, and moreover the latter derivation is cut-free.*

Proof. By induction on the derivation of $t \Downarrow t^*$. In most cases we can simply invert the last rule of $\Phi \vdash t : \tau$, apply the induction hypothesis, and then reapply the rule. For example, suppose $\langle V_1, V_2\rangle \Downarrow \langle V_1^*, V_2^*\rangle$ for some V_1^* and V_2^* such

that $V_1 \Downarrow V_1^*$ and $V_2 \Downarrow V_2^*$. Well $\Phi \vdash \langle V_1, V_2\rangle \stackrel{\mathrm{val}}{:} P \otimes Q$ implies $\Phi \vdash V_1 \stackrel{\mathrm{val}}{:} P$ and $\Phi \vdash V_2 \stackrel{\mathrm{val}}{:} Q$, and we can apply the i.h. to obtain cut-free derivations of $\Phi \vdash V_1^* \stackrel{\mathrm{val}}{:} P$ and $\Phi \vdash V_2^* \stackrel{\mathrm{val}}{:} Q$, whence $\Phi \vdash \langle V_1^*, V_2^*\rangle \stackrel{\mathrm{val}}{:} P \otimes Q$. In the case of a throw $V \rhd \overline{u} \Downarrow V^* \rhd \overline{u}$, then the derivation of $\Phi \vdash V \rhd \overline{u} \stackrel{\mathrm{stm}}{:} \cdot$ ends in either *focus*$\gg$ with premises $\Phi \vdash V \stackrel{\mathrm{val}}{:} P$ and $\overline{u} \stackrel{\mathrm{cnt}}{:} P \in \Phi$, or *cut*$\gg$ with premises $\Phi \vdash V \stackrel{\mathrm{val}}{:} P$ and $\Phi \vdash \overline{u} \stackrel{\mathrm{cnt}}{:} P$. But the latter must have been derived by the *id*$\ll$ rule with $\overline{u} \stackrel{\mathrm{cnt}}{:} P \in \Phi$, and so in both cases we can apply the i.h. to obtain $\Phi \vdash V^* \stackrel{\mathrm{val}}{:} P$ and then *focus*$\gg$ on $\overline{u} \stackrel{\mathrm{cnt}}{:} P$ to obtain a cut-free derivation $\Phi \vdash V^* \rhd \overline{u} \stackrel{\mathrm{stm}}{:} \cdot$. Finally, in the case of $S \Downarrow S^*$ derived from $S \mapsto S'$ and $S' \Downarrow S^*$, we apply the preservation lemma to obtain $\Phi \vdash S' \stackrel{\mathrm{stm}}{:} \cdot$ and then the i.h. for $\Phi \vdash S^* \stackrel{\mathrm{stm}}{:} \cdot$. □

Naturally, one may ask whether cut-elimination really is total, i.e., whether $\Phi \vdash t : \tau$ implies that there exists a t^* such that $t \Downarrow t^*$. But this question is somewhat against the spirit of our present aim—although we have not explicitly included recursion in the language, the whole point is that our methodology is robust in the presence of effects such as non-termination. Thus we leave aside the question of totality for this fragment.[9]

In Section 4, we will see how our methodology extends in a straightforward way to intersection and union types. Before setting to that task, though, let us apply **CU** for a new spin on the old logical technique of double-negation translation.

3.4 Application: the computational content of classical focusing

In this section we use **CU** in order to give a computational proof of the completeness of the focusing strategy for classical sequent calculus, as claimed in Section 2.4.

Figure 11 gives the classical sequent calculus. We use Kleene's G_3 presentation, and omit nullary products and sums since their treatment is essentially the same as their binary versions. To make the relationship to **CU** visually evocative, we will adopt the sequent notation of Section 2.4, augmenting it with proof terms:

$$\begin{array}{rcl}
\Phi \vdash V \stackrel{\mathrm{val}}{:} P & \iff & V : (\Gamma_\Phi \to \Delta_\Phi \gg P) \\
\Phi \vdash K \stackrel{\mathrm{cnt}}{:} P & \iff & K : (P \ll \Gamma_\Phi \to \Delta_\Phi) \\
\Phi \vdash E \stackrel{\mathrm{exp}}{:} N & \iff & E : (\Gamma_\Phi \to \Delta_\Phi \gg N) \\
\Phi \vdash C \stackrel{\mathrm{cov}}{:} N & \iff & C : (N \ll \Gamma_\Phi \to \Delta_\Phi) \\
\Phi \vdash S \stackrel{\mathrm{stm}}{:} \cdot & \iff & S : (\Gamma_\Phi \to \Delta_\Phi)
\end{array}$$

The erasure operator $|-|$ converts a polarized (positive or negative) formula to an ordinary one by collapsing $\otimes$ and $\&$ to $\times$, $\oplus$ and $\parr$ to $+$, $\stackrel{v}{\neg}$ and $\stackrel{n}{\neg}$ to $\neg$, and erasing all shifts.[10] For example, $|X \otimes \downarrow(\overline{Y} \parr \top)| = X \times (\overline{Y} + 1)$.

[9] But see Section 5.

[10] The erasure operator does *not* collapse positive and negative atoms. To understand why it does not, think of polarity as a partitioning of the set of atoms.

$$\frac{}{X, \Gamma \to^c \Delta, X}\ \mathit{init} \qquad \frac{\Gamma \to^c \Delta, A \quad A, \Gamma \to^c \Delta}{\Gamma \to^c \Delta}\ \mathit{cut}$$

$$\frac{A, A \times B, \Gamma \to^c \Delta}{A \times B, \Gamma \to^c \Delta} \times L_1 \qquad \frac{B, A \times B, \Gamma \to^c \Delta}{A \times B, \Gamma \to^c \Delta} \times L_2 \qquad \frac{\Gamma \to^c \Delta, A \times B, A \quad \Gamma \to^c \Delta, A \times B, B}{\Gamma \to^c \Delta, A \times B} \times R$$

$$\frac{A, A + B, \Gamma \to^c \Delta \quad B, A + B, \Gamma \to^c \Delta}{A + B, \Gamma \to^c \Delta} + L \qquad \frac{\Gamma \to^c \Delta, A + B, A}{\Gamma \to^c \Delta, A + B} + R_1 \qquad \frac{\Gamma \to^c \Delta, A + B, B}{\Gamma \to^c \Delta, A + B} + R_2$$

$$\frac{\neg A, \Gamma \to^c \Delta, A}{\neg A, \Gamma \to^c \Delta} \neg L \qquad \frac{A, \Gamma \to^c \Delta, \neg A}{\Gamma \to^c \Delta, \neg A} \neg R$$

Figure 11: Classical sequent calculus

This operator is obviously not injective—many different polarized formulas map to the same ordinary formula (in fact infinitely many, because shifts can be arbitrarily composed). The focusing completeness theorem states that if $|\Gamma| \to^c |\Delta|$, then there exists a proper S such that $S : (\Gamma \to \Delta)$, where a proof term is *proper* if it contains no uses of done. More or less, this means that if a sequent of ordinary formulas is classically provable, then it has a focusing proof given *any polarization of the formulas.* Technically, some polarizations result in sequents containing non-atomic positive formulas in Γ, or non-atomic negative formulas in Δ, which are outside the syntax of **CU**. Thus, we use the following convention:

Definition. *An* extended context *may contain assumptions* $x \overset{val}{:} P$ *or* $\overline{x} \overset{cnt}{:} N$ *with* P *and* N *non-atomic. We write* $\Phi, x \overset{val}{:} P \vdash t : \tau$ *iff* $\forall v \in \|P\|$, $\forall \Phi' \in (v \overset{val}{:} P)^{-1}$, *we have* $\Phi, \Phi' \vdash [v/x]t : \tau$. *Similarly for* $\Phi, \overline{x} \overset{cnt}{:} N \vdash t : \tau$.

This is a purely syntactic convention, but it is compatible with the rules of inference of **CU**, i.e., each rule is valid for extended contexts under the above reading. As a convenience, we will also use variable patterns at non-atomic types to define continuations and expressions, being syntactic sugar for their expansions. For example, $\Phi \vdash \{x \mapsto S\} \overset{\text{cnt}}{:} P$ should be read as $\Phi \vdash \{v \mapsto [v/x]S \mid v \in \|P\|\} \overset{\text{cnt}}{:} P$.

Theorem (Focusing Completeness). *If* $|\Gamma| \to^c |\Delta|$ *then there exists a proper* S *such that* $S : (\Gamma \to \Delta)$.

Proof. By induction on the classical derivation, with a side induction on the formulas in Γ and Δ. We give a few of the cases for positive polarizations:

Case: (*cut*) By the inductive hypothesis we have an $S_1 : (\Gamma \to \Delta, \overline{u} \overset{\text{cnt}}{:} P)$ and an $S_2 : (x \overset{\text{val}}{:} P, \Gamma \to \Delta)$. Take $S = S_1[\{x \mapsto S_2\}/\overline{u}]$. By the substitution lemma, we have $S : (\Gamma \to \Delta)$.

Case: ($\times R$) By the i.h. we have $S_1 : (\Gamma \to \Delta, \overline{u} \stackrel{\text{cnt}}{:} P \otimes Q, \overline{u}_1 \stackrel{\text{cnt}}{:} P)$ and $S_2 : (\Gamma \to \Delta, \overline{u} \stackrel{\text{cnt}}{:} P \otimes Q, \overline{u}_2 \stackrel{\text{cnt}}{:} Q)$. We define S by:

$$S = [\{x_1 \mapsto [\{x_2 \mapsto \langle x_1, x_2 \rangle \rhd \overline{u}\} / \overline{u}_2] S_2\} / \overline{u}_1] S_1$$

The computational gloss of this proof term: "Begin by executing S_1, until possibly a call to $\overline{u}_1$ is made—if so, remember the argument x_1 and execute S_2 until (possibly) a call to $\overline{u}_2$ is made with argument x_2. Now we have $x_1 \stackrel{\text{val}}{:} P$ and $x_2 \stackrel{\text{val}}{:} Q$, so we can finish by throwing $\langle x_1, x_2 \rangle$ to $\overline{u}$." Note the choice of "left-to-right evaluation order" here is arbitrary. We could as well have taken

$$S' = [\{x_2 \mapsto [\{x_1 \mapsto \langle x_1, x_2 \rangle \rhd \overline{u}\} / \overline{u}_1] S_1\} / \overline{u}_2] S_2$$

Case: ($\neg R$) By the i.h., we have $S_1 : (x \stackrel{\text{val}}{:} P, \Gamma \to \Delta, \overline{u} \stackrel{\text{cnt}}{:} \stackrel{v}{\neg} P)$. Then take $S = \text{con}(\{x \mapsto S_1\}) \rhd \overline{u}$.

Case: ($+L$) We have $S_1 : (x_1 \stackrel{\text{val}}{:} P, x \stackrel{\text{val}}{:} P \oplus Q, \Gamma \to \Delta)$ and $S_2 : (x_2 \stackrel{\text{val}}{:} Q, x \stackrel{\text{val}}{:} P \oplus Q, \Gamma \to \Delta)$. Take $S = x \rhd \{\text{inl}(x_1) \mapsto S_1 \mid \text{inr}(x_2) \mapsto S_2\}$.

Case: ($\downarrow R$) Suppose $|\Gamma| \to^c |\Delta|, |\downarrow N|$. Then by definition $|\Gamma| \to^c |\Delta|, |N|$, and by the i.h. there exists a $S_1 : (\Gamma \to \Delta, \overline{x} \stackrel{\text{cov}}{:} N)$. Therefore we have $\exp(\{\overline{x} \mapsto S_1\}) \rhd \overline{u} : (\Gamma \to \Delta, \overline{u} \stackrel{\text{cnt}}{:} \downarrow N)$.

