Adjoint Natural Deduction

Junyoung Jang
McGill University, Montreal, Canada

Sophia Roshal
Carnegie Mellon University, Pittsburgh, USA

Frank Pfenning
Carnegie Mellon University, Pittsburgh, USA

Brigitte Pientka
McGill University, Montreal, Canada

Abstract

Adjoint logic is a general approach to combining multiple logics with different structural properties, including linear, affine, strict, and (ordinary) intuitionistic logics, where each proposition has an intrinsic mode of truth. It has been defined in the form of a sequent calculus because the central concept of independence is most clearly understood in this form, and because it permits a proof of cut elimination following standard techniques.

In this paper we present a natural deduction formulation of adjoint logic and show how it is related to the sequent calculus. As a consequence, every provable proposition has a verification (sometimes called a long normal form). We also give a computational interpretation of adjoint logic in the form of a functional language and prove properties of computations that derive from the structure of modes, including freedom from garbage (for modes without weakening and contraction), strictness (for modes disallowing weakening), and erasure (based on a preorder between modes). Finally, we present a surprisingly subtle algorithm for type checking.

2012 ACM Subject Classification Theory of computation → Proof theory; Theory of computation → Linear logic

Keywords and phrases Substructural Logic, Type Systems, Functional Programming

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.15

Related Version *Full Version*: `https://arxiv.org/abs/2402.01428`

Funding *Junyoung Jang*: Fonds de recherche du Québec – Nature et Technologies (FRQNT)
Brigitte Pientka: Natural Sciences and Engineering Research Council of Canada (NSERC) and Fonds de recherche du Québec – Nature et Technologies (FRQNT)

1 Introduction

A *substructural logic* provides fine control over the use of assumptions during reasoning. It usually does so by denying the general sequent calculus rules of *contraction* (which permits an antecedent to be used more than once) and *weakening* (which permits an antecedent not to be used). Instead, these rules become available only for antecedents of the form $!A$. Ever since the inception of linear logic [23], researchers have found applications in programming languages, for example, to avoid garbage collection [24], soundness of imperative update [53], the chemical abstract machine [2], and session-typed communication [12, 54], to name just a few.

Besides linear logic, there are other substructural logics and type systems of interest. For example, *affine logic* denies general contraction but allows weakening and is the basis for the type system of Alms [51] (an affine functional language) and Rust [50] (an imperative language aimed at systems programming).

If we deny weakening but accept contraction we obtain *strict logic* (a variant of *relevance logic*) where every assumption must be used at least once. On the programming language side, this corresponds to *strictness*, which allows optimizations in otherwise nonstrict functional languages such as Haskell [39]. Interestingly, Church's original λI calculus [18] was also strict in this sense.

The question arises how we can combine such features, both in logics and in type systems. Recently, this question has been tackled through graded or quantitative type systems (see for example, [37, 5, 38, 17, 55, 1]). The essential idea is to track and reason explicitly about the usage of a given assumption through grades. This provides very fine-grained control and allows us to, for example, model linear, strict, and unrestricted usage of assumptions through graded modalities. In this paper, we pursue an alternative, taking a proof-theoretic view with the goal of building a computational interpretation. There are three possible options that emerge from existing proof-theoretic explorations that could serve as a foundation of such a computational interpretation. The first one is by *embedding*. For example, we can embed (structural) intuitionistic logic in linear logic writing $!A \multimap B$ for $A \rightarrow B$. Similarly, we can embed affine logic in linear logic by mapping hypotheses A to $A \mathbin{\&} \mathbf{1}$ so they do not need to be used. The difficulties with such embeddings is that, often, they neither respect proof search properties such as focusing [4] nor do they achieve a desired computational interpretation.

A second approach is taken by *subexponential linear logic* [20, 41, 30] that defines multiple subexponential modalities $!^m A$, where each *mode* m has a specific set of structural properties. As in linear logic, all inferences are carried out on linear formulas, so while it resolves some of the issues with embeddings, it still requires frequent movement into the linear layer using explicit subexponentials.

We pursue a third approach, pioneered by Benton [7] who symmetrically combined (structural) intuitionistic logic with (purely) linear intuitionistic logic. He employs two adjoint modalities that switch between the two layers and works out the proof theoretic and categorical semantics. This approach has the advantage that one can natively reason and compute within the individual logics, so we preserve not only provability but the fine structure of proofs and proof reduction from each component. This has been generalized in prior work [49, 46] by incorporating from subexponential linear logic the idea to have a preorder between modes $m \geq k$ that must be compatible with the structural properties of m and k (explained in more detail in Section 2). This means we can now also model intuitionistic S4 [42] and lax logic [9], representing comonadic and monadic programming, respectively. We hence arrive at a unifying calculus firmly rooted in proof theory that is more general than previous graded modal type systems in that we can *construct* monads as well as comonads. We will briefly address dependently typed variations of the adjoint approach in Section 7.

Most substructural logics and many substructural type systems are most clearly formulated as sequent calculi. However, natural deduction has not only an important foundational role [22, 44, 21], it also has provided a simple and elegant notation for functional programs through the Curry-Howard correspondence [27]. We therefore develop a system of natural deduction for adjoint logic that, in a strong sense, corresponds to the original sequent formulation. It turns out to be surprisingly subtle because we have to manage not only the substructural properties that may be permitted or not, but also respect the preorder between modes. We show that our calculus satisfies some expected properties like *substitution* and has a natural notion of *verification* that corresponds to proofs in long normal form, satisfying a subformula property.

In order to illustrate computational properties, we give an abstract machine and show the consequences of the mode structure: freedom from garbage for linear modes (that is, modes admitting neither weakening nor contraction), strictness for modes that do not admit weakening, and erasure for modes that a final value may not depend on, based on the preorder of modes. We close with an algorithmic type checker for our language which, again, is surprisingly subtle.

2 Adjoint Sequent Calculus

We briefly review the adjoint sequent calculus from [46]. We start with a standard set of possibly substructural propositions, indexing each with a *mode of truth*, denoted by m, k, n, r. Propositions are perhaps best understood by using their linear meaning as a guide, so we uniformly use the notation of linear logic. Also, for programming convenience, we generalize the usual binary and nullary disjunction ($A \oplus B$ and $\mathbf{0}$) and conjunction ($A \mathbin{\&} B$ and $\top$) by using labeled disjunction $\oplus\{\ell : A_m^\ell\}_{\ell \in L}$ and conjunction $\&\{\ell : A_m^\ell\}_{\ell \in L}$. From the linear logical perspective, these are *internal* and *external* choice, respectively; from the programming perspective they are *sums* and *products*. We write P_m for atomic propositions of mode m.

$$
\begin{array}{llll}
\text{Propositions} & A_m, B_m & ::= & P_m \mid A_m \multimap B_m \mid \&\{\ell : A_m^\ell\}_{\ell \in L} \mid {\uparrow}_k^m A_k \quad \text{(negative)} \\
 & & & \mid A_m \otimes B_m \mid \mathbf{1}_m \mid \oplus\{\ell : A_m^\ell\}_{\ell \in L} \mid {\downarrow}_m^n A_n \quad \text{(positive)} \\
\text{Contexts} & \Gamma & ::= & \cdot \mid \Gamma, x : A_m \quad \text{(unordered)}
\end{array}
$$

Each mode m comes with a set $\sigma(m) \subseteq \{\mathsf{W}, \mathsf{C}\}$ of *structural properties*, where W stands for weakening and C stands for contraction. We further have a preorder $m \geq r$ that specifies that a proof of the succedent C_r may depend on an antecedent A_m. This is enforced using the presupposition that in a sequent $\Gamma \vdash C_r$, every antecedent A_m in Γ must satisfy $m \geq r$, written as $\Gamma \geq r$. We have the additional stipulation of *monotonicity*, namely that $m \geq k$ implies $\sigma(m) \supseteq \sigma(k)$. This is required for cut elimination to hold. Furthermore, we presuppose that in ${\uparrow}_k^m A_k$ we have $m \geq k$ and for ${\downarrow}_m^n A_n$ we have $n \geq m$. Also, contexts may not have any repeated variables and we will implicitly apply variable renaming to maintain this presupposition. Finally, we abbreviate $\cdot, x : A$ as just $x : A$.

In preparation for natural deduction, instead of explicit rules of weakening and contraction (see [46] for such a system) we have a context merge operation $\Gamma_1 \mathbin{;} \Gamma_2$. Since, as usual in the sequent calculus, we read the rules bottom-up, it actually describes a nondeterministic split of the context that is pervasive in the presentations of linear logic [4].

$$
\begin{array}{rclcll}
(\Gamma_1, x : A_m) & ; & (\Gamma_2, x : A_m) & = & (\Gamma_1 ; \Gamma_2), x : A_m & \text{provided } \mathsf{C} \in \sigma(m) \\
(\Gamma_1, x : A_m) & ; & \Gamma_2 & = & (\Gamma_1 ; \Gamma_2), x : A_m & \text{provided } x \notin \mathrm{dom}(\Gamma_2) \\
\Gamma_1 & ; & (\Gamma_2, x : A_m) & = & (\Gamma_1 ; \Gamma_2), x : A_m & \text{provided } x \notin \mathrm{dom}(\Gamma_1) \\
(\cdot) & ; & \Gamma_2 & = & \Gamma_2 & \\
\Gamma_1 & ; & (\cdot) & = & \Gamma_1 &
\end{array}
$$

Note that the context merge is a partial operation, which prevents, for example, the use of an antecedent without contraction in both premises of the $\otimes R$ rule.

The complete set of rules can be found in Figure 1. In the rules, we write Γ_{W} for a context in which weakening can be applied to every antecedent, that is, $\mathsf{W} \in \sigma(m)$ for every antecedent $x : A_m$. Also, as is often the case in presentations of the sequent calculus, we omit explicit variable names that tag antecedents. We only discuss the rules for ${\downarrow}_m^n A_n$ because they illustrate the combined reasoning about structural properties and modes.

$$\frac{}{\Gamma_{\mathsf{W}}\,;A_m \vdash A_m}\ \mathsf{id} \qquad \frac{\Gamma \geq m \geq r \quad \Gamma \vdash A_m \quad \Gamma', A_m \vdash C_r}{\Gamma\,;\Gamma' \vdash C_r}\ \mathsf{cut}$$

$$\frac{\Gamma, A_m \vdash B_m}{\Gamma \vdash A_m \multimap B_m}\ {\multimap}R \qquad \frac{\Gamma \geq m \quad \Gamma \vdash A_m \quad \Gamma', B_m \vdash C_r}{\Gamma\,;\Gamma'\,;A_m \multimap B_m \vdash C_r}\ {\multimap}L$$

$$\frac{\Gamma \vdash A_m^\ell \quad (\forall \ell \in L)}{\Gamma \vdash \&\{\ell : A_m^\ell\}_{\ell \in L}}\ \&R \qquad \frac{\Gamma, A_m^\ell \vdash C_r \quad (\ell \in L)}{\Gamma\,;\&\{\ell : A_m^\ell\}_{\ell \in L} \vdash C_r}\ \&L$$

$$\frac{\Gamma \vdash A_k}{\Gamma \vdash {\uparrow_k^m} A_k}\ {\uparrow}R \qquad \frac{k \geq r \quad \Gamma, A_k \vdash C_r}{\Gamma\,;{\uparrow_k^m} A_k \vdash C_r}\ {\uparrow}L$$

$$\frac{\Gamma \vdash A_m \quad \Gamma' \vdash B_m}{\Gamma\,;\Gamma' \vdash A_m \otimes B_m}\ {\otimes}R \qquad \frac{\Gamma, A_m, B_m \vdash C_r}{\Gamma\,;A_m \otimes B_m \vdash C_r}\ {\otimes}L$$

$$\frac{}{\Gamma_{\mathsf{W}} \vdash \mathbf{1}_m}\ \mathbf{1}R \qquad \frac{\Gamma \vdash C_r}{\Gamma\,;\mathbf{1}_m \vdash C_r}\ \mathbf{1}L$$

$$\frac{\Gamma \vdash A_m^\ell \quad (\ell \in L)}{\Gamma \vdash \oplus\{\ell : A_m^\ell\}_{\ell \in L}}\ {\oplus}R \qquad \frac{\Gamma, A_m^\ell \vdash C_r \quad (\forall \ell \in L)}{\Gamma\,;\oplus\{\ell : A_m^\ell\}_{\ell \in L} \vdash C_r}\ {\oplus}L$$

$$\frac{\Gamma' \geq n \quad \Gamma' \vdash A_n}{\Gamma_{\mathsf{W}}\,;\Gamma' \vdash {\downarrow_m^n} A_n}\ {\downarrow}R \qquad \frac{\Gamma, A_n \vdash C_r}{\Gamma\,;{\downarrow_m^n} A_n \vdash C_r}\ {\downarrow}L$$

Figure 1 Implicit Adjoint Sequent Calculus.