□

Letting $\vdash^c$ stand for classical truth, the following corollaries are immediate:

Corollary (Glivenko's Theorem). *if $\vdash^c |P|$ then there is a proper V such that $\vdash V \stackrel{val}{:} \downarrow\uparrow P$.*

Proof. Let $V = \exp(\{\text{con}(\overline{u}) \mapsto S\})$, where $S : (\cdot \to \overline{u} \stackrel{\text{cnt}}{:} P)$. □

Corollary. *if $\vdash^c |N|$ then there is a proper E such that $\vdash E \stackrel{exp}{:} N$.*

Proof. Let $E = \{\overline{x} \mapsto S\}$, where $S : (\cdot \to \overline{x} \stackrel{\text{cov}}{:} N)$. □

From these corollaries, one could reasonably claim that negative polarization allows a more direct encoding of classical truth—though of course the counter-claim would be that positive polarization allows a more direct encoding of classical falsehood.

4 Intersections, unions, and subtyping

With the methodology of Section 3 in place, adding intersections and unions to **CU** is a relatively simple exercise—they are encoded analogously to products and sums, without the associated term constructors. This implies, naturally, that there are *two* forms of intersection—one positive and one negative—just as

there are two forms of product, and likewise two forms of union In either case, we define the connective by its value/covalue introduction rules, and deduce the pattern-decomposition rule so as to satisfy the inversion property. Subtyping is *not* a part of this definition—rather, it is a derived notion, indeed the generalization of the identity principle.

In the below, positive and negative intersections and unions are distinguished by a polarity marker (e.g., $\pitchfork$ vs. $\barwedge$), but sometimes the marker is left out when clear from context (e.g., we write $P \cap Q$ rather than $P \pitchfork Q$).

4.1 Preliminaries

Much of the work on intersection and union types for practical programming has imposed a *refinement restriction* [FP91], viz. that the extended type system is only used to verify richer properties of programs that are already typable in the base type system (e.g., ML's), rather than to make additional programs typable. This restriction is achieved syntactically, by allowing only intersections/unions of types that are refinements of a common type. So for example, although the function $\lambda x.xx$ could be given type $(X \cap (X \to X)) \to X$ in a general intersection type system, that type is ill-formed in a refinement system for the simply-typed lambda calculus.

In the setting of **CU**, an intermediate stance appears to be reasonable: an intersection/union is well-formed so long as it combines types that have the same set of *patterns.* Thus ${}^{v}\neg P \cap {}^{v}\neg{}^{v}\neg P$ is still well-formed, because $\|{}^{v}\neg P\| = \|{}^{v}\neg{}^{v}\neg P\| = \{\mathrm{con}(\overline{u})\}$, and we take $\|{}^{v}\neg P \cap {}^{v}\neg{}^{v}\neg P\| = \{\mathrm{con}(\overline{u})\}$. However, $X \cup (X \otimes Y)$ is an illegal refinement type, since $\|X\| = \{x\}$ but $\|X \otimes Y\| = \{\langle x, y\rangle\}$. Explicitly, we do *not* take $\|X \cup (X \otimes Y)\| = \{x, \langle x, y\rangle\}$. In order to be able to define $\|-\|$ in the case of empty intersections and unions, the syntax of types includes pattern-set annotations: $\top^{\mathbb{V}}, \bot_{\mathbb{V}}$. Other than in the definition of $\|-\|$, though, these annotations do not play an explicit role in the type system and we will usually write them invisibly. There is an *implicit* side condition that to be able to assert $V \overset{\mathrm{val}}{:} P$, we must have $V = \sigma(v)$ for some $v \in \|P\|$ (and similarly to assert $C \overset{\mathrm{cov}}{:} N$). In particular, it is a presupposition that for all types τ appearing in judgments, $\|\tau\|$ is defined.

To decompose pattern hypotheses involving (positive) intersections and (negative) unions, we will find it necessary to allow multiple type assumptions for the same variable in a context. For example, a continuation variable $\overline{u}$ might have two types P_1 and P_2, or no types. Abusing notation, we will alternately write such sets of hypotheses either as $(\overline{u} \overset{\mathrm{cnt}}{:} P_1, \overline{u} \overset{\mathrm{cnt}}{:} P_2)$ and $(\cdot)$, or $(\overline{u} \overset{\mathrm{cnt}}{:} P_1, P_2)$ and $(\overline{u} \overset{\mathrm{cnt}}{:} \cdot)$—in general, we alternately write $(l : \tau_1, \ldots, l : \tau_n)$, or $(l : \tau_1, \ldots, \tau_n)$. The former allows us to treat contexts as sets and keep the notation $l : \tau \in \Phi$ to test whether τ is included among any of the types for l, while the latter is more honest about binding structure and conveniently locates all the types for a variable in one place. We write $\Phi_1 \wedge \Phi_2$ for the operation of combining two contexts mentioning the same variables, distinguished from the previously employed operation Φ_1, Φ_2 which combines disjoint contexts.

Positive type $\quad P, Q \ ::= \ \cdots \mid \top\!\!\top_+^{\mathbb{V}} \mid P \mathbin{\pitchfork} Q \mid \bot\!\!\bot_{\mathbb{V}}^+ \mid P \uplus Q$

$$\frac{}{\Phi \vdash V \stackrel{\text{val}}{:} \top\!\!\top_+} \gg\top\!\!\top_+ \qquad \frac{\Phi \vdash V \stackrel{\text{val}}{:} P \quad \Phi \vdash V \stackrel{\text{val}}{:} Q}{\Phi \vdash V \stackrel{\text{val}}{:} P \mathbin{\pitchfork} Q} \gg\pitchfork$$

$$\text{(no rule for } \bot\!\!\bot^+) \qquad \frac{\Phi \vdash V \stackrel{\text{val}}{:} P}{\Phi \vdash V \stackrel{\text{val}}{:} P \uplus Q} \quad \frac{\Phi \vdash V \stackrel{\text{val}}{:} Q}{\Phi \vdash V \stackrel{\text{val}}{:} P \uplus Q} \gg\uplus$$

$$\frac{}{\cdot \Rightarrow v \stackrel{\text{val}}{:} \top\!\!\top_+} \qquad \frac{\Phi_1 \Rightarrow v \stackrel{\text{val}}{:} P \quad \Phi_2 \Rightarrow v \stackrel{\text{val}}{:} Q}{\Phi_1 \wedge \Phi_2 \Rightarrow v \stackrel{\text{val}}{:} P \mathbin{\pitchfork} Q}$$

$$\text{(no rule for } \bot\!\!\bot^+) \qquad \frac{\Phi \Rightarrow v \stackrel{\text{val}}{:} P}{\Phi \Rightarrow v \stackrel{\text{val}}{:} P \uplus Q} \quad \frac{\Phi \Rightarrow v \stackrel{\text{val}}{:} Q}{\Phi \Rightarrow v \stackrel{\text{val}}{:} P \uplus Q}$$

$$\|\top\!\!\top_+^{\mathbb{V}}\| = \|\bot\!\!\bot_{\mathbb{V}}^+\| = \mathbb{V} \qquad \|P \mathbin{\pitchfork} Q\| = \|P \uplus Q\| = \mathbb{V} \quad \text{if } \|P\| = \|Q\| = \mathbb{V}$$

Figure 12: Positive intersections and unions

4.2 The positive (strict) case

Figure 12 describes positive intersection and union types. With the foregoing preface about the refinement restriction, the introduction rules are completely standard—or are they? They impose a value restriction, which did not figure in intersection type systems until [DP00] found that one was necessary for effectful, call-by-value languages. Of course *all* of the introduction rules for positive types in **CU** are value-typing rules, so what does this "value restriction" really mean? As we observed at the end of Section 3.1, expressions of type $\uparrow P$ may be seen as "possibly effectful computations" of type P—literally, they are expressions with access to their continuation. Now, the value-typing rules of Figure 4 may, in the case of products and sums, be *lifted* to expression-typing rules. For example, the following rule for introducing $\Uparrow(P \otimes Q)$ is derivable:

$$\frac{\Phi \vdash E_1 \stackrel{\text{exp}}{:} \uparrow P \quad \Phi \vdash E_2 \stackrel{\text{exp}}{:} \uparrow Q}{\Phi \vdash \langle E_1, E_2 \rangle \stackrel{\text{exp}}{:} \Uparrow(P \otimes Q)}$$

where $\langle E_1, E_2 \rangle$ is an abbreviation for the left-to-right evaluation $\langle E_1, E_2 \rangle = \{\text{con}(\overline{u}) \mapsto E_1 \lhd \text{con}(\{x \mapsto E_2 \lhd \text{con}(\{y \mapsto \langle x, y \rangle \rhd \overline{u}\})\})\}$. Now, we may perform a similar transformation to lift the union introduction rule to expressions:

$$\frac{\Phi \vdash E \stackrel{\text{exp}}{:} \uparrow P}{\Phi \vdash \{\text{con}(\overline{u}) \mapsto E \lhd \text{con}(\{x \mapsto x \rhd \overline{u}\})\} \stackrel{\text{exp}}{:} \Uparrow(P \uplus Q)}$$

By the premise, it suffices to check $\text{con}(\{x \mapsto x \rhd \overline{u}\})$ at type $\uparrow P$ under assumption $\overline{u} \stackrel{\text{cnt}}{:} P \uplus Q$, which reduces to checking $x \rhd \overline{u} \stackrel{\text{stm}}{:} \cdot$ under additional assumption

$x \overset{\text{val}}{:} P$. This is possible, because from $x \overset{\text{val}}{:} P$ we can derive $x \overset{\text{val}}{:} P \uplus Q$. Note that the expression in the conclusion is just an η-expansion of E, and indeed (as we will eventually prove) the following rule is admissible:

$$\frac{\Phi \vdash E \overset{\text{exp}}{:} \uparrow P}{\Phi \vdash E \overset{\text{exp}}{:} \uparrow (P \uplus Q)}$$

However, both the binary and empty intersection introduction rules can *not* be lifted to expression typing rules. Suppose we try the following:

$$\frac{\Phi \vdash E \overset{\text{exp}}{:} \uparrow P \quad \Phi \vdash E \overset{\text{exp}}{:} \uparrow Q}{\Phi \vdash \{\text{con}(\overline{u}) \mapsto E \lhd \text{con}(\{x \mapsto x \rhd \overline{u}\})\} \overset{\text{exp}}{:} \uparrow (P \pitchfork Q)} \; ??$$

From the premises, we know that it suffices to check $\text{con}(\{x \mapsto x \rhd \overline{u}\})$ under assumption $\overline{u} \overset{\text{cnt}}{:} P \pitchfork Q$ at either type $\uparrow P$ or $\uparrow Q$, which reduces to showing $x \rhd \overline{u} \overset{\text{stm}}{:} \cdot$ under an additional assumption of either $x \overset{\text{val}}{:} P$ or $x \overset{\text{val}}{:} Q$. But individually, either assumption is insufficient for verifying the statement—we require having both assumptions *simultaneously*. Since we do not, there is no way to derive the conclusion. Correspondingly, the "unrestricted" intersection-introduction rule is *not* admissible:

$$\frac{\Phi \vdash E \overset{\text{exp}}{:} \uparrow P \quad \Phi \vdash E \overset{\text{exp}}{:} \uparrow Q}{\Phi \vdash E \overset{\text{exp}}{:} \uparrow (P \pitchfork Q)} \; ??$$

In the case of empty intersection, this is even more obvious. An unrestricted $\mathbb{T}_+$-introduction rule would let us assert $\{\text{con}(\overline{u}) \mapsto \text{fail}\} \overset{\text{exp}}{:} \uparrow \mathbb{T}_+$ and then check $\{\text{con}(\overline{u}) \mapsto \text{fail}\} \lhd \text{con}(\{x \mapsto \text{done}\}) \overset{\text{stm}}{:} \cdot$, a statement that immediately reduces to fail, violating type-safety.

Let us turn now to the rules for decomposing pattern hypotheses. The definitions given are exactly those required for preserving the inversion property, which, recall, states that $\Phi \vdash \sigma(v) \overset{\text{val}}{:} P$ iff $\exists \Phi' \in (v \overset{\text{val}}{:} P)^{-1}$ such that $\Phi \vdash \sigma : \Phi'$. For example, the decomposition $(\text{con}(\overline{u}) : \neg P \cap \neg Q)^{-1} = \left\{(\overline{u} \overset{\text{cnt}}{:} P, Q)\right\}$ reflects that a value $\text{con}(K)$ has type $\neg P \cap \neg Q$ just when *both* $K \overset{\text{cnt}}{:} P$ and $K \overset{\text{cnt}}{:} Q$, while $(\text{con}(\overline{u}) : \neg P \cup \neg Q)^{-1} = \left\{(\overline{u} \overset{\text{cnt}}{:} P), (\overline{u} \overset{\text{cnt}}{:} Q)\right\}$ reflects that $\text{con}(K)$ has type $\neg P \cup \neg Q$ just when *either* $K \overset{\text{cnt}}{:} P$ or $K \overset{\text{cnt}}{:} Q$. We should observe that intersection and union types increase the complexity of type-checking in orthogonal directions. As was mentioned in the preliminaries, intersection pattern-decomposition makes it no longer the case that contexts always contain just a single type for each variable. Union decomposition, on the other hand, makes it no longer the case that the set $(v \overset{\text{val}}{:} P)^{-1}$ is always a singleton.

For a taste of the added expressivity that intersections and unions afford, let us define the type of booleans $\mathbb{B} = 1 \oplus 1$, along with refinements $\mathbb{T} = 1 \oplus \perp\!\!\!\perp$ and $\mathbb{F} = \perp\!\!\!\perp \oplus 1$ and syntactic sugar $t = \text{inl}\,\langle\rangle$, $f = \text{inr}\,\langle\rangle$. Note that by our convention $\|\mathbb{B}\| = \|\mathbb{T}\| = \|\mathbb{F}\| = \{t, f\}$, while

$$(t : \mathbb{B})^{-1} = (t : \mathbb{T})^{-1} = \{(\cdot)\} = (f : \mathbb{B})^{-1} = (f : \mathbb{F})^{-1}$$
$$(t : \mathbb{F})^{-1} = \emptyset = (f : \mathbb{T})^{-1}$$

Negative type $\quad M, N \;::=\; \cdots \mid \top\!\!\top^{\mathbb{C}} \mid M \mathbin{\&} N \mid \bot\!\!\bot_{\mathbb{C}} \mid M \oplus N$

$$\text{(no rule for } \top\!\!\top\text{)} \qquad \frac{\Phi \vdash C \overset{\text{cov}}{:} M}{\Phi \vdash C \overset{\text{cov}}{:} M \mathbin{\&} N} \quad \frac{\Phi \vdash C \overset{\text{cov}}{:} N}{\Phi \vdash C \overset{\text{cov}}{:} M \mathbin{\&} N} \ll\!\&$$

$$\frac{}{\Phi \vdash C \overset{\text{cov}}{:} \bot\!\!\bot} \ll\!\bot\!\!\bot \qquad \frac{\Phi \vdash C \overset{\text{cov}}{:} M \quad \Phi \vdash C \overset{\text{cov}}{:} N}{\Phi \vdash C \overset{\text{cov}}{:} M \oplus N} \ll\!\oplus$$

$$\text{(no rule for } \top\!\!\top\text{)} \qquad \frac{\Phi \Rightarrow c \overset{\text{cov}}{:} M}{\Phi \Rightarrow c \overset{\text{cov}}{:} M \mathbin{\&} N} \quad \frac{\Phi \Rightarrow c \overset{\text{cov}}{:} M}{\Phi \Rightarrow c \overset{\text{cov}}{:} M \mathbin{\&} N}$$

$$\frac{}{\cdot \Rightarrow c \overset{\text{cov}}{:} \bot\!\!\bot} \qquad \frac{\Phi_1 \Rightarrow c \overset{\text{cov}}{:} M \quad \Phi_2 \Rightarrow c \overset{\text{cov}}{:} N}{\Phi_1 \wedge \Phi_2 \Rightarrow c \overset{\text{cov}}{:} M \oplus N}$$

$$\|\top\!\!\top^{\mathbb{C}}\| = \|\bot\!\!\bot_{\mathbb{C}}\| = \mathbb{C} \qquad \|M \mathbin{\&} N\| = \|M \oplus N\| = \mathbb{C} \quad \text{if } \|M\| = \|N\| = \mathbb{C}$$

Figure 13: Negative intersections and unions

Now consider the exclusive-or function encoded in continuation-passing-style, a continuation of type $(\mathbb{B} \otimes \mathbb{B}) \otimes \overset{v}{\neg}\mathbb{B}$:

$$\begin{aligned} xor = \{&\langle\langle t, t\rangle, \mathrm{con}(\overline{u})\rangle \;\mapsto f \rhd \overline{u} \\ \mid &\langle\langle t, f\rangle, \mathrm{con}(\overline{u})\rangle \mapsto t \rhd \overline{u} \\ \mid &\langle\langle f, t\rangle, \mathrm{con}(\overline{u})\rangle \mapsto t \rhd \overline{u} \\ \mid &\langle\langle f, f\rangle, \mathrm{con}(\overline{u})\rangle \mapsto f \rhd \overline{u}\} \end{aligned}$$

As the reader can easily verify, more precise typing ascriptions additionally hold:

$$\begin{array}{ll} xor \overset{\text{cnt}}{:} (\mathbb{T} \otimes \mathbb{T}) \otimes \overset{v}{\neg}\mathbb{F} & xor \overset{\text{cnt}}{:} (\mathbb{T} \otimes \mathbb{F}) \otimes \overset{v}{\neg}\mathbb{T} \\ xor \overset{\text{cnt}}{:} (\mathbb{F} \otimes \mathbb{T}) \otimes \overset{v}{\neg}\mathbb{T} & xor \overset{\text{cnt}}{:} (\mathbb{F} \otimes \mathbb{F}) \otimes \overset{v}{\neg}\mathbb{F} \end{array}$$

Indeed, abbreviating $\overset{v}{\neg}(P \otimes \overset{v}{\neg}Q)$ by $P \overset{v}{\to} Q$, we can give $\mathrm{con}(xor)$ an intersection type:

$$(\mathbb{T} \otimes \mathbb{T}) \overset{v}{\to} \mathbb{F} \;\cap\; (\mathbb{T} \otimes \mathbb{F}) \overset{v}{\to} \mathbb{T} \;\cap\; (\mathbb{F} \otimes \mathbb{T}) \overset{v}{\to} \mathbb{T} \;\cap\; (\mathbb{F} \otimes \mathbb{F}) \overset{v}{\to} \mathbb{F}$$

4.3 The negative (lazy) case

Figure 13 defines negative intersections and unions, dually (of course) to positive unions and intersections. One perhaps surprising implication of this duality is the necessity of a *covalue* restriction for call-by-name. Essentially this reconstructs the fact that in effectful settings, unions may only be eliminated in evaluation position [DP04], and whereas every call-by-value continuation corresponds to an evaluation context, not every call-by-name continuation does. The restriction is required, at least for empty unions, even when the only effect is non-termination. There are no values of type $\bot\!\!\bot^{\vdash}$, but a non-terminating

expression could inhabit the type $\bot\!\!\!\bot$—and so the call-by-name continuation $\{\exp(u) \mapsto \text{fail}\}$, which simply ignores its expression argument and crashes, should not be allowed to have type $\downarrow\bot\!\!\!\bot$, although it can have type $\bot\!\!\!\bot^+$.

On the other hand, intersection introduction does not require a value restriction in call-by-name languages, even effectful ones. In **CU**, this is reflected by the admissibility of the following rule:

$$\frac{\Phi \vdash E \stackrel{\text{exp}}{:} M \quad \Phi \vdash E \stackrel{\text{exp}}{:} N}{\Phi \vdash E \stackrel{\text{exp}}{:} M \mathbin{\&} N}$$

4.4 Subtyping: theory

We purposefully left subtyping out of the above discussion, on the philosophy that intersection and union types—like any other logical type constructors—should be defined solely by their value or covalue introduction rules. Given that starting point, how do we recover a notion of subtyping? The answer should not be too surprising. Already in Section 3.2, we observed that the logically admissible identity principles were witnessed by η-expansion in **CU**, and then added the identity rules $id\!\ll$ and $\gg\!id$ simply to cut short this process of η-expansion. Subtyping is no different.

Let us first demonstrate with some examples. We define the *identity coercion* $id^{\mathbb{B}}_{\overline{u}}$, a continuation which takes a boolean argument and passes it along to the continuation $\overline{u}$:

$$id^{\mathbb{B}}_{\overline{u}} = \{t \mapsto t \rhd \overline{u} \mid f \mapsto f \rhd \overline{u}\}$$

We also define $id^{\neg\mathbb{B}}_{\overline{u}}$, which takes a boolean-continuation as argument, composes it with $id^{\mathbb{B}}$, and passes it along to $\overline{u}$:

$$id^{\neg\mathbb{B}}_{\overline{u}} = \left\{\text{con}(\overline{u}') \mapsto \text{con}(id^{\mathbb{B}}_{\overline{u}'}) \rhd \overline{u}\right\}$$

The reader may easily verify that $\overline{u} \stackrel{\text{cnt}}{:} \mathbb{B} \vdash id^{\mathbb{B}}_{\overline{u}} \stackrel{\text{cnt}}{:} \mathbb{B}$, as well as $\overline{u} \stackrel{\text{cnt}}{:} \mathbb{B} \vdash id^{\mathbb{B}}_{\overline{u}} \stackrel{\text{cnt}}{:} \mathbb{T}$, $\overline{u} \stackrel{\text{cnt}}{:} \mathbb{B} \vdash id^{\mathbb{B}}_{\overline{u}} \stackrel{\text{cnt}}{:} \mathbb{F}$, and $\overline{u} \stackrel{\text{cnt}}{:} \mathbb{T}, \mathbb{F} \vdash id^{\mathbb{B}}_{\overline{u}} \stackrel{\text{cnt}}{:} \mathbb{B}$. Then we also have:[11]

$$\dfrac{\dfrac{\dfrac{\overline{u}' \stackrel{\text{cnt}}{:} \mathbb{B} \vdash id^{\mathbb{B}}_{\overline{u}'} \stackrel{\text{cnt}}{:} \mathbb{T}}{\overline{u}' \stackrel{\text{cnt}}{:} \mathbb{B} \vdash \text{con}(id^{\mathbb{B}}_{\overline{u}'}) \stackrel{\text{val}}{:} \stackrel{v}{\neg}\mathbb{T}} \quad \dfrac{\overline{u}' \stackrel{\text{cnt}}{:} \mathbb{B} \vdash id^{\mathbb{B}}_{\overline{u}'} \stackrel{\text{cnt}}{:} \mathbb{F}}{\overline{u}' \stackrel{\text{cnt}}{:} \mathbb{B} \vdash \text{con}(id^{\mathbb{B}}_{\overline{u}'}) \stackrel{\text{val}}{:} \stackrel{v}{\neg}\mathbb{F}}}{\overline{u}' \stackrel{\text{cnt}}{:} \mathbb{B} \vdash \text{con}(id^{\mathbb{B}}_{\overline{u}'}) \stackrel{\text{val}}{:} \stackrel{v}{\neg}\mathbb{T} \cap \stackrel{v}{\neg}\mathbb{F}}}{\overline{u} \stackrel{\text{cnt}}{:} \stackrel{v}{\neg}\mathbb{T} \cap \stackrel{v}{\neg}\mathbb{F}, \overline{u}' \stackrel{\text{cnt}}{:} \mathbb{B} \vdash \text{con}(id^{\mathbb{B}}_{\overline{u}'}) \rhd \overline{u} \stackrel{\text{stm}}{:} \cdot}}{\overline{u} \stackrel{\text{cnt}}{:} \stackrel{v}{\neg}\mathbb{T} \cap \stackrel{v}{\neg}\mathbb{F} \vdash id^{\neg\mathbb{B}}_{\overline{u}} \stackrel{\text{cnt}}{:} \stackrel{v}{\neg}\mathbb{B}}$$

as well as:

[11] For clarity, here we have removed hypotheses from the context once they are focused upon, since identity coercions use foci linearly.

$$\frac{P \leq \mathbf{Q} \quad \overline{u} \stackrel{\text{cnt}}{:} \mathbf{Q} \in \Phi}{\Phi \vdash \overline{u} \stackrel{\text{cnt}}{:} P}\ sub{\ll} \qquad \frac{u \stackrel{\text{exp}}{:} \mathbf{M} \in \Phi \quad \mathbf{M} \leq N}{\Phi \vdash u \stackrel{\text{exp}}{:} N}\ {\gg}sub$$

where

$P \leq \mathbf{Q}$ iff $\forall v \in \|P\|, \Phi \in (v \stackrel{\text{val}}{:} P)^{-1}\ \exists Q \in \mathbf{Q}, \Phi' \in (v \stackrel{\text{val}}{:} Q)^{-1}.\Phi \vdash \Phi'$

$\mathbf{M} \leq N$ iff $\forall c \in \|N\|, \Phi \in (c \stackrel{\text{cov}}{:} N)^{-1}\ \exists M \in \mathbf{M}, \Phi' \in (c \stackrel{\text{cov}}{:} M)^{-1}.\Phi \vdash \Phi'$

Figure 14: **CU** subtyping

$$\dfrac{\dfrac{\dfrac{\overline{u}' \stackrel{\text{cnt}}{:} \mathbb{T}, \mathbb{F} \vdash id^{\mathbb{B}}_{\overline{u}'} \stackrel{\text{cnt}}{:} \mathbb{B}}{\overline{u}' \stackrel{\text{cnt}}{:} \mathbb{T}, \mathbb{F} \vdash \text{con}(id^{\mathbb{B}}_{\overline{u}'}) \stackrel{\text{val}}{:} \neg\mathbb{B}}}{\overline{u} \stackrel{\text{cnt}}{:} \neg\mathbb{B}, \overline{u}' \stackrel{\text{cnt}}{:} \mathbb{T}, \mathbb{F} \vdash \text{con}(id^{\mathbb{B}}_{\overline{u}'}) \rhd \overline{u} \stackrel{\text{stm}}{:} \cdot}}{\overline{u} \stackrel{\text{cnt}}{:} \neg\mathbb{B} \vdash id^{\neg\mathbb{B}}_{\overline{u}} \stackrel{\text{cnt}}{:} \neg\mathbb{T} \cap \neg\mathbb{F}}$$

From these derivations, the reader may surmise (respectively) that $\neg\mathbb{B} \leq \neg\mathbb{T} \cap \neg\mathbb{F}$ and $\neg\mathbb{T} \cap \neg\mathbb{F} \leq \neg\mathbb{B}$—and indeed that η-expansion now witnesses general subtyping relationships, beyond mere reflexivity.

In essence, this is our *definition* of subtyping. In the positive case, $P \leq Q$ just when $\overline{u} \stackrel{\text{cnt}}{:} Q \vdash id_{\overline{u}} \stackrel{\text{cnt}}{:} P$ for an appropriate identity coercion $id_{\overline{u}}$, while in the negative case, $M \leq N$ when $u \stackrel{\text{exp}}{:} M \vdash id_u \stackrel{\text{exp}}{:} N$. Figure 14 now internalizes this process of building identity coercions, via the subtyping rules $sub{\ll}$ and ${\gg}sub$—replacements for our previous $id{\ll}$ and ${\gg}id$. We use a slightly more convenient but equivalent definition of subtyping, basically expanding what it would mean to type-check identity coercions. The relations $P \leq \mathbf{Q}$ and $\mathbf{M} \leq N$ (where $\mathbf{Q}$ and $\mathbf{M}$ are sets of types) are defined inductively using the inversion operator, referring back to the judgment $\Phi \vdash \Phi'$, which just means that $\Phi \vdash l : \tau$ for all $l : \tau \in \Phi'$, and which can be further expanded by the following proposition:

Proposition. *$\Phi \vdash \Phi'$ iff all of the following hold: (i) $x \stackrel{val}{:} X \in \Phi'$ implies $x \stackrel{val}{:} X \in \Phi$; (ii) $\overline{u} \stackrel{cnt}{:} P \in \Phi'$ implies $\overline{u} \stackrel{cnt}{:} \mathbf{Q} \in \Phi$ and $P \leq \mathbf{Q}$; (iii) $u \stackrel{exp}{:} N \in \Phi'$ implies $u \stackrel{exp}{:} \mathbf{M} \in \Phi$ and $\mathbf{M} \leq N$; (iv) $\overline{x} \stackrel{cov}{:} \overline{X} \in \Phi'$ implies $\overline{x} \stackrel{cov}{:} \overline{X} \in \Phi$.*

Now we can easily prove that subtyping satisfies reflexivity (i.e., the identity principle) and transitivity.

Proposition (Reflexivity). *(i) $P \leq P$; (ii) $N \leq N$; (iii) $\Phi \vdash \Phi$*

Proof. Simultaneously by induction on P, N, and Φ. □

Lemma (Transitivity). *(i) if $P \leq \mathbf{Q}$ and for all $Q \in \mathbf{Q}$, $Q \leq \mathbf{R}$ then $P \leq \mathbf{R}$; (ii) if $\mathbf{M} \leq N$ and for all $M \in \mathbf{M}$, $\mathbf{L} \leq M$, then $\mathbf{L} \leq N$; (iii) if $\Phi \vdash \Phi'$ and $\Phi' \vdash \Phi''$ then $\Phi \vdash \Phi''$.*

Proof. Simultaneously by induction on $\mathbf{Q}$, $\mathbf{M}$, and Φ'. □

4.5 Type safety, cut-elimination, and decidability

Since we designed the new type constructors to satisfy the inversion property, the proofs of type safety and partial cut-elimination from Section 3.3 go through almost completely unchanged for **CU** extended with intersection and union types and subtyping. In particular, those proofs did not rely on the coincidence that $(v \overset{\text{val}}{:} P)^{-1}$ and $(c \overset{\text{cov}}{:} N)^{-1}$ were singleton sets or that contexts contained exactly one type for each bound variable. The main additional burden comes from the use of hypotheses through arbitrary subtyping (rather than merely reflexivity), which necessitates proving a few minor lemmas.

Lemma (Replacement). *If* $\Phi, \Phi_2 \vdash t : \tau$ *and* $\Phi_1 \vdash \Phi_2$ *then* $\Phi, \Phi_1 \vdash t : \tau$.

Proof. By induction on the derivation of $\Phi, \Phi_2 \vdash t : \tau$. The interesting case is when $\Phi, \Phi_2 \vdash \overline{u} \overset{\text{cnt}}{:} P$ is derived by application of $sub{\ll}$ (or analogously ${\gg}sub$) with $\overline{u} \overset{\text{cnt}}{:} \mathbf{Q} \in \Phi_2$, $P \leq \mathbf{Q}$. By assumption, there exists $\overline{u} \overset{\text{cnt}}{:} \mathbf{R} \in \Phi_1$ such that for all $Q \in \mathbf{Q}$, $Q \leq \mathbf{R}$. Then $P \leq \mathbf{R}$ by transitivity, and we can again apply $sub{\ll}$. □

Lemma (Continuation reverse-inclusion). *If* $\Phi \vdash K \overset{cnt}{:} Q$ *for all* $Q \in \mathbf{Q}$ *and* $P \leq \mathbf{Q}$ *then* $\Phi \vdash K \overset{cnt}{:} P$.

Lemma (Expression inclusion). *If* $\Phi \vdash E \overset{exp}{:} M$ *for all* $M \in \mathbf{M}$ *and* $\mathbf{M} \leq N$ *then* $\Phi \vdash E \overset{exp}{:} N$.

Proof. We show the positive case. If $K = \overline{u}$, this is immediate by transitivity. Otherwise, we check $\Phi \vdash K \overset{\text{cnt}}{:} P$ by application of $blur{\ll}$. Let $v \in \|P\|$, $\Phi' \in (v \overset{\text{val}}{:} P)^{-1}$. By definition of $P \leq \mathbf{Q}$, there exists $Q \in \mathbf{Q}$ and $\Phi'' \in (v \overset{\text{val}}{:} Q)^{-1}$ such that $\Phi' \vdash \Phi''$. By inverting $\Phi \vdash K \overset{\text{cnt}}{:} Q$, we have that there is a $(v \mapsto S) \in K$ and that $\Phi, \Phi'' \vdash S \overset{\text{stm}}{:} \cdot$. Then $\Phi, \Phi' \vdash S \overset{\text{stm}}{:} \cdot$ by replacement, and thus $\Phi \vdash K \overset{\text{cnt}}{:} P$. □

Now we can reprove the substitution lemma for the extended type system.

Lemma (Substitution). *If* $\Phi, \Phi' \vdash t : \tau$ *and* $\Phi \vdash \sigma : \Phi'$ *then* $\Phi \vdash \sigma(t) : \tau$.

Proof. The only additional case to consider is when a continuation or expression variable in the domain of σ is used through subtyping, like so:

$$\frac{P \leq \mathbf{Q} \quad \overline{u} \overset{\text{cnt}}{:} \mathbf{Q} \in \Phi'}{\Phi, \Phi' \vdash \overline{u} \overset{\text{cnt}}{:} P}\ sub{\ll}$$

By assumption $\Phi \vdash \sigma(\overline{u}) \overset{\text{cnt}}{:} Q$ for all $Q \in \mathbf{Q}$, hence $\Phi \vdash \sigma(\overline{u}) \overset{\text{cnt}}{:} P$ by the continuation reverse-inclusion lemma. □

Since the proofs of progress, preservation, and type safety from Section 3.3 relied entirely on the inversion property and the substitution lemma, they go through without modification.

Proposition (Progress). *If* S *is a serious cut then there exists an* S' *such that* $S \mapsto S'$.

Lemma (Preservation). *If* $\Phi \vdash S \overset{stm}{:} \cdot$ *and* $S \mapsto S'$ *then* $\Phi \vdash S' \overset{stm}{:} \cdot$.