First, the ${\downarrow}R$ rule.

$$\frac{\Gamma' \geq n \quad \Gamma' \vdash A_n}{\Gamma_{\mathsf{W}}\,;\Gamma' \vdash {\downarrow_m^n} A_n}\ {\downarrow}R$$

Because we presuppose the conclusion is well-formed, we know $\Gamma_{\mathsf{W}}\,;\Gamma' \geq m$ since ${\downarrow_m^n} A_n$ has mode m. Again, by presupposition $n \geq m$ and we have to explicitly check that $\Gamma' \geq n$ because it doesn't follow from knowing that $\Gamma_{\mathsf{W}}\,;\Gamma' \geq m$. There may be some antecedents A_k in the conclusion such that $k \not\geq n$. If the mode k admits weakening, we can sort them into Γ_{W}. If it does not, then the rule is simply not applicable.

On to the ${\downarrow}L$ rule:

$$\frac{\Gamma, A_n \vdash C_r}{\Gamma\,;{\downarrow_m^n} A_n \vdash C_r}\ {\downarrow}L$$

By presupposition on the conclusion, we know $\Gamma\,;{\downarrow_m^n} A_n \geq r$ which means that $\Gamma \geq r$ and $m \geq r$. Since $n \geq m$ we have $n \geq r$ by transitivity, so $\Gamma, A_n \geq r$ and we do not need any explicit check. The formulation of the antecedents in the conclusion $\Gamma\,;{\downarrow_m^n} A_n$ means that if mode m admits contraction, then the antecedent ${\downarrow_m^n} A_n$ may also occur in Γ, that is, it may be preserved by the rule. If m does not admit contraction, this occurrence of ${\downarrow_m^n} A_n$ is not carried over to the premise.

This implicit sequent calculus satisfies the expected theorems, due to [49, 46] and, most closely reflecting the precise form of our formulation, [45]. They follow standard patterns, modulated by the substructural properties and the preorder on modes.

▶ **Theorem 1** (Admissibility of Weakening and Contraction). *The following are admissible:*

$$\frac{\Gamma_{\mathsf{W}} \geq m \quad \Gamma \vdash A_m}{\Gamma_{\mathsf{W}}\,;\Gamma \vdash A_m}\ \mathsf{weaken} \qquad \frac{\mathsf{C} \in \sigma(m) \quad \Gamma, A_m, A_m \vdash C_r}{\Gamma, A_m \vdash C_r}\ \mathsf{contract}$$

▶ **Theorem 2** (Admissibility of Cut and Identity).

(i) *In the system without cut, cut is admissible.*

(ii) *In the system with identity restricted to atoms P_m, the general identity is admissible.*

We call a proof *cut-free* if it does not contain cut and *long* if the identity is restricted to atomic propositions P. It is an immediate consequence of Theorem 2 that every derivable sequent has a long cut-free proof. The subformula property of cut-free proofs directly implies that a cut-free proof of a sequent $\Gamma_m \vdash A_m$ where all subformulas are of mode m is directly a proof in the logic captured by the mode m. Moreover, an arbitrary proof can be transformed into one of this form by cut elimination. These strong *conservative extension* properties are a hallmark of adjoint logic.

Since our main interest lies in natural deduction, we consider only three examples.

▶ **Example 3** (G3). We obtain the standard sequent calculus G3 [31] for intutionistic logic with a single mode U. All side conditions are automatically satisfied since $\mathsf{U} \geq \mathsf{U}$.

▶ **Example 4** (LNL and DILL). By specializing the rules to two modes, U and L with the order $\mathsf{U} > \mathsf{L}$, we obtain a minor variant of Linear/Non Linear Logic (LNL) in its *parsimonious presentation* [8]. Our notation is $FX = {\downarrow}_{\mathsf{L}}^{\mathsf{U}} X$ and $GA = {\uparrow}_{\mathsf{L}}^{\mathsf{U}} A$. Significant here is that we do not just model provability, but the exact structure of proofs except that our structural rules remain implicit.

We obtain the sequent calculus formulation of dual intuitionistic linear logic (DILL) [6, 15] by restricting the formulas of mode U so that they only contain ${\uparrow}_{\mathsf{L}}^{\mathsf{U}} A_{\mathsf{L}}$. In this version we have $!A = {\downarrow}_{\mathsf{L}}^{\mathsf{U}} {\uparrow}_{\mathsf{L}}^{\mathsf{U}} A$. Again, the rules of dual intuitionistic linear logic are modeled precisely.

▶ **Example 5** (Intuitionistic Subexponential Linear Logic). Subexponential linear logic [40, 41] also uses a preorder of modes, each of which permits specific structural rules. We obtain a formulation of *intuitionistic subexponential linear logic* by adding a new distinguished mode L with $m \geq \mathsf{L}$ for all given subexponential modes m, retaining all the other relations. We further restrict all modes m except for L to contain only ${\uparrow}_{\mathsf{L}}^{m} A_{\mathsf{L}}$, forcing all logical inferences to take place at mode L.

Compared to [16] our system does not contain $?A$ and is not focused; compared to [29], our base logic is linear rather than ordered. Also, all of our structural rules are implicit.

3 Adjoint Natural Deduction

Substructural *sequent calculi* have recently found interesting computational interpretations [12, 54, 13, 43, 48], including adjoint logic [47]. In this paper, we look instead at *functional* interpretations, which are most closely related to *natural deduction*. Some guide is provided by natural deduction systems for *linear logic* (see, for example, [2, 10, 52]), but already they are not entirely straightforward. For example, some of these calculi do not satisfy subject reduction. The interplay between modes and substructural properties creates some further complications. The closest blueprint to follow is probably Benton's [8], but his system does not exhibit the full generality of adjoint logic and is also not quite "parsimonious" in the sense of the LNL sequent calculus.

In the interest of economy, we present the calculus with proof terms and two bidirectional typing judgments, $\Delta \vdash e \Longleftarrow A_m$ (expression e checks against A_m) and $\Delta \vdash s \Longrightarrow A_m$ (expression s *synthesizes* A_m). The syntax for expressions is summarized in Figure 2. The bidirectional nature will allow us to establish a precise relationship to the sequent calculus (Section 4), but it does not immediately yield a type checking algorithm since the context merge operation is highly nondeterministic when used to split contexts. An algorithmic system can be found in Section 6.

We obtain the vanilla typing judgment by replacing both checking and synthesis judgments with $\Delta \vdash e : A$, dropping the rules $\Rightarrow/\Leftarrow$ and $\Leftarrow/\Rightarrow$, and removing the syntactic form $(e : A_m)$. We further obtain a pure natural deduction system by removing the proof terms, although uses of the hypothesis rule then need to be annotated with variables in order to avoid any ambiguities.

Checkable Exps.

$$
\begin{array}{rcll}
e & ::= & \lambda x.\, e & (\multimap) \\
 & \mid & \{\ell \Rightarrow e_\ell\}_{\ell \in L} & (\&) \\
 & \mid & \mathbf{susp}\ e & (\uparrow) \\
 & \mid & (e_1, e_2) & (\otimes) \\
 & \mid & (\,) & (\mathbf{1}) \\
 & \mid & \ell(e) & (\oplus) \\
 & \mid & \mathbf{down}\ e & (\downarrow) \\
 & \mid & \mathbf{match}\ s\ M & \\
 & \mid & s &
\end{array}
$$

Synthesizable Exps.

$$
\begin{array}{rcll}
s & ::= & x & \\
 & \mid & s\ e & (\multimap) \\
 & \mid & s.\ell & (\&) \\
 & \mid & \mathbf{force}\ s & (\uparrow) \\
 & \mid & (e : A_m) &
\end{array}
$$

Matches

$$
\begin{array}{rcll}
M & ::= & (x_1, x_2) \Rightarrow e' & (\otimes) \\
 & \mid & (\,) \Rightarrow e' & (\mathbf{1}) \\
 & \mid & (\ell(x) \Rightarrow e_\ell)_{\ell \in L} & (\oplus) \\
 & \mid & \mathbf{down}\ x \Rightarrow e' & (\downarrow)
\end{array}
$$

Figure 2 Expressions for Bidirectional Natural Deduction.

The rules maintain a few important invariants, particularly *independence*:

(i) $\Delta \vdash e \Longleftarrow A_m$ presupposes $\Delta \geq m$ **(ii)** $\Delta \vdash s \Longrightarrow A_m$ presupposes $\Delta \geq m$

This is somewhat surprising because we think of the synthesis judgment $s \Longrightarrow A_m$ as proceeding top-down rather than bottom-up. Indeed, there are other choices with dependence and structural properties being checked in different places. We picked this particular form because we want general typing $e : A_m$ to arise from collapsing the checking/synthesis distinction. This means that the two rules $\Rightarrow/\Leftarrow$ and $\Leftarrow/\Rightarrow$ should have no conditions because those would disappear. The algorithmic system in Section 6 checks the conditions in different places.

As an example of interesting rules we revisit $\downarrow^n_m A_n$ (where $n \geq m$ is presupposed). The introduction rule of natural deduction mirrors the right rule of the sequent calculus, which is the case throughout.

$$
\frac{\Gamma' \geq n \quad \Gamma' \vdash A_n}{\Gamma_{\mathsf{W}}\,;\Gamma' \vdash \downarrow^n_m A_n}\downarrow R
\qquad\qquad
\frac{\Delta' \geq n \quad \Delta' \vdash e \Longleftarrow A_n}{\Delta_{\mathsf{W}}\,;\Delta' \vdash \mathbf{down}\ e \Longleftarrow \downarrow^n_m A_n}\downarrow I
$$

As is typical for these translations, the elimination rules turns the left rule "upside down" because (like all rules in natural deduction) the principal formula is on the right-hand side of judgment, not the left as in the sequent calculus. This means we now have some conditions to check.

$$\frac{\Delta \vdash s \Longrightarrow A_m}{\Delta \vdash s \Longleftarrow A_m} \Rightarrow/\Leftarrow \qquad \frac{\Delta \vdash e \Longleftarrow A_m}{\Delta \vdash (e : A_m) \Longrightarrow A_m} \Leftarrow/\Rightarrow \qquad \frac{}{\Delta_{\mathsf{W}} \,;\, x : A_m \vdash x \Longrightarrow A_m} \text{hyp}$$

$$\frac{\Delta, x : A_m \vdash e \Longleftarrow B_m}{\Delta \vdash \lambda x.\, e \Longleftarrow A_m \multimap B_m} \multimap I \qquad \frac{\Delta \vdash s \Longrightarrow A_m \multimap B_m \quad \Delta' \vdash e \Longleftarrow A_m}{\Delta \,;\, \Delta' \vdash s\, e \Longrightarrow B_m} \multimap E$$

$$\frac{\Delta \vdash e_\ell \Longleftarrow A_m^\ell \quad (\forall \ell \in L)}{\Delta \vdash \{\ell \Rightarrow e_\ell\}_{\ell \in L} \Longleftarrow \&\{\ell : A_m^\ell\}_{\ell \in L}} \& I \qquad \frac{\Delta \vdash s \Longrightarrow \&\{\ell : A_m^\ell\}_{\ell \in L} \quad (\ell \in L)}{\Delta \vdash s.\ell \Longrightarrow A_m^\ell} \& E$$

$$\frac{\Delta \vdash e \Longleftarrow A_k}{\Delta \vdash \mathbf{susp}\ e \Longleftarrow \uparrow_k^m A_k} \uparrow I \qquad \frac{\Delta' \geq m \quad \Delta' \vdash s \Longrightarrow \uparrow_k^m A_k}{\Delta_{\mathsf{W}} \,;\, \Delta' \vdash \mathbf{force}\ s \Longrightarrow A_k} \uparrow E$$

$$\frac{\Delta \vdash e_1 \Longleftarrow A_m \quad \Delta' \vdash e_2 \Longleftarrow B_m}{\Delta \,;\, \Delta' \vdash (e_1, e_2) \Longleftarrow A_m \otimes B_m} \otimes I$$