Corollary (Type safety). *If* S *is a closed well-typed statement, then either* $S = \text{done}$ *or else there is an* S' *such that* $S \mapsto S'$*, and moreover any such* S' *is well-typed.*

Finally, to extend partial cut-elimination, we first show the following:

Lemma (Substitution inclusion). *If* $\Phi \vdash \sigma : \Phi_1$ *and* $\Phi_1 \vdash \Phi_2$ *then* $\Phi \vdash \sigma : \Phi_2$.

Proof. This immediately reduces to continuation reverse-inclusion and expression inclusion. □

Lemma (Value inclusion). *If* $\Phi \vdash V \overset{val}{:} P$ *and* $P \leq \mathbf{Q}$ *then* $\exists Q \in \mathbf{Q}$ *such that* $\Phi \vdash V \overset{val}{:} Q$.

Lemma (Covalue reverse-inclusion). *If* $\Phi \vdash C \overset{cov}{:} N$ *and* $\mathbf{M} \leq N$ *then* $\exists M \in \mathbf{M}$ *such that* $\Phi \vdash C \overset{cov}{:} M$.

Proof. Value inclusion: Let $V = \sigma(v)$. By inversion-reduction on $\Phi \vdash \sigma(v) \overset{\text{val}}{:} P$, there exists $\Phi' \in (v \overset{\text{val}}{:} P)^{-1}$ such that $\Phi \vdash \sigma : \Phi'$. Since $P \leq \mathbf{Q}$, there exists $\Phi'' \in (v \overset{\text{val}}{:} Q)^{-1}$ such that $\Phi' \vdash \Phi''$. Hence $\Phi \vdash \sigma : \Phi''$ by substitution inclusion, and $\Phi \vdash \sigma(v) \overset{\text{val}}{:} Q$ by inversion-expansion. □

Theorem (Partial cut-elimination). *If* $\Phi \vdash t : \tau$ *and* $t \Downarrow t^*$ *then* $\Phi \vdash t^* : \tau$*, and moreover the latter derivation is cut-free.*

Proof. By induction on the derivation of $t \Downarrow t^*$, now with a secondary induction on the derivation of $\Phi \vdash t : \tau$. As in Section 3.3, almost every case is immediate by appealing to the induction hypothesis. For example if $\Phi \vdash V \overset{\text{val}}{:} P \cap Q$ and $V \Downarrow V^*$, then $\Phi \vdash V \overset{\text{val}}{:} P$ and $\Phi \vdash V \overset{\text{val}}{:} Q$ implies $\Phi \vdash V^* \overset{\text{val}}{:} P$ and $\Phi \vdash V^* \overset{\text{val}}{:} Q$, whence $\Phi \vdash V^* \overset{\text{val}}{:} P \cap Q$. The more interesting case is when the typing derivation was obtained by a cut against a use of subtyping:

$$\dfrac{\Phi \vdash V \overset{\text{val}}{:} P \qquad \dfrac{P \leq \mathbf{Q} \quad \overline{u} \overset{\text{cnt}}{:} \mathbf{Q} \in \Phi}{\Phi \vdash \overline{u} \overset{\text{cnt}}{:} P}\ {\ll}sub}{\Phi \vdash V \rhd \overline{u} \overset{\text{stm}}{:} \cdot}\ cut{\gg}$$

Then $\Phi \vdash V^* \overset{\text{val}}{:} P$ by the i.h., but by value inclusion there must be a $Q \in \mathbf{Q}$ such that $\Phi \vdash V^* \overset{\text{val}}{:} Q$. So we can replace the $cut{\gg}$ with a $focus{\gg}$:

$$\dfrac{\Phi \vdash V^* \overset{\text{val}}{:} Q \quad \overline{u} \overset{\text{cnt}}{:} Q \in \Phi}{\Phi \vdash V^* \rhd \overline{u} \overset{\text{stm}}{:} \cdot}\ focus{\gg}$$

□

Finally, we make a few remarks about decidability. Type-checking the *cut-free* fragment of the language is manifestly decidable, since every rule other than $cut{\gg}$ and ${\ll}cut$ satisfies the subformula property and has finitely many premises. For the same reason, subtyping is likewise manifestly decidable.[12] We

[12] And indeed has a fairly simple axiomatization, which I elide here. The trick is to axiomatize the relation between two sets, $\mathbf{A} \leqslant \mathbf{B}$.

$$(P \otimes Q) \cap (P \otimes R) \equiv P \otimes (Q \cap R) \qquad (P \otimes Q) \cup (P \otimes R) \equiv P \otimes (Q \cup R)$$
$$(L \& M) \cap (L \& N) \equiv L \& (M \cap N) \qquad (L \& M) \cup (L \& N) \equiv L \& (M \cup N)$$
$$(P \oplus Q) \cap (P \oplus R) \equiv P \oplus (Q \cap R) \qquad (P \oplus Q) \cup (P \oplus R) \equiv P \oplus (Q \cup R)$$
$$(L \parr M) \cap (L \parr N) \equiv L \parr (M \cap N) \qquad (L \parr M) \cup (L \parr N) \equiv L \parr (M \cup N)$$
$$P \otimes \bot\!\!\!\bot^+ \leq \bot\!\!\!\bot^+ \qquad \top\!\!\!\top_- \leq M \parr \top\!\!\!\top_-$$

Figure 15: Distributivity laws for products and sums

$$\overset{v}{\neg}(P \cup Q) \equiv \overset{v}{\neg} P \cap \overset{v}{\neg} Q \qquad \overset{v}{\neg} P \cup \overset{v}{\neg} Q \leq \overset{v}{\neg}(P \cap Q)$$
$$\overset{n}{\neg}(M \cup N) \leq \overset{n}{\neg} M \cap \overset{n}{\neg} N \qquad \overset{n}{\neg} M \cup \overset{n}{\neg} N \equiv \overset{n}{\neg}(M \cap N)$$
$$\Downarrow(M \cap N) \equiv \downarrow M \cap \downarrow N \qquad \downarrow M \cup \downarrow N \leq \Downarrow(M \cup N)$$
$$\Uparrow(P \cap Q) \leq \uparrow P \cap \uparrow Q \qquad \uparrow P \cup \uparrow Q \equiv \Uparrow(P \cup Q)$$

Figure 16: Distributivity laws for negations and shifts

conjecture, however, that type-checking the full language (with subtyping and cuts) is undecidable, in analogy with the well-known correspondence between typeability in intersection type systems and normalizability. For example, the expression $E = \{\exp(u) \mapsto u \lhd \exp(u)\}$ can be given type $\neg N \cup \neg\neg N$ for any N. The statement $E \lhd \exp(\{\exp(u) \mapsto \text{done}\})$ type-checks, and normalizes in two steps. But the self-application $E \lhd \exp(E)$ does not normalize—and does not check. In any event, we expect that type-checking could be made decidable by the requirement of type annotations on cuts, though the precise form of these annotations is a question for investigation (cf. [DP04] for some of the issues involved).

4.6 Subtyping: examples (and counterexamples)

In addition to a few unsurprising facts—such as that strict pairs are strict ($P \otimes \bot\!\!\!\bot^+ \leq \bot\!\!\!\bot^+$) but lazy pairs are not (in general $N \& \bot\!\!\!\bot^- \not\leq \bot\!\!\!\bot^-$)—the subtyping relationship reveals a surprising amount about the world of **CU**. In this concluding section we investigate some of the most interesting distributivity principles and *non*-principles induced by **CU**'s syntactically-defined subtyping relationship.

We give some basic distributivity laws for products and sums in Figure 15 (strict and lazy versions satisfy most of the same laws), and then for the positive/negative negations and shift operators in Figure 16. We stress that by definition, each one of these laws is witnessed by an identity coercion. For example, corresponding to $\overset{v}{\neg} P \cap \overset{v}{\neg} Q \leq \overset{v}{\neg}(P \cup Q)$ we have the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\overline{u}' \overset{\text{cnt}}{:} P, Q \vdash id_{\overline{u}'} \overset{\text{cnt}}{:} P \cup Q}}{\overline{u}' \overset{\text{cnt}}{:} P, Q \vdash \text{con}(id_{\overline{u}'}) \overset{\text{val}}{:} \overset{v}{\neg}(P \cup Q)}}{\overline{u} \overset{\text{cnt}}{:} \overset{v}{\neg}(P \cup Q), \overline{u}' \overset{\text{cnt}}{:} P, Q \vdash \text{con}(id_{\overline{u}'}) \rhd \overline{u} \overset{\text{stm}}{:} \cdot}}{\overline{u} \overset{\text{cnt}}{:} \overset{v}{\neg}(P \cup Q) \vdash \{\text{con}(\overline{u}') \mapsto \text{con}(id_{\overline{u}'}) \rhd \overline{u}\} \overset{\text{cnt}}{:} \overset{v}{\neg} P \cap \overset{v}{\neg} Q}$$

Where a law is one-sided (e.g., $\overset{n}{\neg}(M \cup N) \leq \overset{n}{\neg} M \cap \overset{n}{\neg} N$) and the converse law is missing ($\overset{n}{\neg} M \cap \overset{n}{\neg} N \leq \overset{n}{\neg}(M \cup N)$), the corresponding identity coercion fails to type-check.

With this bit of data, we can provide logical explanations for many of the anomalies uncovered in the literature on refinement type systems. The fact that $\uparrow P \cap \uparrow Q \not\leq \uparrow(P \cap Q)$ may be seen as another explanation of the value restriction on intersection introduction from [DP00], while $\downarrow(M \cup N) \not\leq \downarrow M \cup \downarrow N$ explains the "evaluation-context restriction" for union elimination in [DP04]. Using the CPS encodings $P \overset{v}{\to} Q := \overset{v}{\neg}(P \otimes \overset{v}{\neg} Q)$, $M \overset{n}{\to} N := \overset{n}{\neg} M \parr N$, we may inspect the thorny issue of distributivity principles for function types. As first observed in [DP00], the standard rule $(A \to B) \cap (A \to C) \leq A \to (B \cap C)$ is not valid under call-by-value:

$$\begin{array}{rcl}
(P \overset{v}{\to} Q) \cap (P \overset{v}{\to} R) & := & \overset{v}{\neg}(P \otimes \overset{v}{\neg} Q) \cap \overset{v}{\neg}(P \otimes \overset{v}{\neg} R) \\
& \equiv & \overset{v}{\neg}((P \otimes \overset{v}{\neg} Q) \cup (P \otimes \overset{v}{\neg} R)) \\
& \equiv & \overset{v}{\neg}(P \otimes (\overset{v}{\neg} Q \cup \overset{v}{\neg} R)) \\
& \not\leq & \overset{v}{\neg}(P \otimes \overset{v}{\neg}(Q \cap R)) \\
& := & P \overset{v}{\to} (Q \cup R)
\end{array}$$

However, it is valid for call-by-name functions:

$$\begin{array}{rcl}
(L \overset{n}{\to} M) \cap (L \overset{n}{\to} N) & := & (\overset{n}{\neg} L \parr M) \cap (\overset{n}{\neg} L \parr N) \\
& \equiv & \overset{n}{\neg} L \parr (M \cap N) \\
& := & L \overset{n}{\to} (M \cap N)
\end{array}$$

And dually, the distributivity rule $(P \overset{v}{\to} R) \cap (Q \overset{v}{\to} R) \leq (P \cup Q) \overset{v}{\to} R$ is valid, but not its negative counterpart.