$$\frac{\Delta \vdash s \Longrightarrow A_m \otimes B_m \quad \Delta \geq m \geq r \quad \Delta', x_1 : A_m, x_2 : B_m \vdash e' \Longleftarrow C_r}{\Delta \,;\, \Delta' \vdash \mathbf{match}\ s\ ((x_1, x_2) \Rightarrow e') \Longleftarrow C_r} \otimes E$$

$$\frac{}{\Delta_{\mathsf{W}} \vdash (\,) \Longleftarrow \mathbf{1}_m} \mathbf{1} I \qquad \frac{\Delta \vdash s \Longrightarrow \mathbf{1}_m \quad \Delta \geq m \geq r \quad \Delta' \vdash e' \Longleftarrow C_r}{\Delta \,;\, \Delta' \vdash \mathbf{match}\ s\ ((\,) \Rightarrow e') \Longleftarrow C_r} \mathbf{1} E$$

$$\frac{\Delta \vdash e \Longleftarrow A_m^\ell}{\Delta \vdash \ell(e) \Longleftarrow \oplus\{\ell : A_m^\ell\}_{\ell \in L}} \oplus I$$

$$\frac{\Delta \vdash s \Longrightarrow \oplus\{\ell : A_m^\ell\}_{\ell \in L} \quad \Delta \geq m \geq r \quad \Delta', x : A_m^\ell \vdash e_\ell \Longleftarrow C_r \quad (\forall \ell \in L)}{\Delta \,;\, \Delta' \vdash \mathbf{match}\ s\ (\ell(x) \Rightarrow e_\ell)_{\ell \in L} \Longleftarrow C_r} \oplus E$$

$$\frac{\Delta' \geq n \quad \Delta' \vdash e \Longleftarrow A_n}{\Delta_{\mathsf{W}} \,;\, \Delta' \vdash \mathbf{down}\ e \Longleftarrow \downarrow_m^n A_n} \downarrow I \qquad \frac{\Delta \vdash s \Longrightarrow \downarrow_m^n A_n \quad \Delta \geq m \geq r \quad \Delta', x : A_n \vdash e' \Longleftarrow C_r}{\Delta \,;\, \Delta' \vdash \mathbf{match}\ s\ (\mathbf{down}\ x \Rightarrow e') \Longleftarrow C_r} \downarrow E$$

Figure 3 Implicit Bidirectional Natural Deduction.

$$\frac{\Gamma, A_n \vdash C_r}{\Gamma \,;\, \downarrow_m^n A_n \vdash C_r} \downarrow L \qquad \frac{\Delta \vdash s \Longrightarrow \downarrow_m^n A_n \quad \Delta \geq m \geq r \quad \Delta', x : A_n \vdash e' \Longleftarrow C_r}{\Delta \,;\, \Delta' \vdash \mathbf{match}\ s\ (\mathbf{down}\ x \Rightarrow e') \Longleftarrow C_r} \downarrow E$$

$\Delta \geq m$ is needed to enforce independence on the first premise. $m \geq r$ together with $n \geq m$ enforces independence on the second premise. Similar restrictions appear in the other elimination rules for the positive connectives ($\otimes$, $\mathbf{1}$, $\oplus$).

We often say a natural deduction is *normal*, which means that it cannot be reduced, but under which collection of reductions? The difficulty here is that rewrite rules that reduce an introduction of a connective immediately followed by its elimination are not sufficient to achieve deductions that are *analytic* in the sense that they satisfy the subformula property. To obtain analytic deductions, we have to add *permuting conversions*.

We follow a different approach by directly characterizing *verifications* [21, 36], which are proofs that can be seen as constructed by applying introduction rules bottom-up and elimination rules top-down. By definition, verifications satisfy the subformula property and are therefore analytic and a suitable “normal form” even without defining a set of reductions.

How does this play out here? It turns out that if $\Delta \vdash e \Longleftarrow A_m$ then the corresponding proof of A_m (obtained by erasure of expressions) is a *verification* if the $\Leftarrow/\Rightarrow$ rule is disallowed and the $\Rightarrow/\Leftarrow$ rule is restricted to atomic propositions P. As we will see in Section 4, this corresponds precisely to a cut-free sequent calculus derivation where the identity is restricted to atomic propositions. Proof-theoretically, the meaning of a proposition is determined by its *verifications*, which, by definition, only decompose the given proposition into its components. Compare this with general proofs that do not obey such a restriction.

In the next section we will prove that every proposition that has a natural deduction also has a verification by relating the sequent calculus and natural deduction.

▶ **Example 6** (Church's λI calculus)**.** Church [18] introduced the λI calculus in which each bound variable requires at least one occurrence. We obtain the simply-typed λI calculus with one mode S with $\sigma(\mathsf{S}) = \{\mathsf{C}\}$ and using $A_\mathsf{S} \multimap B_\mathsf{S}$ as the only type constructor.

Similarly, we obtain the simply-typed λ-calculus with a single mode U with $\sigma(\mathsf{U}) = \{\mathsf{W}, \mathsf{C}\}$ and the simply-typed *linear* λ-calculus with a single mode L with $\sigma(\mathsf{L}) = \{\,\}$, using $A_\mathsf{L} \multimap B_\mathsf{L}$ as the only type constructor.

▶ **Example 7** (Intuitionistic Natural Deduction)**.** We obtain (structural) intutionistic natural deduction with a single mode U with $\sigma(\mathsf{U}) = \{\mathsf{W}, \mathsf{C}\}$, where we can define $A \vee B = \oplus\{\mathbf{inl} : A, \mathbf{inr} : B\}$ and $\bot = \oplus\{\,\}$, $A \wedge B = \&\{\pi_1 : A, \pi_2 : B\}$ and $\top = \&\{\,\}$ and $A \to B = A \multimap B$.

▶ **Example 8** (Intuitionistic S4)**.** We obtain the fragment of intuitionistic S4 in its dual formulation [42] without possibility ($\Diamond A$) with two modes V and U with $\mathsf{V} > \mathsf{U}$ and $\sigma(\mathsf{V}) = \sigma(\mathsf{U}) = \{\mathsf{W}, \mathsf{C}\}$. As in the DILL example of the adjoint sequent calculus, the mode V is inhabited only by types $\uparrow_\mathsf{U}^\mathsf{V} A_\mathsf{U}$ and we define $\Box A_\mathsf{U} = \downarrow_\mathsf{U}^\mathsf{V} \uparrow_\mathsf{U}^\mathsf{V} A_\mathsf{U}$, which is a *comonad*. The judgment $\Delta\ ;\ \Gamma \vdash C$ *true* with valid hypotheses Δ and true hypothesis Γ is modeled by $\Delta_\mathsf{V}, \Gamma_\mathsf{U} \vdash C_\mathsf{U}$.

The structure of *verifications* is modeled almost exactly with one small exception: we allow the form $\Delta_\mathsf{V} \vdash C_\mathsf{V}$. Because any proposition $B_\mathsf{V} = \uparrow_\mathsf{U}^\mathsf{V} A_\mathsf{U}$, there is only one applicable rule to construct a verification of this judgment: $\uparrow I$ (which, not coincidentally, is invertible).

▶ **Example 9** (Lax Logic)**.** We obtain natural deduction for lax logic [9, 42] with two modes, U and X, with $\mathsf{U} > \mathsf{X}$ and $\sigma(\mathsf{U}) = \sigma(\mathsf{X}) = \{\mathsf{W}, \mathsf{C}\}$. The mode X is inhabited only by $\downarrow_\mathsf{X}^\mathsf{U} A_\mathsf{U}$. We define $\bigcirc A_\mathsf{U} = \uparrow_\mathsf{X}^\mathsf{U} \downarrow_\mathsf{X}^\mathsf{U} A_\mathsf{U}$, which is a strong monad [9].

We model the rules of Pfenning and Davies [42] exactly, except that we allow hypotheses B_X, which must have the form $\downarrow_\mathsf{X}^\mathsf{U} A_\mathsf{U}$. We can eagerly apply $\downarrow E$ to obtain A_U, which again does not lose completeness by the invertibility of $\downarrow L$ in the sequent calculus. We can also obtain linear versions of these relationships following [11], although the term calculi do not match up exactly.

4 Relating Sequent Calculus and Natural Deduction

Rather than trying to find a complete set of proof reductions for natural deduction, we translate a proof to the sequent calculus, apply cut and identity elimination, and then translate the resulting proof back to natural deduction. This is not essential, but it simultaneously proves the soundness and completeness of natural deduction for adjoint logic and the completeness of verifications. This allows us to focus on the computational interpretation in Section 5 that is a form of substructural functional programming.

In general we see the following patterns in the correctness proofs:

- The identity corresponds to $\Rightarrow/\Leftarrow$
- Cut corresponds to $\Leftarrow/\Rightarrow$
- Right rules correspond to introduction rules
- Left rules correspond to upside-down elimination rules
  - For negative connectives ($\multimap$, $\&$, $\uparrow$) they are just reversed
  - For positive connectives ($\otimes$, $\mathbf{1}$, $\oplus$, $\downarrow$) in addition a new hypothesis is introduced in a second premise

The last point justifies reading a hypothesis $x : A_m$ as $x \Longrightarrow A_m$.

For completeness of natural deduction, one might expect to prove that $\Gamma \vdash C$ in the sequent calculus implies $\Gamma \vdash e \Longleftarrow C$ in natural deduction. While this holds, a direct proof would not generate a verification from a cut-free proof. Intuitively, the way the proof proceeds instead is to take a sequent $x_1 : A_1, \ldots, x_n : A_n \vdash C$ (ignoring modes for the moment) and annotate each antecedent with a synthesizing term and the succedent with an expression $s_1 \Longrightarrow A_1, \ldots, s_n \Longrightarrow A_n \vdash e \Longleftarrow C$. This means we have to account for the variables in s_i, and we do this with a substitution θ assigning synthesizing terms to each antecedent in Γ. We therefore define substitutions as mapping from variables to synthesizing terms.

$$\text{Substitutions} \quad \theta \quad ::= \quad \cdot \mid \theta, x \mapsto s$$

We type substitutions with the judgment $\Delta \vdash \theta \Longrightarrow \Gamma$, where Δ contains the free variables in θ. This judgment must respect independence and the structural properties of each antecedent in Γ, as defined by the following rules:

$$\frac{}{\cdot \vdash (\cdot) \Longrightarrow (\cdot)} \qquad \frac{\Delta \vdash \theta \Longrightarrow \Gamma \quad \Delta' \geq m \quad \Delta' \vdash s \Longrightarrow A_m}{\Delta \,;\, \Delta' \vdash (\theta, x \mapsto s) \Longrightarrow (\Gamma, x : A_m)}$$

We will use silently that if $\Delta \vdash \theta \Longrightarrow \Gamma$ and $\Gamma \geq m$ then $\Delta \geq m$.

We write $e(x)$ and $s'(x)$ for terms with (possibly multiple, possibly no) occurrences of x and $e(s)$ and $s'(s)$ for the result of substituting s for x, respectively. Because variables $x : A$ synthesize their types $x \Longrightarrow A$, the following admissible rules are straightforward assuming the premises satisfy our presuppositions.

▶ **Theorem 10** (Substitution Property). *The following properties are admissible:*

(i) *If $\Delta \vdash s \Longrightarrow A_m$ and $\Delta', x : A_m \vdash e(x) \Longleftarrow C_r$ then $\Delta \,;\, \Delta' \vdash e(s) \Longleftarrow C_r$*

(ii) *If $\Delta \vdash s \Longrightarrow A_m$ and $\Delta', x : A_m \vdash s'(x) \Longrightarrow B_k$ then $\Delta \,;\, \Delta' \vdash s'(s) \Longrightarrow B_k$*

Proof. By a straightforward simultaneous rule induction on the second given derivation. In some cases we need to apply monotonicity. For example, if m admits contraction and $\Delta \geq m$, then each hypothesis in Δ must also admit contraction. ◀

Now we have the pieces in place to prove the translation from the sequent calculus to natural deduction.

▶ **Lemma 11** (Context Split). *If $\Delta \vdash \theta \Longrightarrow (\Gamma; \Gamma')$ then there exists θ_1 and θ_2 and Δ_1 and Δ_2 such that $\Delta = \Delta_1; \Delta_2$ and $\Delta_1 \vdash \theta_1 \Longrightarrow \Gamma$ and $\Delta_2 \vdash \theta_2 \Longrightarrow \Gamma'$.*

Proof. By case analysis on the definition of context merge operation and induction on $\Delta \vdash \theta \Longrightarrow (\Gamma; \Gamma')$. We rely on associativity and commutativity of context merge. We show two cases.

Case: $(\Gamma_1, x : A_m) \,;\, (\Gamma_2, x : A_m) = (\Gamma_1 \,;\, \Gamma_2), x : A_m$ and $\mathsf{C} \in \sigma(m)$

$$\frac{\Delta \vdash \theta_{12} \Longrightarrow \Gamma_1 \,;\, \Gamma_2 \quad \Delta' \geq k \quad \Delta' \vdash s \Longrightarrow A_m}{\Delta \,;\, \Delta' \vdash (\theta_{12}, x \mapsto s) \Longrightarrow (\Gamma_1 \,;\, \Gamma_2), x : A_m}$$

$\Delta_1 \vdash \theta_1 \Longrightarrow \Gamma_1$ and
$\Delta_2 \vdash \theta_2 \Longrightarrow \Gamma_2$ and
$\Delta = \Delta_1 \,;\, \Delta_2$ — by IH
$\Delta_1 \,;\, \Delta' \vdash \theta_1, x \mapsto s \Longrightarrow \Gamma_1, x : A_m$ — by rule
$\Delta_2 \,;\, \Delta' \vdash \theta_2, x \mapsto s \Longrightarrow \Gamma_2, x : A_m$ — by rule
since $\mathsf{C} \in \sigma(m)$ and $\Delta' \geq m$, we have $\mathsf{C} \in \sigma(k)$ for any $B_k \in \Delta'$ — by monotonicity
$(\Delta_1 \,;\, \Delta') \,;\, (\Delta_2 \,;\, \Delta') = (\Delta_1 \,;\, \Delta_2) \,;\, \Delta' = \Delta \,;\, \Delta'$ — by previous line

Case: $\Gamma_1 \,;\, (\Gamma_2, x : A_m) = (\Gamma_1 \,;\, \Gamma_2), x : A_m$ and $x \notin \mathrm{dom}(\Gamma_1)$

$$\frac{\Delta \vdash \theta_{12} \Longrightarrow \Gamma_1 \,;\, \Gamma_2 \quad \Delta' \geq k \quad \Delta' \vdash s \Longrightarrow A_m}{\Delta \,;\, \Delta' \vdash (\theta_{12}, x \mapsto s) \Longrightarrow ((\Gamma_1 \,;\, \Gamma_2), x \mapsto A_m)}$$

$\Delta_1 \vdash \theta_1 \Longrightarrow \Gamma_1$ and
$\Delta_2 \vdash \theta_2 \Longrightarrow \Gamma_2$ and
$\Delta = \Delta_1 \,;\, \Delta_2$ — by IH
$\Delta_2 \,;\, \Delta' \vdash \theta_2, x \mapsto s \Longrightarrow \Gamma_2, x \mapsto A_m$ — by rule
$\Delta_1 \,;\, (\Delta_2 \,;\, \Delta') = (\Delta_1 \,;\, \Delta_2) \,;\, \Delta' = \Delta \,;\, \Delta'$ — by associativity of context merge

◀

▶ **Theorem 12** (From Sequent Calculus to Natural Deduction). *theoremseqtond*
If $\Gamma \vdash A_r$ *and* $\Delta \vdash \theta \Longrightarrow \Gamma$ *then* $\Delta \vdash e \Longleftarrow A_r$ *for some* e*.*

Proof. By rule induction on the derivation $\mathcal{D}$ of $\Gamma \vdash A_r$ and applications of inversion on the definition of substitution. We present several indicative cases. In this proof we write out the variables labeling the antecedents in sequents to avoid ambiguities.

Case: $\mathcal{D}$ ends in the identity.

$$\mathcal{D} = \quad \frac{}{\Gamma_{\mathsf{W}} \,;\, x : A_m \vdash A_m}\ \mathsf{id}$$

$\Delta \vdash \theta \Longrightarrow (\Gamma_{\mathsf{W}} \,;\, x : A_m)$ — Given
$\theta = (\theta_{\mathsf{W}}, x \mapsto s)$ — By inversion
$\Delta = (\Delta_{\mathsf{W}} \,;\, \Delta')$ with $\Delta_W \vdash \theta_{\mathsf{W}} \Longrightarrow \Gamma_{\mathsf{W}}$ and $\Delta' \vdash s \Longrightarrow A_m$ — By context split
Δ_{W} satisfies weakening — By monotonicity
$\Delta' \vdash s \Longleftarrow A_m$ — By rule $\Rightarrow/\Leftarrow$
$\Delta_{\mathsf{W}} \,;\, \Delta' \vdash s \Longleftarrow A_m$ — By weakening
$\Delta \vdash s \Longleftarrow A_m$ — Since $\Delta = (\Delta_{\mathsf{W}} \,;\, \Delta')$

Case: $\mathcal{D}$ ends in cut.

$$\mathcal{D} = \quad \frac{\Gamma_1 \geq m \geq r \quad \overset{\mathcal{D}_1}{\Gamma_1 \vdash A_m} \quad \overset{\mathcal{D}_2}{\Gamma_2, x : A_m \vdash C_r}}{\Gamma_1 \,;\, \Gamma_2 \vdash C_r}\ \mathsf{cut}$$

$\Delta \vdash \theta \Longrightarrow (\Gamma_1 \,;\, \Gamma_2)$	Given
$\Delta = (\Delta_1 \,;\, \Delta_2)$, $\theta = (\theta_1, \theta_2)$ with $\Delta_1 \vdash \theta_1 \Longrightarrow \Gamma_1$ and $\Delta_2 \vdash \theta_2 \Longrightarrow \Gamma_2$	By context split
$\Delta_1 \vdash e_1 \Longleftarrow A_m$	By IH on $\mathcal{D}_1$
$\Delta_1 \vdash (e_1 : A_m) \Longrightarrow A_m$	by rule $\Leftarrow/\Rightarrow$
$\Delta_2, x : A_m \vdash (\theta_2, x \mapsto x) \Longrightarrow (\Gamma_2, x : A_m)$	By subst. rule
$\Delta_2, x : A_m \vdash e_2(x) \Longleftarrow C_r$	By IH on $\mathcal{D}_2$
$\Delta_1 \,;\, \Delta_2 \vdash e_2(e_1 : A_m) \Longleftarrow C_r$	By substitution (Theorem 10)

◀

While there are no substitutions involved, the other direction has to take care to introduce a cut *only* for uses of the $\Leftarrow/\Rightarrow$ rule, and identity *only* for uses of the $\Rightarrow/\Leftarrow$ rule. This requires a generalization of the induction hypothesis so that the elimination rules can be turned "upside down".

▶ **Theorem 13** (From Natural Deduction to Sequent Calculus). *theoremndtoseq*
(i) *If* $\Delta \vdash e \Longleftarrow C_r$ *then* $\Delta \vdash C_r$
(ii) *If* $\Delta \vdash s \Longrightarrow A_m$ *and* $\Delta', x : A_m \vdash C_r$ *then* $\Delta \,;\, \Delta' \vdash C_r$

Proof. By simultaneous rule induction on $\Delta \vdash e \Longleftarrow C_r$ and $\Delta \vdash s \Longrightarrow A_m$. We provide several sample cases.
Case: The derivation ends in $\Rightarrow/\Leftarrow$.

$$\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}'\\ \Delta \vdash s \Longrightarrow A_m\end{array}}{\Delta \vdash s \Longleftarrow A_m}\ \Rightarrow/\Leftarrow$$

$x : A_m \vdash A_m$	By identity rule
$\Delta \vdash A_m$	By IH(ii) with $\Delta' = (\cdot)$

Case: The derivation ends in $\Leftarrow/\Rightarrow$.

$$\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}'\\ \Delta \vdash e \Longleftarrow A_m\end{array}}{\Delta \vdash (e : A_m) \Longrightarrow A_m}\ \Leftarrow/\Rightarrow$$

$\Delta', x : A_m \vdash C_r$ and $\Delta \geq r$	Assumption
$\Delta \vdash A_m$	By IH(i) on $\mathcal{D}'$
$\Delta \,;\, \Delta' \vdash C_m$	By rule of cut

◀

As mentioned above, verifications are the foundational equivalent of normal forms in natural deduction. Using the two translations above we can show that every provable proposition has a verification. While we have not written the translations out as functions, they constitute the computational contents of our constructive proof of Theorem 12 and Theorem 13.

▶ **Theorem 14.** *If* $\Delta \vdash e \Longleftarrow A_m$ *then there exists a verification of* $\Delta \vdash e \Longleftarrow A_m$*.*

Proof. Given an arbitrary deduction of $\Delta \vdash e \Longleftarrow A_m$, we can use Theorem 13 (i) to translate it to a sequent derivation of $\Delta \vdash A_m$. By the admissibility of cut and identity (Theorem 2), we can obtain a long cut-free proof of $\Delta \vdash A_m$. We observe that the translation of Theorem 12 translates only cut to $\Leftarrow/\Rightarrow$ and only identity to $\Rightarrow/\Leftarrow$. Using the translation back to natural deduction from a long cut-free proof therefore results in a verification. ◀

5 Dynamics

As mentioned in Section 3, we obtain a simple typing judgment $\Delta \vdash e : A$ by collapsing the distinction between $e \Longleftarrow A$ and $s \Longrightarrow A$, using e as a universal notation for all expressions. Furthermore, the annotation $(e : A_m)$ is removed and the rules $\Rightarrow/\Leftarrow$ and $\Leftarrow/\Rightarrow$ are also removed. The resulting rules remain *syntax-directed* in the sense that for every form of expression there is a unique typing rule.

We further annotate the mode-changing constructors with the mode of their subject, which in each case is uniquely determined by the typing derivation. Some of these annotations are necessary, because the computation rules depend on them; other information is redundant but kept for clarity.

- $\mathbf{susp}_k^m\ e : {\uparrow}_k^m A_k$ if $e : A_k$
- $\mathbf{down}_m^n\ e : {\downarrow}_m^n A_n$ if $e : A_n$
- $\mathbf{force}_k^m\ e : A_k$ if $e : {\uparrow}_k^m A_k$
- $\mathbf{match}_m\ e\ M_r : C_r$ if $e : A_m$

We give a sequential call-by-value semantics similar to the K machine (e.g., [26]), but maintaining a global environment similar to the Milner Abstract Machine [3]. There are two forms of state in the machine:

- $\eta\ ;\ K \triangleright_m e$ (evaluate e of mode m under continuation stack K and environment η)
- $\eta\ ;\ K \blacktriangleleft_m v$ (pass value v of mode m to continuation stack K in environment η)

In the first, e is an expression to be evaluated and K is a stack of continuations that the value of e is passed to for further computation. The second then passes this value v to the continuation stack.

The global environment η maps variables to values, but these values may again reference other variables. In this way it is like Launchbury's [32] heap. We can exploit this to model the call-by-need evaluation strategy, which can be found in our extended version (link in preamble). Because we maintain a global environment, we do not need to build closures, nor do we need to substitute values for variables. Instead, we only (implicitly) rename variables to make them globally unique. This form of specification allows us to isolate the *dynamic use of variables*, which means we can observe the computational consequences of modes and their substructural nature. We could also use the translation to the sequent calculus and then observe the consequence with an explicit heap [48, 43], but in this paper we study natural deduction and functional computation more directly.

The syntax for continuations, environments, values, and machine states is summarized in Figure 4. Although not explicitly polarized (as in [33]), values of negative type ($\multimap$, $\&$, $\uparrow$) are lazy in the sense that they abstract over unevaluated expressions, while values of positive types ($\otimes$, $\mathbf{1}$, $\oplus$, $\downarrow$) are constructed from other values. This will be significant in our analysis of the computational properties of modes. Continuation frames just reflect the left-to-right call-by-value nature of evaluation.

Values are typed as expressions. Frames are typed with $\Gamma \vdash f : B_k < A_m$, which means f takes a value of type A_m and passes a value of type B_k further up the continuation stack. We show the rules for continuations. Note that the non-empty continuation rule has a mode k in the premises that doesn't appear in the conclusion.

$$\frac{}{\Delta_{\mathsf{W}} \vdash \epsilon : A_m < A_m} \qquad \frac{\Delta \vdash K : C_r < B_k \quad \Delta' \vdash f : B_k < A_m}{\Delta\ ;\ \Delta' \vdash K \cdot f : C_r < A_m}$$

Regarding environments we face a fundamental choice. One possibility is to extend the term language of natural deduction with explicit constructs for weakening and contraction. Then, similar to Girard and Lafont [24], no garbage collection would be required during evaluation since uniqueness of references to variables would be maintained.