The restrictions in [DP00] and [DP04] were found to be necessary in order to preserve type safety in call-by-value languages with effects. The fact that these observations have a purely type-theoretic explanation in **CU**—through a subtyping relationship that still validates many interesting distributivity principles—is both surprising and reassuring. But one may still be unsatisfied by this explanation, as it is not self-contained. The subtyping principles that we have shown to be logically invalid lead to safety violations in at least some effectful languages, but do they have actual counterexamples in **CU**? To make pre-

cise what is meant by a "counterexample," we define a notion of orthogonality, adapting that of [Gir01] and [VM04].[13]

Definition (Orthogonality). *We say that* $V \perp K$ *if* $V \rhd K \not\mapsto^*$ fail. *Likewise,* $E \perp C$ *if* $E \lhd C \not\mapsto^*$ fail.

We can use orthogonality to define a "semantic" subtyping relationship.

Definition (Semantic subtyping). *(pos.)* $P \preccurlyeq Q$ *iff for all closed* $V \stackrel{val}{:} P$ *and* $K \stackrel{cnt}{:} Q$*, we have* $V \perp K$*; (neg.)* $M \preccurlyeq N$ *iff for all closed* $E \stackrel{exp}{:} M$ *and* $C \stackrel{cov}{:} N$*, we have* $E \perp C$*.*

This relation satisfies a few basic properties:

Proposition (Reflexivity). $P \preccurlyeq P$ *and* $N \preccurlyeq N$*.*

Proof. By type safety. □

Proposition (Subtyping soundness). $P \leq Q$ *implies* $P \preccurlyeq Q$*, and* $M \leq N$ *implies* $M \preccurlyeq N$*.*

Proof. Both implications follow in two ways: by value inclusion, $P \preccurlyeq Q$ reduces to $Q \preccurlyeq Q$, and by continuation reverse-inclusion it reduces to $P \preccurlyeq P$. □

Semantic subtyping is effectively the largest possible relation we can use while preserving type safety, so it is natural to ask: how close is $\leq$ to $\preccurlyeq$?

This question does not have a straightforward answer, however. The virtue of the syntactic subtyping relationship is that it is defined modularly, in terms of the inversion operator. Yet the semantic subtyping relationship depends heavily on whether we are considering the entire language **CU**, a fragment, or an extension. For instance, since $\mathbb{F} = \mathbb{1}^\perp \oplus 1$ is logically equivalent to "truth", which is irrefutable, there are no closed *proper* (i.e. done-free) continuations of type $\mathbb{F}$. So if we restricted our attention to **CU** without done, we could conclude that $\mathbb{B} \preccurlyeq \mathbb{F}$. Even weaker semantic notions of subtyping, such as value inclusion, suffer from this lack of modularity—there are no closed proper values of type $\neg^v 1$, so should one conclude that $\neg^v 1$ is a subtype of $\mathbb{1}^\perp$? Despite the fact that identity coercions reside purely within the canonical fragment, somehow they are "aware" of the possibility of extension, and hence that only certain inferences are permissible.

Indeed, we can construct counterexamples to some invalid subtyping principles by considering the following extension to **CU**:

Definition (Non-deterministic CU). *For any pair of statements* S_1 *and* S_2*, we can build the statement* $S_1 \parallel S_2$ *(read "*S_1 *or maybe* S_2*"). *$S_1 \parallel S_2$ *is well-typed in any context if both* S_1 *and* S_2 *are well-typed, and it has reduction rules* $S_1 \parallel S_2 \mapsto S_1$ *and* $S_1 \parallel S_2 \mapsto S_2$*.*

[13] The definition in [Gir01] is slightly different: $V \perp K$ if $V \rhd K \Downarrow$ done. We follow [VM04] in encoding orthogonality as type-safety rather than termination.

Our proofs of type safety and partial cut-elimination extend modularly and immediately to non-deterministic **CU**—the reader is welcome to check that these proofs made absolutely no use of the happenstance that evaluation was deterministic, and again, this was part of the point of the methodology. Now, let us begin by giving an expression $E_{bit} \stackrel{\text{exp}}{\vdots} \uparrow\mathbb{B}$ to compute a boolean nondeterministically:

$$E_{bit} = \{\mathrm{con}(\overline{u}) \mapsto (t \rhd \overline{u}) \parallel (f \rhd \overline{u})\}$$

Observe that by the reduction rules, for any continuation K, $E_{bit} \lhd \mathrm{con}(K)$ reduces to either $t \rhd K$ or $f \rhd K$. Now, $\mathbb{B} \equiv \mathbb{T} \cup \mathbb{F}$, so that $\uparrow\mathbb{B} \equiv \uparrow(\mathbb{T} \cup \mathbb{F}) \equiv \uparrow\mathbb{T} \cup \uparrow\mathbb{F}$ by the distributivity laws, and hence $\mathrm{exp}(E_{bit}) : \downarrow(\uparrow\mathbb{T} \cup \uparrow\mathbb{F})$. However, we do not have $\mathrm{exp}(E_{bit}) : \downarrow\uparrow\mathbb{T} \cup \downarrow\uparrow\mathbb{F}$: for that to be the case, we would require that E_{bit} accepts either a $\mathbb{T}$-continuation or an $\mathbb{F}$-continuation as argument—which it does not, since it must be able to pass it either a t or f.

As a value inclusion, then, the principle $\downarrow(M \cup N) \leq \downarrow M \cup \downarrow N$ is definitely invalid in general (here with $M = \uparrow\mathbb{T}$, $N = \uparrow\mathbb{F}$). But can we actually construct a continuation that distinguishes between these two types? We start by defining $K_{eq} \stackrel{\text{cnt}}{\vdots} (\mathbb{T} \otimes \mathbb{T}) \cup (\mathbb{F} \otimes \mathbb{F})$, a continuation that only successfully terminates if its pair of arguments are equal:

$$K_{eq} = \{\langle t, t\rangle \mapsto \mathrm{done} \mid \langle f, f\rangle \mapsto \mathrm{done} \mid \langle t, f\rangle \mapsto \mathrm{fail} \mid \langle f, t\rangle \mapsto \mathrm{fail}\}$$

Now consider the following continuation:

$$K_{test} = \{\mathrm{exp}(u) \mapsto u \lhd \mathrm{con}(\{x \mapsto u \lhd \mathrm{con}(\{y \mapsto \langle x, y\rangle \rhd K_{eq}\})\})\}$$

K_{test} takes an expression computing a boolean as argument, runs it twice, and then passes the pair of results to K_{eq}. Execution is therefore always safe so long as the expression is guaranteed to compute the *same* boolean on both calls. That is, we have $K_{test} \stackrel{\text{cnt}}{\vdots} \downarrow\uparrow\mathbb{T} \cup \downarrow\uparrow\mathbb{F}$. The pair $\mathrm{exp}(E_{bit}) \not\perp K_{test}$ is then our desired counterexample to $\downarrow(\uparrow\mathbb{T} \cup \uparrow\mathbb{F}) \preccurlyeq \downarrow\uparrow\mathbb{T} \cup \downarrow\uparrow\mathbb{F}$, as at least under some executions $\mathrm{exp}(E_{bit}) \rhd K_{test} \Downarrow \mathrm{fail}$.

We could of course dualize this counterexample to obtain one for $\uparrow P \cap \uparrow Q \preccurlyeq \uparrow(P \cap Q)$, though it necessitates use of the perhaps unfamiliar "co-booleans." But intuitively, the reason why $\downarrow(M \cup N) \leq \downarrow M \cup \downarrow N$ is unsound and a "covalue restriction" is needed is that our language allows effectful expressions to be used any number of times—the continuation K_{test} duplicates its argument u, which can compute a different result at each instance. Therefore if we dualize K_{test}, we obtain an expression that duplicates its continuation—just like the original example of the unsoundness of ML with callcc without a value restriction [HL91].

5 Proposed work

I have argued that evaluation order is a logical concept—polarity—and have presented a language, **CU**, in which strict and lazy types may be combined freely—including intersections and unions, and even in the presence of effects.

Still, as I mentioned throughout the text, various theoretical questions about **CU** remain open, and I hope to resolve these in my thesis work. Moreover, I believe that the value of **CU** is not in the language itself, but in the methodology used to construct it: each connective/type constructor was defined purely through a single judgment (i.e., truth/value typing, or falsehood/covalue typing), while the rules for the other judgments were derived through inversion. In particular, I intend to show that the methodology extends to a richer language and type system.

5.1 Open questions about CU

- **Normalization.** At the end of Section 3.3, I side-stepped the question of termination for the unrefined fragment of **CU**, since it was somewhat tangential to the aim of my preliminary work. Termination should be provable by standard techniques—though it would be interesting if the discipline of focusing enables alternative proofs. Known techniques for proving strong normalization of intersection type systems (cf. [Bak04]) could also be applied to **CU** with unions and intersections, though such a result seems even more tangential, since my main interest is in refinement types for general-purpose programming languages.

- **Refinement restriction and type annotations.** The "relaxed" refinement restriction described in Section 4.1 is not completely satisfying. Does it have a logical basis? Is there a reason to adopt a full refinement restriction? Or to abandon the refinement restriction altogether? This question deserves some exploration. Likewise, I should investigate the precise form of type annotations required on cuts (and whether the contextual annotations of [DP04] are applicable).

- **Subtyping completeness.** At the end of Section 4.6, I showed that certain syntactically invalid subtyping schemas are in general operationally unsound, but I did not prove that subtyping completeness holds for nondeterministic **CU**—and in fact this does not appear to be the case. One could ask, is there a natural extension of **CU** for which subtyping completeness holds? Or is incompleteness in some sense fundamental? At least we should be able the question of when $P \preccurlyeq Q$ implies $P \leq Q$ to some simpler criterion on the inhabitants (i.e., values and continuations) of types P and Q.

- **Purity and involutive negation.** A question that is a sort of converse to subtyping completeness: if we restrict **CU** to more "pure" fragments, can we reconstruct a stronger syntactic subtyping relationship? This question is significant, because although we have analyzed call-by-name evaluation with effects, most of the interest in call-by-name comes from Haskell, a mostly pure, actually call-by-need language. The typing and subtyping principles we have derived are therefore conservative—and as we have observed, deal correctly with the presence of non-termination—but are

they overly conservative? Is it possible to logically recover a pure fragment by defining a set of restricted connectives? The analysis of the value and covalue restrictions in Section 4.6 leads naturally to the suggestion of linearity—but that does not seem like it would help at least in so far as subtyping, because identity coercions (in addition to being canonical) are already linear. A more promising direction is to use the involutive negation $(-)^{\perp}$, mentioned in Footnote 4. For example, if implication is CPS encoded as $P \to N := P^{\perp} \parr N$, then it satisfies *both* the distributivity laws $(P \to N) \cap (Q \to N) \leq (P \cup Q) \to N$ and $(P \to M) \cap (P \to N) \leq P \to (M \cap N)$. Is this the (CPS encoding of the) pure function space?

5.2 Extensions of CU

- **Implication.** Although the CPS encodings of call-by-value and call-by-name function spaces are faithful, they are only "full" in the presence of control operators. That is, not every continuation of type $P \otimes {}^{v}\!\neg Q$ corresponds to a call-by-value function (unless it can have control effects), nor every expression of type ${}^{n}\!\neg N \parr M$ to a call-by-name function. A first-class treatment of functions is therefore desirable from not only a practical but also a theoretical point-of-view. I conjecture that such an analysis could also lead (through duality) to a logical reconstruction of *delimited* continuations [DF90].