Frames

$$
\begin{array}{lcll}
f & ::= & \_\ e_2 \mid v_1\ \_ & (\multimap) \\
 & \mid & \_.\ell & (\&) \\
 & \mid & \mathbf{force}_k^m\ \_ & (\uparrow) \\
 & \mid & (\_, e_2) \mid (v_1, \_) & (\otimes) \\
 & \mid & \ell(\_) & (\oplus) \\
 & \mid & \mathbf{down}_m^n\ \_ & (\downarrow) \\
 & \mid & \mathbf{match}_m\ \_\ M_r & (\otimes, \mathbf{1}, \oplus, \downarrow)
\end{array}
$$

Continuations

$$K \quad ::= \quad \epsilon \mid K \cdot f$$

Environments

$$\eta \quad ::= \quad \cdot \mid \eta, x \mapsto v \mid \eta, [x \mapsto v]$$

States

$$
\begin{array}{lcl}
S & ::= & \eta\ ;\ K \rhd_m e \\
 & \mid & \eta\ ;\ K \blacktriangleleft_m v
\end{array}
$$

Values

$$
\begin{array}{lcll}
v & ::= & \lambda x.\, e(x) & (\multimap) \\
 & \mid & \{\ell \Rightarrow e_\ell\}_{\ell \in L} & (\&) \\
 & \mid & \mathbf{susp}_k^m\ e & (\uparrow) \\
 & \mid & (v_1, v_2) & (\otimes) \\
 & \mid & (\,) & (\mathbf{1}) \\
 & \mid & \ell(v) & (\oplus) \\
 & \mid & \mathbf{down}_m^n\ v & (\downarrow)
\end{array}
$$

Figure 4 Machine States.

We pursue here an alternative that leads to slightly deeper properties. We leave the structural rules implicit as in the rules so far. This means that variables of linear mode (that is, a mode that allows neither weakening nor contraction) have uniqueness of reference and their bindings can be deallocated when dereferenced. Variables of structural mode (that is, a mode that allows both weakening and contraction) are simply persistent in the dynamics and therefore could be subject to an explicit garbage collection algorithm.

A difficulty arises with variables that only admit contraction but not weakening. After they are dereferenced the first time, they may or may not be dereferenced again. That is, they could be implicitly weakened after the first access. In order to capture this we introduce a new form of typing $[x : A_m]$ and binding $[x \mapsto v]$ we call *provisional*. A provisional binding does not need to be referenced even if m does not admit weakening. The important new property is that an "ordinary" variable $y : A_k$ that does not admit weakening can not appear in a binding $[x \mapsto v]$. In addition, all the usual independence requirements have to be observed.

The rules for typing expressions, continuations, etc. are extended in the obvious way, allowing variables $[x : A_m]$ to be used or ignored (as a part of some Δ_W). We extend the context merge operation as follows, keeping in mind that $x : A_m$ may require an occurrence of x (depending on $\sigma(m)$), while $[x : A_m]$ does not.

$$
\begin{array}{rclcll}
(\Delta_1, [x : A_m] & ; & (\Delta_2, [x : A_m]) & = & (\Delta_1\, ;\, \Delta_2), [x : A_m] & \text{provided } \mathsf{C} \in \sigma(m) \\
(\Delta_1, x : A_m) & ; & (\Delta_2, [x : A_m]) & = & (\Delta_1\, ;\, \Delta_2), x : A_m & \text{provided } \mathsf{C} \in \sigma(m) \\
(\Delta_1, [x : A_m] & ; & (\Delta_2, x : A_m) & = & (\Delta_1\, ;\, \Delta_2), x : A_m & \text{provided } \mathsf{C} \in \sigma(m) \\
(\Delta_1, [x : A_m]) & ; & \Delta_2 & = & (\Delta_1\, ;\, \Delta_2), [x : A_m] & \text{provided } x \notin \mathrm{dom}(\Delta_2) \\
\Delta_1 & ; & (\Delta_2, [x : A_m]) & = & (\Delta_1\, ;\, \Delta_2), [x : A_m] & \text{provided } x \notin \mathrm{dom}(\Delta_1)
\end{array}
$$

We have the following typing rules for environments. Δ_W now means that every declaration in Δ can be weakened, either explicitly because its mode allows weakening, or implicitly because it is provisional.

$$
\frac{}{(\cdot) : (\cdot)}
\qquad
\frac{\eta : (\Delta\, ;\, \Delta') \quad \Delta' \geq m \quad \Delta' \vdash v : A_m}{(\eta, x \mapsto v) : (\Delta, x : A_m)}
\qquad
\frac{\eta : (\Delta\, ;\, \Delta'_{\mathsf{W}}) \quad \Delta'_{\mathsf{W}} \geq m \quad \Delta'_{\mathsf{W}} \vdash v : A_m}{(\eta, [x \mapsto v]) : (\Delta, [x : A_m])}
$$

As an example, consider $\eta_0 = (x \mapsto (\,), y \mapsto \lambda f.\, f\, x)$ where the mode of variables is immaterial, but let's fix them to be L with $\sigma(\mathsf{L}) = \{\,\}$.

$$\dfrac{\dfrac{\dfrac{}{(\cdot) : (\cdot)} \quad \dfrac{}{\cdot \vdash (\,) : \mathbf{1}_{\mathsf{L}}}}{(x \mapsto (\,)) : (x : \mathbf{1}_{\mathsf{L}})} \qquad \dfrac{\vdots}{x : \mathbf{1}_{\mathsf{L}} \vdash \lambda f.\, f\, x : (\mathbf{1}_{\mathsf{L}} \multimap A_{\mathsf{L}}) \multimap A_{\mathsf{L}}}}{(x \mapsto (\,), y \mapsto \lambda f.\, f\, x) : (y : (\mathbf{1}_{\mathsf{L}} \multimap A_{\mathsf{L}}) \multimap A_{\mathsf{L}})}$$

We observe that the binding of $x \mapsto (\,)$ does not contribute a declaration $x : \mathbf{1}$ to the result context due to the occurrence of x in the value of y.

Now consider a slightly modified version where the mode of both x and y is S with $\sigma(\mathsf{S}) = \{\mathsf{C}\}$, and the binding of $y \mapsto \ldots$ becomes provisional. This modified example is no longer well-typed.

$$\dfrac{\dfrac{\dfrac{}{(\cdot) : (\cdot)} \quad \dfrac{}{\cdot \vdash (\,) : \mathbf{1}_{\mathsf{S}}}}{(x \mapsto (\,)) : (x : \mathbf{1}_{\mathsf{S}})} \qquad \dfrac{\vdots}{x : \mathbf{1}_{\mathsf{S}} \vdash \lambda f.\, f\, x : (\mathbf{1}_{\mathsf{S}} \multimap A_{\mathsf{S}}) \multimap A_{\mathsf{S}}}}{(x \mapsto (\,), [y \mapsto \lambda f.\, f\, x]) : (y : [(\mathbf{1}_{\mathsf{S}} \multimap A_{\mathsf{S}}) \multimap A_{\mathsf{S}})]} \; ??$$

The problem is at the rule application marked by ??. The variable y does not need to be used, despite its mode, because the binding is provisional. This means that x might also not be used because its only occurrence is in the value of y. But that is not legal, since the mode of x does not admit weakening and the binding is not provisional.

We type abstract machine states with the type of their final answer, that is $S : C_r$.

$$\dfrac{\eta : (\Delta \,;\, \Delta') \quad \Delta \vdash K : C_r < A_m \quad \Delta' \vdash e : A_m}{(\eta \,;\, K \rhd_m e) : C_r} \qquad \dfrac{\eta : (\Delta \,;\, \Delta') \quad \Delta \vdash K : C_r < A_m \quad \Delta' \vdash v : A_m}{(\eta \,;\, K \blacktriangleleft_m v) : C_r}$$

We now continue with the computational rules for our abstract machine. The full set of rules can be found in Figure 5. We factor out passing a value to a match, $\eta \,;\, v \blacktriangleright_m M = \eta' \,;\, e'$ that produces a (possibly extended) environment η' and expression e'. In all cases below, we presuppose the variable names are chosen so the extended environment has unique bindings for each variable. For an extension with mutual recursion, see our extended version (link in preamble).

We obtain the following expected theorems of preservation and progress.

▶ **Theorem 15** (Preservation). *theorempreservation If $S : A$ and $S \longrightarrow S'$ then $S' : A$.*

Proof. By cases on $S \longrightarrow S'$, applying inversion to the typing of S and assembling a typing derivation of S' from the resulting information.

The trickiest case involves dereferencing a variable $x \mapsto v$ admitting contraction. It is sound because every variable y occurring in v must also admit contraction by monotonicity and, furthermore, such variables still have an occurrence in the value v that is being returned. Therefore in the typing of the environment we can now type $[x \mapsto v]$ with $[x : A_m]$. ◀

A machine state is *final* if it has the form $\eta \,;\, \epsilon \blacktriangleleft_m v$, that is, if a value is returned to the empty continuation in some global environment η. In order to prove progress, we need to characterize values of a given type using a *canonical forms* property. Note that we allow a context Δ to provide for the variables that may be embedded in a value of negative type ($\multimap$, $\&$, $\uparrow$), but that a variable by itself does not count as a value.

$$
\begin{array}{llrlcll}
\eta & ; & () & \blacktriangleright_m\ (() \Rightarrow e') & = & \eta & ;\ e' \\
\eta & ; & (v_1, v_2) & \blacktriangleright_m\ ((x_1, x_2) \Rightarrow e'(x_1, x_2)) & = & \eta, x_1 \mapsto v_1, x_2 \mapsto v_2 & ;\ e'(x_1, x_2) \\
\eta & ; & \ell(v) & \blacktriangleright_m\ (\ell(x) \Rightarrow e'_\ell(x))_{\ell \in L} & = & \eta, x \mapsto v & ;\ e'_\ell(x) \\
\eta & ; & \mathbf{down}^n_m\, v & \blacktriangleright_m\ (\mathbf{down}(x) \Rightarrow e'(x)) & = & \eta, x \mapsto v & ;\ e'(x)
\end{array}
$$

$$
\begin{array}{lcll}
\eta, x \mapsto v, \eta' \,;\, K \triangleright_m x & \longrightarrow & \eta, \eta' \,;\, K \blacktriangleleft_m v & (\mathsf{C} \notin \sigma(m)) \\
\eta, x \mapsto v, \eta' \,;\, K \triangleright_m x & \longrightarrow & \eta, [x \mapsto v], \eta' \,;\, K \blacktriangleleft_m v & (\mathsf{C} \in \sigma(m)) \\
\\
\eta \,;\, K \triangleright_r \mathbf{match}_m\ e\ M_r & \longrightarrow & \eta \,;\, K \cdot (\mathbf{match}_m\ \_\ M_r) \triangleright_m e & (\otimes, \mathbf{1}, \oplus, \downarrow) \\
\eta \,;\, K \cdot (\mathbf{match}_m\ \_\ M_r) \blacktriangleleft_m v & \longrightarrow & \eta' \,;\, K \triangleright_r e' \quad \text{where } \eta \,;\, v \blacktriangleright_m M_r = \eta' \,;\, e' & \\
\\
\eta \,;\, K \triangleright_m \lambda x.\, e(x) & \longrightarrow & \eta \,;\, K \blacktriangleleft_m \lambda x.\, e(x) & (\multimap) \\
\eta \,;\, K \triangleright_m (e_1\, e_2) & \longrightarrow & \eta \,;\, K \cdot (\_\ e_2) \triangleright_m e_1 & \\
\eta \,;\, K \cdot (\_\ e_2) \blacktriangleleft_m v_1 & \longrightarrow & \eta \,;\, K \cdot (v_1\ \_) \triangleright_m e_2 & \\
\eta \,;\, K \cdot (\lambda x.\, e(x), \_) \blacktriangleleft_m v_2 & \longrightarrow & \eta, x \mapsto v \,;\, K \triangleright_m e(x) & \\
\\
\eta \,;\, K \triangleright_m \{\ell \Rightarrow e_\ell\}_{\ell \in L} & \longrightarrow & \eta \,;\, K \blacktriangleleft_m \{\ell \Rightarrow e_\ell\}_{\ell \in L} & (\&) \\
\eta \,;\, K \triangleright_m e.\ell & \longrightarrow & \eta \,;\, K \cdot (\_.\ell) \triangleright_m e & \\
\eta \,;\, K \cdot (\_.\ell) \blacktriangleleft_m \{\ell \Rightarrow e_\ell\}_{\ell \in L} & \longrightarrow & \eta \,;\, K \triangleright_m e_\ell & (\ell \in L) \\
\\
\eta \,;\, K \triangleright_m \mathbf{susp}^m_k\ e & \longrightarrow & \eta \,;\, K \blacktriangleleft_m \mathbf{susp}^m_k\ e & (\uparrow) \\
\eta \,;\, K \triangleright_k \mathbf{force}^m_k\ e & \longrightarrow & \eta \,;\, K \cdot (\mathbf{force}^m_k\ \_) \triangleright_m e & \\
\eta \,;\, K \cdot (\mathbf{force}^m_k\ \_) \blacktriangleleft_m \mathbf{susp}^m_k e & \longrightarrow & \eta \,;\, K \triangleright_k e & \\
\\
\eta \,;\, K \triangleright_m (e_1, e_2) & \longrightarrow & \eta \,;\, K \cdot (\_, e_2) \triangleright_m e_1 & (\otimes) \\
\eta \,;\, K \cdot (\_, e_2) \blacktriangleleft_m v_1 & \longrightarrow & \eta \,;\, K \cdot (v_1, \_) \triangleright_m e_2 & \\
\eta \,;\, K \cdot (v_1, \_) \blacktriangleleft_m v_2 & \longrightarrow & \eta \,;\, K \blacktriangleleft_m (v_1, v_2) & \\
\\
\eta \,;\, K \triangleright_m () & \longrightarrow & \eta \,;\, K \blacktriangleleft_m () & (\mathbf{1}) \\
\\
\eta \,;\, K \triangleright_m \ell(e) & \longrightarrow & \eta \,;\, K \cdot \ell(\_) \triangleright_m e & (\oplus) \\
\eta \,;\, K \cdot \ell(\_) \blacktriangleleft_m v & \longrightarrow & \eta \,;\, K \blacktriangleleft_m \ell(v) & \\
\\
\eta \,;\, K \triangleright_m \mathbf{down}^n_m\ e & \longrightarrow & \eta \,;\, K \cdot \mathbf{down}^n_m\ \_ \triangleright_n e & (\downarrow) \\
\eta \,;\, K \cdot (\mathbf{down}^n_m\ \_) \blacktriangleleft_n v & \longrightarrow & \eta \,;\, K \blacktriangleleft_m \mathbf{down}^n_m\ v &
\end{array}
$$

Figure 5 Computation Rules.

▶ **Lemma 16** (Canonical Forms). *If $\Delta \vdash v : A_m$ then one of the following applies:*

(i) *if $A_m = B_m \multimap C_m$ then $v = \lambda x.\, e(x)$ for some e*

(ii) *if $A_m = \&\{\ell : A^\ell_m\}_{\ell \in L}$ then $v = \{\ell \Rightarrow e_\ell\}_{\ell \in L}$ for some set e_ℓ*

(iii) *if $A_m = \uparrow^m_k B_k$ then $v = \mathbf{susp}^m_k\ e$*

(iv) *if $A_m = B_m \otimes C_m$ then $v = (v_1, v_2)$ for values v_1 and v_2*

(v) *if $A_m = \mathbf{1}$ then $v = ()$*

(vi) *if $A_m = \oplus\{\ell : B^\ell_m\}_{\ell \in L}$ then $v = \ell(v')$ for some $\ell \in L$ and value v'*

(vii) *if $A_m = \downarrow^n_m A_n$ then $v = \mathbf{down}^n_m v'$ for some value v'*

Proof. As usual, by inversion on typing and the possible forms of values, remembering that variables do not count as values. ◀

▶ **Theorem 17** (Progress). *theoremprogress If $S : C_r$ then either S is final or $S \mapsto S'$ for some S'*

Proof. By cases on the typing derivation for the configuration and inversion on the typing of the embedded frames, values, and expressions. We apply the canonical forms theorem when we need the shape of a value. ◀

Purely positive types play an important role because we view values of these types as *directly observable*, while values of negative types can only be observed indirectly through their elimination forms.

$$\text{Purely positive types} \quad A^+, B^+ \quad ::= \quad A^+ \otimes B^+ \mid \mathbf{1} \mid \oplus\{\ell : A^+_\ell\}_{\ell \in L} \mid {\downarrow} A^+$$

Values of purely positive types are closed, even if values of negative types may not be.

▶ **Lemma 18** (Positive Values). *If $\Delta \vdash v : A^+_r$ then $\cdot \vdash v : A^+_r$ and all declarations in Δ admit weakening (either due to their mode or because they are provisional).*

Proof. By induction on the structure of the typing derivation, recalling that variables are not values. ◀

We call a variable $x : A_m$ *linear* if $\sigma(m) = \{\,\}$, that is, the mode m admits neither weakening or contraction. We extend this term to types, bindings in the environment, etc. in the obvious way.

▶ **Theorem 19** (Freedom from Garbage). *theoremfreedom If $\cdot \vdash e : A^+_r$ and $\cdot \,;\, \epsilon \rhd_r e \longrightarrow^* \eta \,;\, \epsilon \blacktriangleleft_r v$, then η does not contain a binding $x \mapsto v$ with $\sigma(m) = \{\,\}$ where m is the mode of x.*

Proof. Because A^+_r is purely positive, we know by Lemma 18 that v is closed.

When the continuation K is empty, the typing rule for valid states implies that $\eta : \Delta$ and $\Delta \vdash v : A^+_r$ for some Δ. Since v is closed, Δ cannot contain any linear variables.

Then we prove by induction on the typing of η that none of variables in η can be linear. In the inductive case

$$\frac{\eta' : (\Delta \,;\, \Delta') \quad \Delta' \geq m \quad \Delta' \vdash v : A_m}{(\eta', x \mapsto v) : (\Delta, x : A_m)}$$

we know that m must admit weakening or contraction or both. Since $\Delta' \geq m$, by monotonicity, Δ' must also admit weakening or contraction and we can apply the induction hypothesis to $\eta' : (\Delta \,;\, \Delta')$. ◀

We call a variable x_m, an expression $e : A_m$, or a binding $x \mapsto v$ *strict* if $\sigma(m) \subseteq \{\mathsf{C}\}$, that is, m does not admit weakening.

▶ **Theorem 20** (Strictness). *theoremstrictness If $\cdot \vdash e : A^+_r$ and $\cdot \,;\, \epsilon \rhd_r e \longrightarrow^* \eta \,;\, \epsilon \blacktriangleleft_r v$, then every* strict *binding in η is of the form $[x \mapsto v]$.*

Proof. Because A^+_r is purely positive, we know by Lemma 18 that v is closed.

When the continuation K is empty, the typing rule for valid states implies $\eta : \Delta$ and $\Delta \vdash v : A^+_r$ for some Δ. Since v is closed, Δ contains strict variables only in the form $[x : A_m]$.

We prove by induction on the typing of η all strict variables in η have the form $[x \mapsto w]$. There are two inductive cases.

$$\frac{\eta' : (\Delta \,;\, \Delta') \quad \Delta' \geq m \quad \Delta' \vdash w : A_m}{(\eta', x \mapsto w) : (\Delta, x : A_m)}$$

Since m is not strict, it must admit weakening. Since $\Delta' \geq m$, every variable in Δ' must also admit weakening by monotonicity, so we can apply the induction hypothesis to $\Delta\,;\Delta'$.

$$\frac{\eta' : (\Delta\,;\Delta'_{\mathsf{W}}) \quad \Delta'_{\mathsf{W}} \geq m \quad \Delta'_{\mathsf{W}} \vdash w : A_m}{(\eta', [x \mapsto v]) : (\Delta, [x : A_m])}$$

Any declaration in Δ'_{W} either directly admits weakening or is of the form $[y : A_k]$ for a strict k so we can apply the induction hypothesis to $\eta' : (\Delta\,;\Delta'_{\mathsf{W}})$. ◀

In this context of call-by-value, this property expresses that every strict variable will be read at least once, since a binding $[x \mapsto v]$ arises only from reading the value of x. In call-by-need it means that the value is indeed needed.

▶ **Theorem 21** (Dead Code)**.** *theoremdeadcode If* $\cdot \vdash e : A_r^+$ *and* $\cdot\,;\epsilon \rhd_r e \longrightarrow^* \eta\,;\epsilon \blacktriangleleft_r v$ *then every state during the computation either evaluates* $\rhd_m$ *or returns* $\blacktriangleleft_m$ *for* $m \geq r$*.*

Proof. Most rule do not change the subject's mode. Several rules potentially raise the mode, name evaluating a **match**, a **force**, or a **down**. For each of these there is a corresponding rule lowering the mode back to its original, namely return a value to a **match**, to a **force**, or to a **down**.

We say the mode of a frame f is the mode of the following state after a value is returned to f. We prove by induction over the computation that in all states, all continuation frames and subjects have modes $m \geq r$. ◀

▶ **Theorem 22** (Erasure)**.** *corollaryerasure Assume* $\cdot \vdash e : A_r^+$ *and* $\cdot\,;\epsilon \rhd_r e \longrightarrow^* \eta\,;\epsilon \blacktriangleleft_r v$*. Let* Ω *be a new term of every type and no transition rule.*

If we obtain e' *by replacing all subterms of type* B_k *for* $k \not\geq r$ *with* Ω*, then evaluation* e' *still terminates in a final state. This final state differs from* v *in that subterms of mode* $k \not\geq r$ *are also replaced by* Ω*.*

Proof. The computation of e' parallels that of e. It would only get stuck for a state $\eta'\,;K' \rhd_k \Omega$, but that is impossible by the preceding dead code theorem since $k \not\geq r$. ◀

6 Algorithmic Type Checking

The bidirectional type system of Section 3 is not yet algorithmic, among other things because splitting a given context into $\Delta = (\Delta_1\,;\Delta_2)$ is nondeterministic. One standard solution is to track which hypotheses are used in one premise (which ends up Δ_1), subtract them from the available ones, and pass the remainder into the second premise (which ends up Δ_2 together with an overall remainder) [14]. This originated in proof search, but here when we actually have a proof terms available to check, other options are available. *Additive* resource management computes the *used* hypotheses (rather than the unused ones) and merges ("adds") them [37, 5], which is conceptually slightly simpler and also has been shown to be more efficient [28].

The main complication in the additive approach are internal and external choice, more specifically, the $\&R$ and $\oplus L$ rules when the choice is empty. For example, while checking $\Delta \vdash \{\,\} \Longleftarrow \&\{\,\}$ any subset of Δ could be used. We reuse the idea from the dynamics to have provisional hypotheses $[x : A_m]$. In the additive approach, the context merge for provisional hypotheses then no longer requires contraction since such variables do not occur (but could be considered as used). There are a plethora of different judgments, but it is not clear how to simplify them. In defining the additive approach, the main two judgments are $\Gamma \vdash s \Longrightarrow A_m \;/\; \Xi$ and $\Gamma \vdash e \Longleftarrow A_m \;/\; \Xi$ which we summarize as $\Gamma \vdash e \Longleftrightarrow A_m \;/\; \Xi$ where Γ is a plain (that is, free of provisional hypotheses) context containing *all* variables that might occur in e (regardless of mode or structural properties) and Ξ is a context that may contain provisional hypotheses. We maintain the mode invariant $\Xi \geq m$ (even if it may be the case that $\Gamma \not\geq m$). The rules can be found in Figure 6. We show some of the crucial properties to understand the rules, defining some of these operations later with these properties in mind.

Because we keep the contexts Δ free of provisional hypotheses, we define the relation $\Xi \sqsupseteq \Delta$ which may remove or keep provisional hypotheses.

$$\begin{array}{rcllcll}
(\Xi, x : A_m) & \sqsupseteq & (\Delta, x : A) & \text{if} & \Xi & \sqsupseteq & \Delta \\
(\Xi, [x : A_m]) & \sqsupseteq & (\Delta, x : A) & \text{if} & \Xi & \sqsupseteq & \Delta \\
(\Xi, [x : A_m]) & \sqsupseteq & \Delta & \text{if} & \Xi & \sqsupseteq & \Delta \\
(\cdot) & \sqsupseteq & (\cdot) & & & &
\end{array}$$

With this relation, we can state the *soundness* of algorithmic typing.