- **(Co)Inductive types.** This is another extension of obvious practical value. The main theoretical question is how inversion interacts with induction. For example, if inductive types are added to **CU** in a straightforward way, then it seems to make sense for continuations to be defined by arbitrary, infinite sets of pattern branches. Essentially, the form of **CU**'s continuation-typing rule is highly reminiscent of the so called Ω-rule [BFPS81]. Certainly a restriction could be mandated so as to avoid including in the language continuations that may not even be computable—but is there a *logical* way to impose such a restriction? Is it reflected in the definition of subtyping?

- **Polymorphic and existential types.** Note that the "standard" polarizations for quantifiers are universal-negative and existential-positive. In [Gir01, §6.2], Girard identifies several "shocking equalities" for the positive universal quantifier (e.g., $\forall X.P \oplus Q = \forall X.P \oplus \forall X.Q$). I suspect these may become less shocking as statements about polymorphism in a call-by-value language, and in particular I see no barrier to extending my logical analysis of the value restriction on intersection introduction to the usual value restriction on polymorphism in ML. I plan to investigate the full matrix of second-order quantifiers—positive/negative universal/existential—in both explicit and implicit (i.e., refinement type) style.

- **First-order quantifiers, index refinements, and modal types.** The same questions could be asked about the matrix of first-order quanti-

fiers. For one, this would address the interaction of *index refinements* [XP99] with evaluation order. Interestingly, this should also shed some light on *modal types,* given their Kripke reading as quantification over worlds. Since $\Box$ corresponds to universal quantification, its standard interpretation is lazy: for example in [VCHP04], the S5 $\Box$ is given a Curry-Howard reading as a suspended computation that can be run at any node on a network. But what about universally valid *values?* A "$\Box$-like" type operator internalizing such values clearly has its own use, distinct from the standard $\Box$, and indeed Park proposes one in [Par06]. This very much looks like a polarity distinction.

- **Dependent types.** Even more interesting, but more difficult, would be a polarized dissection of the dependent quantifiers Π and Σ. Doing so in **CU** might give (via focusing) a *non*-degenerate interpretation of dependent types for classical logic, in response to [Her05]. It may also shed some light on CLF, an extension of LF with linear connectives that makes essential use of a polarity distinction—but in a slightly different way from that described here [WCPW02].

6 Related Work

This work began as a study of subtyping in refinement type systems, but evolved to incorporate ideas from many other areas of programming languages and logic, as it turned that these were useful in giving a satisfactory answer to the origin of operationally-sensitive typing phenomena. A brief survey of these related fields follows.

Refinement types. Refinement types attempt to increase the expressive power of type systems for practical programming languages so as to capture more invariants, without changing the languages' fundamental character [FP91]. Although this research drew from very old theoretical foundations on intersection types [BCDC83], those foundations were found to be shaky in this more operational setting. Davies and Pfenning found examples showing the necessity of a value restriction on intersection introduction, as well as the unsoundness of the standard subtyping distributivity rule for function types [DP00]. The design of a refinement system with union types [DP04] uncovered further dissonance with prior theoretical studies [BDCD95]. Recent work by Vouillon and Melliès [VM04, MV05] attempts to address some of the issues involved in subtyping intersection and union types in an operational setting. In [Vou04], Vouillon proves that a particular subtyping relationship for union types is sound and complete with respect to *arbitrary* evaluation order. While this result is interesting (and we adopted their elegant machinery of biorthogonality), we see it as somewhat in the wrong spirit from a logical perspective. For although it is possible to give a conservative subtyping relationship that is "agnostic" as to evaluation strategy, it is impossible to formulate evaluation order-agnostic *typing* rules for refinement types (witness the value and covalue restrictions)—and

we have argued that subtyping comes logically after typing.

Duality of computation. Beginning with Filinski's master's thesis [Fil89], a line of work has explored a duality between call-by-value and call-by-name evaluation in the presence of first-class continuations. Filinski was inspired by categorical duality; logical studies of this duality have been done largely in the setting of classical logic, particularly based upon the $\lambda\mu$-calculus [Par92]. For example in [CH00], Curien and Herbelin define a programming language with control operators (the $\overline{\lambda}\mu\tilde{\mu}$-calculus) as a Curry-Howard interpretation of a sequent calculus for implicational classical logic, while in [Wad03], Wadler reformulates this language as a "dual calculus" for propositional classical logic (with implication defined via De Morgan translation). Both papers analyze the duality between call-by-value and call-by-name as one of alternative cut-reduction strategies—without explicitly using polarity to encode these strategies at the level of types. On the other hand, Curien and Herbelin come very close to defining dual focusing calculi as "well-behaved subsyntaxes" of the $\overline{\lambda}\mu\tilde{\mu}$-calculus (cf. [CH00, §5]), as does Shan in a recent extension of Wadler's calculus [Sha05]. In even more recent work, Dyckhoff and Lengrand define a positive focusing system and use the implication fragment to encode call-by-value lambda calculus [DL06]. The key technical concept that seems to be missing from all these works is the notion of inversion, which arises naturally from the judgmental interpretation, and as we have seen is fundamental to the definition of subtyping.

From a more semantic front, Streicher and Reus give a domain-theoretic interpretation of call-by-name based on "negative domains" and continuation semantics [SR98], an extension of earlier work with Lafont [LRS93], which was itself inspired by Girard's polarized deconstruction of classical logic (see below). Perhaps closest in spirit to **CU** is Paul-Blain Levy's call-by-push-value language [Lev01], which maintains separate syntactic categories of "value types" and "computation types" that may in retrospect be seen as a polarity distinction. Recently, Levy has defined a "jumbo" λ-calculus with pattern-matching constructs—essentially for their inversion properties—arguing that this is necessary to obtain the right type isomorphisms [Lev06].

Polarity and focusing. The discovery of linear logic [Gir87] and its crisp symmetries led to some early attempts at explaining strict vs. lazy evaluation within linear logic [Abr93]. Focusing [And92] greatly improved the understanding of these symmetries, and sparked interest in the idea of polarity as a way of taming misbehaved logics [Gir91, Gir93]. It was reconstructed in the "colour protocol" of Danos et al. for studying normalization of proofs [DJS97], and used significantly in the dependent type theory CLF in order to maintain an appropriate notion of canonical forms [WCPW02]. Recently the theory of polarity has been developed in greater depth, both in Ludics [Gir01] and in Olivier Laurent's dissertation [Lau02]. In [Lau05], Laurent pursues an analysis in various ways technically similar to ours, defining dual translations into polarized linear logic and then studying the different type *isomorphisms* induced by η-expansion in a purely logical setting. Subtyping becomes an interesting problem mainly with the addition of "non-logical" type constructors such as unions and intersections, and thus Ludics attempts to reappropriate those by broadening the definition

of "logic."

Ludics. This work profited heavily from the ideas in "Locus Solum" [Gir01]. As the syntax and semantics of **CU** developed, we found that more and more mysterious ludic concepts suddenly made a great deal of sense through the operational intuitions of a programming language. In turn, we were able to incorporate more from Ludics into **CU**. This is not to say that **CU** is the syntax *of* Ludics—indeed, there are substantial differences between Girard's "designs" and programs of **CU**. For one, the use of foci is *affine* in designs, while it is unrestricted in **CU**. Moreover, Ludics does not have designs corresponding directly to values (or covalues), but only "positive actions," which are analogous to statements $V \rhd \overline{u}$. It seems (at least to me) more natural to have the separate value typing judgment—particularly from a programming point of view, but also because it emerges out of the rational reconstruction of polarity à la Dummett, and gives meaning to the inversion operator. In any case we intend to further explore the connection between **CU** and Ludics.

Constructive negation and co-intuitionistic logic. As mentioned in Section 2.2, the notion of constructive refutation is originally due to Nelson [Nel49]. The system defined in that paper (via a realizability interpretation) actually corresponds to the fragment of polarized logic with only the judgments *true* and *false*, and negation defined as the $(-)^{\perp}$ connective. What we call co-intuitionistic logic, wherein truth is defined as *unfalse*, has been explored in various formulations (alternatively called "anti-intuitionistic" or "dual intuitionistic" logic), first by Czermak [Cze77] and Goodman [Goo81]. In a recent paper (and very much in line with our pragmatist interpretation), Shramko argues that co-intuitionistic logic is the "logic of scientific research," tieing it to Popper's conception of science as a continual process of falsification [Shr05]. For an extensive survey of the literature on constructive negation, see [Vak06].

Realizability. There is a suggestive formal resemblance between our Curry-Howard interpretation of focusing and the realizability interpretation of intuitionistic logic [Kle45, Tv88]. Loosely, building an interpretation of logic around the notion of realizers and abstract "methods" of transforming realizers into other realizers, corresponds to Dummett's verificationist meaning-theory, i.e., the idea that we first define canonical proofs, and then justify an arbitrary inference by exhibiting a method of transforming canonical proofs of the premises into a canonical proof of the conclusion—and therefore the judgment $V \stackrel{\text{val}}{:} P$ may be roughly read as "V realizes P". (A similar observation was made in [Gir01, Appendix A, "Realisability"].) The analogy is not quite precise, because our interpretation makes more distinctions than does realizability—for example, the latter defines the realizer for $\neg P$ as a function from realizers of P to realizers of absurdity, whereas we view continuations abstractly as functions from *patterns* of type P to well-typed statements. Still, it seems likely that in the extensive literature on realizability there is something to say about the questions in this thesis. As one example strengthening this connection, the idea that a type system is sound under extensions (related to the question of subtyping completeness) was explored within a realizability setting by Howe (and more recently by Tsukada) under the heading of "computational open-endedness"

[How91, Tsu01].

The logic of logic. While we have traced our judgmental reconstruction of polarity to Dummett's especially insightful account in "The Logical Basis of Metaphysics," many others have considered similar questions about the meaning and the justification of the logical laws. Dummett's work is in part a response to a well-known exchange between Prior and Belnap [Pri67, Bel67]. (In particular, while agreeing with Belnap's analysis of the "Tonk" puzzle, Dummett denies Belnap's contention that if an entire set of introduction and elimination rules satisfies some "harmony" conditions, it is self-justifying [Dum91, p. 251].) Belnap eventually refined and formalized his ideas in the form of Display Logic [Bel82], which has seen a recent resurgence of interest. Part of its appeal is a general cut-admissibility theorem that Belnap proved for any displayed logic satisfying a particular set of (eight) conditions (cf. [DG02]). Likewise, Avron and Lev derive a general cut-elimination theorem for "canonical" sequent calculi, given that each pair of left- and right-rules for a connective satisfies a "coherence" property [AL01]. Both of these approaches have similarities to ours that deserve investigation—but the primary difference may again be traced to Dummett's insight, that defining a connective requires a *bias* towards one side (introduction/right-rules) or the other (elimination/left-rules), thus giving rise to polarity.

Gentzen's idea of biasing towards *introduction* rules and justifying the elimination rules was first explored philosophically by Prawitz [Pra65, Pra74]. As with Belnap's work, Prawitz's has seen a burst of renewed interest, following von Plato's analysis of natural deduction and the framework of "general elimination rules" [Pla01]. While Dummett bases his notion of proof-theoretic justification on Prawitz's, it is a proper extension since he considers the justification of *arbitrary* inferences, rather than merely the elimination rules for a particular connective. This is what makes his analysis so closely related to focusing.

Of course another famous investigation of the justification of the logical laws is Martin-Löf's [ML96]. While his 1983 Siena Lectures lack some of the important ideas of Dummett's earlier analysis,[14] they elucidate the very notion of judgment, which we have taken for granted. And also "Locus Solum," subtitled "From the rules of logic to the logic of rules," resulted from a long meditation on the meanings of the connectives and the nature of logical deduction. We have gone back to Dummett not only because his was an early exposition of the idea of polarity, but so as to tie together these somewhat disparate works.