▶ **Theorem 23** (Soundness of Algorithmic Typing). *If $\Gamma \vdash e \Longleftrightarrow A_m \;/\; \Xi$ and $\Xi \sqsupseteq \Delta$ then $\Delta \vdash e \Longleftrightarrow A_m$.*

Proof. By rule induction on the algorithmic typing derivation and inversion of the $\Xi \sqsupseteq \Delta$ judgment. ◀

For completeness we need a different relation $\Delta \geq \Xi$ which means that Ξ contains a legal subset of the hypotheses in Δ. This means hypotheses in Δ might be in Ξ (possibly provisional) or not, but then only if they can be weakened.

$$\begin{array}{rcllclll}
(\Delta, x : A_m) & \geq & (\Xi, x : A_m) & \text{if} & \Delta & \geq & \Xi & \\
(\Delta, x : A_m) & \geq & (\Xi, [x : A_m]) & \text{if} & \Delta & \geq & \Xi & \\
(\Delta, x : A_m) & \geq & \Xi & \text{if} & \Delta & \geq & \Xi & \text{provided } \mathsf{W} \in \sigma(m) \\
(\cdot) & \geq & (\cdot) & & & & &
\end{array}$$

With this relation we can state the *completeness* of algorithmic typing.

▶ **Theorem 24** (Completeness of Algorithmic Typing). *If $\Delta \vdash e \Longleftrightarrow A_m$ then $\Delta \vdash e \Longleftrightarrow A_m \;/\; \Xi$ for some Ξ with $\Delta \geq \Xi$*

Proof. By rule induction on the given bidirectional typing. ◀

For the algorithm itself we need several operations. Some key properties of these operations that are needed in the soundness and completness proof can be found in the extended version of this paper (link in preamble).

The first, $\Xi \setminus x : A$ removes $x : A$ from Ξ if this is legal operation. Its prototypical use is in the $\multimap I$ rule. For the rule application to be correct the new variable $x : A_m$ must either have been used and therefore occur in Ξ, or the mode m must allow weakening.

$$\begin{array}{rcclll}
(\Xi, x : A_m) & \setminus & x : A_m & = & \Xi & \\
(\Xi, [x : A_m]) & \setminus & x : A_m & = & \Xi & \\
(\Xi, y : B_k) & \setminus & x : A_m & = & (\Xi \setminus x : A_m), y : B_k & \text{provided } y \neq x \\
(\Xi, [y : B_k]) & \setminus & x : A_m & = & (\Xi \setminus x : A_m), [y : B_k] & \text{provided } y \neq x \\
(\cdot) & \setminus & x : A_m & = & (\cdot) & \text{provided } \mathsf{W} \in \sigma(m)
\end{array}$$

We also need two forms of context restriction. The first $\Xi\|_m$ removes all hypotheses whose mode is *not* greater or equal to m to restore our invariant. It fails if Ξ contains a *used* hypothesis B_r with $r \not\geq m$. It is used only in the $\uparrow I$ rule to restore the invariant.

The second form of context restriction occurs in the case of an empty internal or external choice. All of the hypothesis that are allowed by the independence principle could be considered used, but they might also not. We write $[\Gamma|_m]$. It is used only in the nullary case for internal and external choice.

$$
\begin{array}{rcll|rcll}
(\Xi, x : A_k)\|_m & = & \Xi\|_m, x : A_k & (k \geq m) & [(\Gamma, x : A_k)|_m] & = & [\Gamma|_m], [x : A_k] & (k \geq m) \\
(\Xi, [x : A_k])\|_m & = & \Xi\|_m, [x : A_k] & (k \geq m) & & & & \\
(\Xi, [x : A_k])\|_m & = & \Xi\|_m & (k \not\geq m) & [(\Gamma, x : A_k)|_m] & = & [\Gamma|_m] & (k \not\geq m) \\
(\cdot)\|_m & = & & (\cdot) & [(\cdot)|_m] & = & (\cdot) &
\end{array}
$$

We come to the final operation $\Xi_1 \sqcup \Xi_2$ which is needed for $\& I$ and $\oplus E$. Variables used in one branch must also be used in all other branches, or be available for weakening, either because they are provisional or because their mode admits weakening. This idea is captured formally by the definition of $\sqcup$.

$$
\begin{array}{rclcll}
(\Xi_1, x : A_m) & \sqcup & (\Xi_2, x : A_m) & = & (\Xi_1 \sqcup \Xi_2), x : A_m & \\
(\Xi_1, [x : A_m]) & \sqcup & (\Xi_2, x : A_m) & = & (\Xi_1 \sqcup \Xi_2), x : A_m & \\
(\Xi_1, x : A_m) & \sqcup & (\Xi_2, [x : A_m]) & = & (\Xi_1 \sqcup \Xi_2), x : A_m & \\
(\Xi_1, [x : A_m]) & \sqcup & (\Xi_2, [x : A_m]) & = & (\Xi_1 \sqcup \Xi_2), [x : A_m] & \\
(\Xi_1, x : A_m) & \sqcup & \Xi_2 & = & (\Xi_1 \sqcup \Xi_2), x : A_m & \text{for } x \notin \mathrm{dom}(\Xi_2),\ \mathsf{W} \in \sigma(m) \\
\Xi_1 & \sqcup & (\Xi_2, x : A_m) & = & (\Xi_1 \sqcup \Xi_2), x : A_m & \text{for } x \notin \mathrm{dom}(\Xi_1),\ \mathsf{W} \in \sigma(m) \\
(\Xi_1, [x : A_m]) & \sqcup & \Xi_2 & = & \Xi_1 \sqcup \Xi_2 & \text{for } x \notin \mathrm{dom}(\Xi_2) \\
\Xi_1 & \sqcup & (\Xi_2, [x : A_m]) & = & \Xi_1 \sqcup \Xi_2 & \text{for } x \notin \mathrm{dom}(\Xi_1) \\
(\cdot) & \sqcup & (\cdot) & = & (\cdot) &
\end{array}
$$

7 Conclusion

We have presented a natural deduction formulation of adjoint logic. By carefully constructing these rules and the translations to and from the sequent calculus, we automatically obtained the presence of long normal forms for the proofs in natural deduction. We then presented a computational interpretation in the form of a state machine with a global context which leads to proofs of some properties of programs that come directly from having a mode hierarchy. Lastly, we presented an algorithmic type checking system, that due to the empty sum and (positive) product constructors requires a somewhat complicated approach.

There have been recent proposals to extend the adjoint approach to combining logics to dependent types. Licata et al. [34, 35] permit dependent types and richer connections between the logics that are combined, but certain properties such as independence are no longer fundamental and have to be proved in each case where they apply. While they mostly stay within a sequent calculus, they also briefly introduce natural deduction. They further provide a categorical semantics. Hanukaev and Eades [25] also permit dependent types and use the graded/algebraic approach to defining their system. However, their approach to dependency appears incompatible with control of contraction, so their adjoint structure is not nearly as general as ours. They also omit empty internal choice (and external choice altogether), which created some of the trickiest issues in our system. Curien et al. [19] investigate call-by-push-value [33] and provide a semantic foundation for the adjunction properties that is flexible enough to accommodate effects. It also incorporates Benton's mixed

$$\frac{\Gamma \vdash s \Longrightarrow A_m \;/\; \Xi}{\Gamma \vdash s \Longleftarrow A_m \;/\; \Xi}\ {\Rightarrow}/{\Leftarrow} \qquad \frac{\Gamma \vdash e \Longleftarrow A_m \;/\; \Xi}{\Gamma \vdash (e : A_m) \Longrightarrow A_m \;/\; \Xi}\ {\Leftarrow}/{\Rightarrow} \qquad \frac{x : A_m \in \Gamma}{\Gamma \vdash x \Longrightarrow A_m \;/\; (x : A_m)}\ \mathsf{hyp}$$

$$\frac{\Gamma, x : A_m \vdash e \Longleftarrow B_m \;/\; \Xi}{\Gamma \vdash \lambda x.\, e \Longleftarrow A_m \multimap B_m \;/\; (\Xi \setminus x : A_m)}\ {\multimap}I \qquad \frac{\Gamma \vdash s \Longrightarrow A_m \multimap B_m \;/\; \Xi \quad \Gamma \vdash e \Longleftarrow A_m \;/\; \Xi'}{\Gamma \vdash s\, e \Longrightarrow B_m \;/\; \Xi \,;\, \Xi'}\ {\multimap}E$$

$$\frac{}{\Gamma \vdash \{\,\} \Longleftarrow \&_m\{\,\} \;/\; [\Gamma|_m]}\ \&I_0 \qquad \frac{\Gamma \vdash e_\ell \Longleftarrow A_m^\ell \;/\; \Xi_\ell \quad (\forall \ell \in L \neq \emptyset)}{\Gamma \vdash \{\ell \Rightarrow e_\ell\}_{\ell \in L} \Longleftarrow \&\{\ell : A_m^\ell\}_{\ell \in L} \;/\; \sqcup_{\ell \in L} \Xi_\ell}\ \&I$$

$$\frac{\Gamma \vdash e \Longrightarrow \&\{\ell : A_m^\ell\}_{\ell \in L} \;/\; \Xi \quad (\ell \in L)}{\Gamma \vdash e.\ell \Longrightarrow A_m^\ell \;/\; \Xi}\ \&E$$

$$\frac{\Gamma \vdash e \Longleftarrow A_k \;/\; \Xi}{\Gamma \vdash \mathbf{susp}\ e \Longleftarrow {\uparrow}_k^m A_k \;/\; \Xi\|_m}\ {\uparrow}I \qquad \frac{\Gamma \vdash s \Longrightarrow {\uparrow}_k^m A_k \;/\; \Xi}{\Gamma \vdash \mathbf{force}\ s \Longrightarrow A_k \;/\; \Xi}\ {\uparrow}E$$

$$\frac{\Gamma \vdash e_1 \Longleftarrow A_m \;/\; \Xi \quad \Gamma \vdash e_2 \Longleftarrow B_m \;/\; \Xi'}{\Gamma \vdash (e_1, e_2) \Longleftarrow A_m \otimes B_m \;/\; \Xi \,;\, \Xi'}\ {\otimes}I$$

$$\frac{\Gamma \vdash s \Longrightarrow A_m \otimes B_m \;/\; \Xi \quad m \geq r \quad \Gamma, x_1 : A_m, x_2 : B_m \vdash e' \Longleftarrow C_r \;/\; \Xi'}{\Gamma \vdash \mathbf{match}\ s\ ((x_1, x_2) \Rightarrow e') \Longleftarrow C_r \;/\; \Xi \,;\, (\Xi' \setminus x_1 : A_m \setminus x_2 : B_m)}\ {\otimes}E$$

$$\frac{}{\Gamma \vdash (\,) \Longleftarrow \mathbf{1}_m \;/\; (\cdot)}\ \mathbf{1}I \qquad \frac{\Gamma \vdash s \Longrightarrow \mathbf{1}_m \;/\; \Xi \quad m \geq r \quad \Gamma \vdash e' \Longleftarrow C_r \;/\; \Xi'}{\Gamma \vdash \mathbf{match}\ s\ ((\,) \Rightarrow e') \Longleftarrow C_r \;/\; \Xi \,;\, \Xi'}\ \mathbf{1}E$$

$$\frac{\Gamma \vdash e \Longleftarrow A_m^\ell \;/\; \Xi \quad (\ell \in L)}{\Gamma \vdash \ell(e) \Longleftarrow \oplus\{\ell : A_m^\ell\}_{\ell \in L} \;/\; \Xi}\ {\oplus}I \qquad \frac{\Gamma \vdash s \Longrightarrow \oplus_m\{\,\} \;/\; \Xi \quad m \geq r}{\Gamma \vdash \mathbf{match}\ s\ (\,) \Longleftarrow C_r \;/\; \Xi \,;\, [\Gamma|_r]}\ {\oplus}E_0$$

$$\frac{\Gamma \vdash s \Longrightarrow \oplus\{\ell : A_m^\ell\}_{\ell \in L} \;/\; \Xi \quad m \geq r \quad \Gamma, x : A_m^\ell \vdash e_\ell \Longleftarrow C_r \;/\; \Xi'_\ell \quad (\forall \ell \in L \neq \emptyset)}{\Gamma \vdash \mathbf{match}\ s\ (\ell(x) \Rightarrow e_\ell)_{\ell \in L} \Longleftarrow C_r \;/\; \Xi \,;\, \sqcup_{\ell \in L}(\Xi'_\ell \setminus x : A_m^\ell)}\ {\oplus}E$$

$$\frac{\Gamma \vdash e \Longleftarrow A_n \;/\; \Xi}{\Gamma \vdash \mathbf{down}\ e \Longleftarrow {\downarrow}_m^n A_n \;/\; \Xi}\ {\downarrow}I \qquad \frac{\Gamma \vdash s \Longrightarrow {\downarrow}_m^n A_n \;/\; \Xi \quad m \geq r \quad \Gamma, x : A_n \vdash e' \Longleftarrow C_r \;/\; \Xi'}{\Gamma \vdash \mathbf{match}\ s\ (\mathbf{down}\ x \Rightarrow e') \Longleftarrow C_r \;/\; \Xi \,;\, (\Xi' \setminus x : A_n)}\ {\downarrow}E$$

Figure 6 Algorithmic Typing for Natural Deduction.

linear/nonlinear calculus [7] in the form of a sequent calculus but does not consider a general preorder of modes or more flexible structural properties. None of these propose an algorithm for type checking or an operational semantics that would exploit the substructural and mode properties to obtain "free theorems" about well-typed programs as in our dynamics.

We are pursuing several avenues building on the results of this paper. On the foundational side, we are looking for a direct algorithm to convert an arbitrary natural deduction into a verification. On the programming side, we are considering *mode polymorphism*: type-checking the same expression against multiple different modes to avoid code duplication. On the application side, we are considering staged computation, quotation, and metaprogramming, decomposing the usual type $\Box A$ or its contextual analogue along the lines of Example 8.

References

1 Andreas Abel, Nils Anders Danielsson, and Oskar Eriksson. A graded modal dependent type theory with a universe and erasure, formalized. *Proc. ACM Program. Lang.*, 7(ICFP'23):920–954, 2023.

2 Samson Abramsky. Computational interpretations of linear logic. *Theoretical Computer Science*, 111:3–57, 1993.

3 Beniamino Accattoli, Pablo Barenbaum, and Damiano Mazza. Distilling abstract machines. In J. Jeuring and M. Chakravarty, editors, *19th International Conference on Functional Programming (ICFP 2014)*, pages 363–376, Gothenburg, Sweden, September 2014. ACM. Long version available at `arXiv:1406.2370`.

4 Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):197–347, 1992.

5 Robert Atkey. Syntax and semantics of quantitative type theory. In Anuj Dawar and Erich Grädel, editors, *33rd Conference on Logic in Computer Science (LICS 2018)*, pages 56–65, Oxford, UK, July 2018. ACM.

6 Andrew Barber. Dual intuitionistic linear logic. Technical Report ECS-LFCS-96-347, Department of Computer Science, University of Edinburgh, September 1996.

7 P. N. Benton. A mixed linear and non-linear logic: Proofs, terms and models. In Leszek Pacholski and Jerzy Tiuryn, editors, *Selected Papers from the 8th International Workshop on Computer Science Logic (CSL'94)*, pages 121–135, Kazimierz, Poland, September 1994. Springer LNCS 933. An extended version appears as Technical Report UCAM-CL-TR-352, University of Cambridge.

8 P. N. Benton. A mixed linear and non-linear logic: Proofs, terms and models (preliminary report). Technical Report UCAM-CL-TR-352, University of Cambridge, October 1994. URL: `https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-352.html`.

9 P. N. Benton, G. M. Bierman, and V. C. V. de Paiva. Computational types from a logical perspective. *Journal of Functional Programming*, 8(2):177–193, March 1998.

10 P. N. Benton, Gavin M. Bierman, Martin Hyland, and Valeria de Paiva. A term calculus for intuitionistic linear logic. In *International Conference on Typed Lambda Calculi and Applications*, pages 75–90, Utrecht, The Netherlands, 1993. Springer LNCS 664.

11 P. N. Benton and Philip Wadler. Linear logic, monads, and the lambda calculus. In E. Clarke, editor, *Proceedings of the 11th Annual Symposium on Logic in Computer Science*, pages 420–431, New Brunswick, New Jersey, July 1996. IEEE Computer Society Press.

12 Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In *Proceedings of the 21st International Conference on Concurrency Theory (CONCUR 2010)*, pages 222–236, Paris, France, August 2010. Springer LNCS 6269.

13 Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logic propositions as session types. *Mathematical Structures in Computer Science*, 26(3):367–423, 2016. Special Issue on Behavioural Types.

14 Iliano Cervesato, Joshua S. Hodas, and Frank Pfenning. Efficient resource management for linear logic proof search. *Theoretical Computer Science*, 232(1–2):133–163, February 2000. Special issue on Proof Search in Type-Theoretic Languages, D. Galmiche and D. Pym, editors.

15 Bor-Yuh Evan Chang, Kaustuv Chaudhuri, and Frank Pfenning. A judgmental analysis of linear logic. Technical Report CMU-CS-03-131R, Carnegie Mellon University, Department of Computer Science, December 2003.

16 Kaustuv Chaudhuri. Classical and intuitionistic subexponential logics are equally expressive. In *Computer Science Logic*, pages 185–199. Springer LNCS 6247, August 2010.

17 Pritam Choudhury, Harley Eades III, Richard A. Eisenberg, and Stephanie Weirich. A graded dependent type system with a usage-aware semantics. *Proc. ACM Program. Lang.*, 5(POPL'21):1–32, 2021.

18 Alonzo Church. *The Calculi of Lambda-Conversion*, volume 6 of *Annals of Mathematics Studies*. Princeton University Press, 1941.

19 Pierre-Louis Curien, Marcelo P. Fiore, and Guillaume Munch-Maccagnoni. A theory of effects and resources: Adjunction models and polarised calculi. In R. Bodík and R. Majumdar, editors, *Proceedings of the 43rd Symposium on Principles of Programming Languages (POPL 2016)*, pages 44–56, St. Petersburgh, Florida, USA, January 2016. ACM.
20 Vincent Danos, Jean Baptiste Joinet, and Harold Schellinx. The structure of exponentials: Uncovering the dynamics of linear logic proofs. In Georg Gottlob, Alexander Leitsch, and Daniele Mundici, editors, *Computational Logic and Proof Theory*, pages 159–171, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.
21 Michael Dummett. *The Logical Basis of Metaphysics.* Harvard University Press, Cambridge, Massachusetts, 1991. The William James Lectures, 1976.
22 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.
23 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
24 Jean-Yves Girard and Yves Lafont. Linear logic and lazy computation. In H. Ehrig, R. Kowalski, G. Levi, and U. Montanari, editors, *Proceedings of the International Joint Conference on Theory and Practice of Software Development*, volume 2, pages 52–66, Pisa, Italy, March 1987. Springer-Verlag LNCS 250.
25 Peter Hanukaev and Harley Eades, III. Combining dependency, grades, and adjoint logic. In *8th Workshop on Type-Driven Development (TyDe 2023)*, pages 58–70, Seattle, Washington, 2023. ACM. `arXiv:2307.09563`.
26 Robert Harper. *Practical Foundations for Programming Languages.* Cambridge University Press, second edition, April 2016.
27 W. A. Howard. The formulae-as-types notion of construction. Unpublished note. An annotated version appeared in: *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, 479–490, Academic Press (1980), 1969.
28 Jack Hughes and Dominic Orchard. Resourceful program synthesis from graded linear types. In M. Fernández, editor, *30th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR 2020)*, pages 151–170, Bologna, Italy, September 2020. Springer LNCS 12561.
29 Max Kanovich, Stepan Kuznetsov, Vivek Nigam, and Andre Scedrov. Subexponentials in non-commutative linear logic. *ArXiv e-prints*, September 2017. `arXiv:1709.03607`.
30 Max Kanovich, Stepan Kuznetsov, Vivek Nigam, and Andre Scedrov. A logical framework with commutative and non-commutative subexponentials. In *International Joint Conference on Automated Reasoning (IJCAR 2018)*, pages 228–245. Springer LNAI 10900, 2018.
31 Stephen Cole Kleene. *Introduction to Metamathematics.* North-Holland, 1952.
32 John Launchbury. A natural semantics for lazy evaluation. In *20th Annual Symposium on Principles of Programming Languages (POPL 1993)*, pages 144–154, Charleston, South Carolina, January 1993. ACM.
33 Paul Blain Levy. Call-by-push-value: Decomposing call-by-value and call-by-name. *Higher-Order and Symbolic Computation*, 19(4):377–414, 2006.
34 Daniel R. Licata and Michael Shulman. Adjoint logic with a 2-category of modes. In *International Symposium on Logical Foundations of Computer Science (LFCS)*, pages 219–235. Springer LNCS 9537, January 2016.
35 Daniel R. Licata, Michael Shulman, and Mitchell Riley. A fibrational framework for substructural and modal logics. In Dale Miller, editor, *Proceedings of the 2nd International Conference on Formal Structures for Computation and Deduction (FSCD'17)*, pages 25:1–25:22, Oxford, UK, September 2017. LIPIcs.
36 Per Martin-Löf. On the meanings of the logical constants and the justifications of the logical laws. Notes for three lectures given in Siena, Italy. Published in *Nordic Journal of Philosophical Logic*, 1(1):11-60, 1996, April 1983. URL: `http://www.hf.uio.no/ifikk/forskning/publikasjoner/tidsskrifter/njpl/vol1no1/meaning.pdf`.

37 Conor McBride. I got plenty o' nuttin'. In Sam Lindley, Conor McBride, Phil Trinder, and Don Sannella, editors, *A List of Successes That can Change the World—Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday*, pages 207–233. Springer LNCS 9600, 2016.

38 Benjamin Moon, Harley Eades III, and Dominic Orchard. Graded modal dependent type theory. In *30th European Symposium on Programming (ESOP'21)*, volume 12648 of *Lecture Notes in Computer Science*, pages 462–490. Springer, 2021.

39 Alan Mycroft. The theory and practice of transforming call-by-need into call-by-value. In *4th International Symposium on Programming*, pages 269–281. Springer LNCS 83, 1980.

40 Vivek Nigam and Dale Miller. Algorithmic specifications in linear logic with subexponentials. In *Proceedings of the 11th International Conference on Principles and Practice of Declarative Programming (PPDP)*, pages 129–140, Coimbra, Portugal, September 2009. ACM.

41 Vivek Nigam, Elaine Pimental, and Giselle Reis. An extended framework for specifying and reasoning about proof systems. *Journal of Logic and Computation*, 26(2):539–576, 2016.

42 Frank Pfenning and Rowan Davies. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science*, 11:511–540, 2001. Notes to an invited talk at the *Workshop on Intuitionistic Modal Logics and Applications* (IMLA'99), Trento, Italy, July 1999.

43 Frank Pfenning and Klaas Pruiksma. Relating message passing and shared memory, proof-theoretically. In S. Jongmans and A. Lopes, editors, *25th International Conference on Coordination Models and Languages (COORDINATION 2023)*, pages 3–27, Lisbon, Portugal, June 2023. Springer LNCS 13908. Notes to an invited talk.

44 Dag Prawitz. *Natural Deduction*. Almquist & Wiksell, Stockholm, 1965.

45 Klaas Pruiksma. *Adjoint Logic with Applications*. PhD thesis, Carnegie Mellon University, 2024. In preparation.

46 Klaas Pruiksma, William Chargin, Frank Pfenning, and Jason Reed. Adjoint logic. Unpublished manuscript, April 2018. URL: `http://www.cs.cmu.edu/~fp/papers/adjoint18b.pdf`.

47 Klaas Pruiksma and Frank Pfenning. A message-passing interpretation of adjoint logic. *Journal of Logical and Algebraic Methods in Programming*, 120(100637), 2021.

48 Klaas Pruiksma and Frank Pfenning. Back to futures. *Journal of Functional Programming*, 32:e6, 2022.

49 Jason Reed. A judgmental deconstruction of modal logic. Unpublished manuscript, May 2009. URL: `http://www.cs.cmu.edu/~jcreed/papers/jdml2.pdf`.

50 The Rust programming language. Available at `https://www.rust-lang.org/`. Accessed January 15, 2024.

51 Jesse A. Tov and Riccardo Pucella. Practical affine types. In T. Ball and M. Sagiv, editors, *Proceedings of the 38th Symposium on Principles of Programming Languages (POPL 2011)*, pages 447–458. ACM Press, January 2011.

52 Anne S. Troelstra. Natural deduction for intuitionistic linear logic. *Annals of Pure and Applied Logic*, 73(1):79–108, 1995.

53 Philip Wadler. Linear types can change the world. In *IFIP TC*, volume 2, pages 347–359, 1990.

54 Philip Wadler. Propositions as sessions. In *Proceedings of the 17th International Conference on Functional Programming (ICFP 2012)*, pages 273–286, Copenhagen, Denmark, September 2012. ACM Press.

55 James Wood and Robert Atkey. A framework for substructural type systems. In Ilya Sergey, editor, *31st European Symposium on Programming (ESOP 2022)*, pages 376–402, Munich, Germany, April 2022. Springer LNCS 13240.