References

[Abr93] Samson Abramsky. Computational interpretations of linear logic. *Theoretical Computer Science*, 111(1–2):3–57, 1993.

[14] Although apparently Dummett and Martin-Löf were in close correspondence, and discussed the subject of the William James Lectures (Peter Hancock, personal communication).

[AL01] Arnon Avron and Iddo Lev. Canonical propositional Gentzen-type systems. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Automated Reasoning, First International Joint Conference, IJCAR 2001, Siena, Italy, June 18-23, 2001, Proceedings*, volume 2083 of *Lecture Notes in Computer Science*, pages 529–544. Springer, 2001.

[And92] Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):297–347, 1992.

[And01] Jean-Marc Andreoli. Focussing and proof construction. *Annals of Pure and Applied Logic*, 107(1):131–163, 2001.

[Bak04] Steffen van Bakel. Cut-elimination in the strict intersection type assignment system is strongly normalizing, January 2004.

[BCDC83] Henk Barendregt, Mario Coppo, and Mariangiola Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *The Journal of Symbolic Logic*, 48(4):931–940, 1983.

[BDCD95] Franco Barbanera, Mariangiola Dezani-Ciancaglini, and Ugo De'Liguoro. Intersection and union types: syntax and semantics. *Information and Compution*, 119(2):202–230, 1995.

[Bel67] N. D. Belnap. Tonk, plonk and plink. In P. F. Strawson, editor, *Philosophical Logic*, pages 132–137. Oxford University Press, 1967. First appeared in *Analysis* Vol. 22 pp. 130–134 1962.

[Bel82] Nuel D. Belnap, Jr. Display logic. *Journal of Philosophical Logic*, 11(4):375–417, 1982.

[BFPS81] W. Buchholz, S. Feferman, W. Pohlers, and W. Sieg. *Iterated Inductive Definitions and Subsystems of Analysis: Recent Proof-Theoretical Studies*. Springer-Verlag, 1981.

[CH00] Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In *ICFP '00: Proceedings of the fifth ACM SIGPLAN international conference on Functional programming*, pages 233–243. 2000.

[Cze77] J. Czermak. A remark on Gentzen's calculus of sequents. *Notre Dame Journal of Formal Logic*, 18:471–474, 1977.

[DF90] Olivier Danvy and Andrzej Filinski. Abstracting control. In *LFP '90: Proceedings of the 1990 ACM conference on LISP and functional programming*, pages 151–160, New York, NY, USA, 1990. ACM Press.

[DG02] Jeremy E. Dawson and Rajeev Goré. Formalised cut admissibility for display logic. In Victor Carreño, César Muñoz, and Sofiène Tashar, editors, *TPHOLs*, volume 2410 of *Lecture Notes in Computer Science*, pages 131–147. Springer, 2002.

[DJS97] Vincent Danos, Jean-Baptiste Joinet, and Harold Schellinx. A new deconstructive logic: Linear logic. *The Journal of Symbolic Logic*, 62(3):755–807, 1997.

[DL06] Roy Dyckhoff and Stephane Lengrand. LJQ: A strongly focused calculus for intuitionistic logic. In Arnold Beckmann, Ulrich Berger, Benedikt Löwe, and John V. Tucker, editors, *CiE*, volume 3988 of *Lecture Notes in Computer Science*, pages 173–185. Springer, 2006.

[DP00] Rowan Davies and Frank Pfenning. Intersection types and computational effects. In *ICFP '00: Proceedings of the fifth ACM SIGPLAN international conference on Functional programming*, pages 198–208, 2000.

[DP04] Joshua Dunfield and Frank Pfenning. Tridirectional typechecking. In *POPL '04: Proceedings of the 31st ACM Conference on Principles of Programming Languages*, pages 281–292, 2004.

[Dum91] Michael Dummett. *The Logical Basis of Metaphysics*. The William James Lectures, 1976. Harvard University Press, Cambridge, Massachusetts, 1991.

[Fil89] Andrzej Filinski. Declarative continuations and categorical duality. Master's thesis, University of Copenhagen, 1989. Computer Science Department.

[FP91] Tim Freeman and Frank Pfenning. Refinement types for ML. In *PLDI '91: Proceedings of the ACM SIGPLAN 1991 conference on Programming language design and implementation*, pages 268–277, 1991.

[Gen35] Gerhard Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935. English translation in M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131, North-Holland, 1969.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987.

[Gir91] Jean-Yves Girard. A new constructive logic: Classical logic. *Mathematical Structures in Computer Science*, 1:255–296, 1991.

[Gir93] Jean-Yves Girard. On the unity of logic. *Annals of pure and applied logic*, 59(3):201–217, 1993.

[Gir01] Jean-Yves Girard. Locus solum: From the rules of logic to the logic of rules. *Mathematical Structures in Computer Science*, 11(3):301–506, 2001.

[Goo81] Nelson D. Goodman. The logic of contradiction. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 27(2):119–126, 1981.

[Gri90] Timothy G. Griffin. The formulae-as-types notion of control. In *POPL '90: Proceedings of the 17th ACM Conference on Principles of Programming Languages*, pages 47–57. 1990.

[Her05] Hugo Herbelin. On the degeneracy of sigma-types in presence of computational classical logic. In Pawel Urzyczyn, editor, *TLCA '05: Proceedings of the Seventh International Conference on Typed Lambda Calculi and Applications*, volume 3461 of *Lecture Notes in Computer Science*, pages 209–220. Springer, 2005.

[HL91] Bob Harper and Mark Lillibridge. ML with callcc is unsound, 1991. Post to TYPES mailing list, July 8, 1991.

[How91] Douglas J. Howe. On computational open-endedness in martin-Löf's type theory. In *LICS*, pages 162–172. IEEE Computer Society, 1991.

[Kle45] S.C. Kleene. On the interpretation of intuitionistic number theory. *Journal of Symbolic Logic*, 10:109–124, 1945.

[Lau02] Olivier Laurent. *Etude de la polarisation en logique*. Thèse de doctorat, Université Aix-Marseille II, March 2002.

[Lau05] Olivier Laurent. Classical isomorphisms of types. *Mathematical Structures in Computer Science*, 15(5):969–1004, October 2005.

[Lev01] Paul B. Levy. *Call-by-push-value*. PhD thesis, Queen Mary, University of London, 2001.

[Lev06] Paul B. Levy. Jumbo λ-calculus. In *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming, Venice, 2006*, volume 4052 of *Lecture Notes in Computer Science*, 2006.

[LRS93] Yves Lafont, Bernhard Reus, and Thomas Streicher. Continuation semantics or expressing implication by negation. Technical Report 93-21, University of Munich, 1993.

[ML96] Per Martin-Löf. On the meanings of the logical constants and the justifications of the logical laws. *Nordic Journal of Philosophical Logic*, 1(1):11–60, 1996.

[MTHM97] Robin Milner, Mads Tofte, Robert Harper, and David MacQueen. *The Definition of Standard ML,* Revised edition. MIT Press, 1997.

[MV05] Paul-André Melliès and Jérôme Vouillon. Recursive polymorphic types and parametricity in an operational framework. In *LICS '05: Proceedings of the 20th annual IEEE symposium on Logic in Computer Science*, pages 82–91, 2005.

[Nel49] David Nelson. Constructible falsity. *Journal of Symbolic Logic*, 14(1):16–26, 1949.

[Par92] Michel Parigot. $\lambda\mu$-calculus: An algorithmic interpretation of classical natural deduction. In *LPAR '92: Proceedings of the International Conference on Logic Programming and Automated Reasoning*, pages 190–201, London, UK, 1992. Springer-Verlag.

[Par06] Sungwoo Park. A modal language for the safety of mobile values. In Naoki Kobayashi, editor, *Programming Languages and Systems, 4th Asian Symposium, APLAS 2006, Sydney, Australia, November 8-10, 2006, Proceedings*, volume 4279 of *Lecture Notes in Computer Science*, pages 217–233. Springer, 2006.

[PD01] Frank Pfenning and Rowan Davies. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science*, 11(4):511–540, 2001.

[Pla01] Jan von Plato. Natural deduction with general elimination rules. *Archive for Mathematical Logic*, 40(7), oct 2001.

[Pra65] Dag Prawitz. *Natural Deduction: A Proof Theoretical Study.* Almquist and Wiksell, Stockholm, 1965.

[Pra74] Dag Prawitz. On the idea of a general proof theory. *Synthese*, 27:63–77, 1974.

[Pri67] A. N. Prior. The runabout inference ticket. In P. F. Strawson, editor, *Philosophical Logic*, pages 129–131. Oxford University Press, 1967. First appeared in *Analysis* Vol. 21 pp. 38–39 1960.

[Rey72] John C. Reynolds. Definitional interpreters for higher-order programming languages. In *ACM '72: Proceedings of the ACM annual conference*, pages 717–740, 1972.

[Sha05] Chung-chieh Shan. A computational interpretation of classical S4 modal logic. In *IMLA '05: Intuitionistic Modal Logics and Applications Workshop*, 2005.

[Shr05] Yaroslav Shramko. Dual intuitionistic logic and a variety of negations: The logic of scientific research. *Studia Logica*, 80(2-3):347–367, sep 2005.

[SR98] Thomas Streicher and Bernhard Reus. Classical logic, continuation semantics and abstract machines. *Journal of Functional Programming*, 8(6):543–572, November 1998.

[Tsu01] Yasuyuki Tsukada. Martin-Löf's type theory as an open-ended framework. *Int. Journal of Foundations of Computer Science*, 12(1):31–67, 2001.

[Tv88] A. S. Troelstra and D. van Dalen. *Constructivism in Mathematics. An Introduction.*, volume 121 of *Studies in Logic and the Foundations Mathematics*. North-Holland, Amsterdam, 1988.

[Vak06] Dimiter Vakarelov. Non-classical negation in the works of Helena Rasiowa and their impact on the theory of negation. *Studia Logica*, 84(1):105–127, 2006.

[VCHP04] Tom Murphy VII, Karl Crary, Robert Harper, and Frank Pfenning. A symmetric modal lambda calculus for distributed computing. In *LICS*, pages 286–295. IEEE Computer Society, 2004.

[VM04] Jérôme Vouillon and Paul-André Melliès. Semantic types: a fresh look at the ideal model for types. In *POPL '04: Proceedings of the 31st ACM Conference on Principles of Programming Languages*, pages 52–63, 2004.

[Vou04] Jérôme Vouillon. Subtyping union types. In *CSL: 18th Workshop on Computer Science Logic*. LNCS, Springer-Verlag, 2004.

[Wad03] Philip Wadler. Call-by-value is dual to call-by-name. In *ICFP '03: Proceedings of the eighth ACM SIGPLAN International Conference on Functional Programming*, pages 189–201, 2003.

[WCPW02] Kevin Watkins, Iliano Cervesato, Frank Pfenning, and David Walker. A concurrent logical framework I: Judgments and properties. Technical Report CMU-CS-02-101, Department of Computer Science, Carnegie Mellon University, 2002. Revised May 2003.

[Wri95] Andrew K. Wright. Simple imperative polymorphism. *Lisp and Symbolic Computation*, 8(4):343–355, 1995.

[XP99] Hongwei Xi and Frank Pfenning. Dependent types in practical programming. In *POPL '99: Proceedings of the 26th ACM Conference on Principles of Programming Languages*, pages 214–227, New York, NY, USA, 1999. ACM Press.