Adjoint Logic and Its Concurrent Operational Interpretation

Klaas Pruiksma
Computer Science Department
Carnegie Mellon University
kpruiksm@andrew.cmu.edu

William Chargin
Computer Science Department
Carnegie Mellon University
wchargin@gmail.com

Frank Pfenning
Computer Science Department
Carnegie Mellon University
fp@cs.cmu.edu

Jason Reed
Wesleyan University
jcreed@gmail.com

ABSTRACT

Adjoint logic provides a schematic way to combine multiple logics, some of which may be substructural, through modal operators that are adjoint to each other. We provide a simple formulation for adjoint logic with explicit structural rules. The adjoint logic is parameterized by a preorder of modes of truth characterizing (potential) dependence between the modes. We demonstrate that suitable choices of this preorder allow us to directly embed various logics including lax logic, judgmental S4, LNL, and intuitionistic subexponential linear logic into adjoint logic.

Under the proofs-as-programs paradigm, proofs correspond to concurrent processes and cut reduction to synchronous communication. We show how to restructure the sequent calculus so that cut reduction entails asynchronous communication and give an operational interpretation that provides session-typed communication extended with multicast and distributed garbage collection.

1 INTRODUCTION

How do we combine logics? One approach is to embed a less expressive into a more expressive logic. This is the approach, for example, taken by Girard [1987] who represents the usual intuitionistic implication $A \to B$ as linear implication $!A \multimap B$ through the use of the exponential modality $!A$ that controls weakening and contraction. The rules of the source logic then become derived or admissible rules in the target logic. If we are interested in the computational interpretation of proofs via proof reduction, we then have to reconsider or reconstruct the meaning through the translation.

An alternative is to keep the original logics intact and provide modal operators we call *shifts* to switch between them. This is the approach, for example, taken by Benton [1994]. As we will see, Girard's approach can be seen as a special case of Benton's.

Of course, the properties satisfied by the shifts cannot be arbitrary or the result will not be a proper combination of the two logics. In this paper we restrict our attention to intuitionistic logics and, in particular, we take the verificationist perspective [Dummett 1991; Gentzen 1935] where the meanings of the logical connectives in each logic are defined by the left and right rules of the sequent calculus. Cut elimination and identity expansion are necessary to justify this point of view. As we will see in Section 4, taking the intuitionistic point of view will provide the opportunity for a uniform operational interpretation of all logics under consideration.

All logics we consider satisfy associativity and exchange among the antecedents, and may or may not satisfy weakening or contraction. We identify a logic by its *mode of truth* m and write $\sigma(m) \subseteq \{\mathrm{W}, \mathrm{C}\}$ for the structural rules satisfied by mode m. We use the same definition for the logical connectives at all modes. For example, $A_m \multimap_m B_m$ denotes implication, which could be linear ($\sigma(m) = \{\,\}$), structural ($\sigma(m) = \{\mathrm{W}, \mathrm{C}\}$), affine ($\sigma(m) = \{\mathrm{W}\}$), or strict ($\sigma(m) = \{\mathrm{C}\}$). We often drop the subscript on the logical connective when it can be uniquely determined from context.

We give a variation of Reed's first and unpublished definition of adjoint logic [Reed 2009] by using explicit structural rules where allowed by the mode and just a single pair of left and right rules for each of the logical connectives and shifts. This formulation allows an elegant proof of cut elimination closely modeled upon Gentzen's original proof [Gentzen 1935] using the rule of *multicut*. Cut elimination immediately yields a conservative extension result for the combined logic over all of its modes of truth. We then annotate sequents with process expressions extending prior work by Caires and Pfenning [2010]; Caires et al. [2016] and Pfenning and Griffith [2015]. Pleasingly, in this formulation the process expressions for the analogous connectives at different modes have exactly the same simple form. Modes satisfying contraction permit multicast when sending. Moreover, we find that separating out the structural rules together with several logical transformations exploiting multicut leads to a precise operational semantics in which there are no untethered processes that need to be garbage-collected at the end of a computation despite the presence of weakening and contraction.

We now introduce our formulation of adjoint logic, including proofs of cut elimination and identity expansion (Section 2); define process expressions, define typing, and provide an operational semantics (Section 4); and prove preservation and progress (Section 5). We close with some remarks on related work (Section 6) and a brief conclusion.

2 ADJOINT LOGIC

Adjoint logic can be thought of as a schema to define particular logics. The schema is parameterized by a set of modes of truth m, where each proposition and logical connective is indexed by its mode. Furthermore, each mode intrinsically carries a set of structural properties $\sigma(m) \subseteq \{\mathrm{W}, \mathrm{C}\}$ where W stands for *weakening* and

Under Submission, January 2018,
2018. ACM ISBN ...$15.00
https://doi.org/

C stands for *contraction*. As a concession to simplicity of the presentation, in this paper we always allow exchange, although nothing stands in the way of an even more general framework [Pfenning 2016]. In addition, an instance requires a preorder between modes, where $m \geq k$ expresses that the proof of a proposition of mode k may depend on a hypotheses of mode m. This preorder embodies the *declaration of independence*:

> *A proof of A_k may only depend on hypotheses B_m for $m \geq k$.*

The form of a sequent is

$$\Psi \vdash A_k \quad \text{where } \Psi \geq k$$

where Ψ is a collection of *antecedents* of the form $(x_i : B^i_{m_i})$ with each $m_i \geq k$, where all the variables x_i are distinct. This critical presupposition is abbreviated as $\Psi \geq k$. Furthermore, the order of the antecedents does not matter since we always allow exchange.

In addition, we require the preorder between modes to be compatible with their structural properties: that is, $m \geq k$ implies $\sigma(m) \supseteq \sigma(k)$. This is necessary to guarantee cut elimination (see Example 2.4).

Finally, we may define fragments by restricting the set of propositions we consider for a given mode.

The propositions at each mode are constructed uniformly, remaining within the same mode, except for the *shift operators* that move between modes. They are $\uparrow^m_k A_k$ (pronounced *up*), which is a proposition at mode m and requires $m \geq k$; and $\downarrow^\ell_m A_\ell$ (*down*), which is also a proposition at mode m, and which requires $\ell \geq m$.

At this point we can already write out the syntax of propositions.

$$\begin{array}{rcl} A_m, B_m & ::= & p_m \mid A_m \multimap_m B_m \mid A_m \otimes_m B_m \mid \mathbf{1}_m \\ & & \mid \bigoplus_{i \in I}{}_m A^i_m \mid \&_{i \in I}{}_m A^i_m \mid \uparrow^m_k A_k \mid \downarrow^\ell_m A_\ell \end{array}$$

Here p_m stands for atomic propositions at mode m. Anticipating the needs of our operational interpretation, we have generalized internal and external choice to n-ary constructors parameterized by an index set I. So we write $A^1_m \oplus A^2_m = \bigoplus_{i \in \{1,2\}} A^i_m$.

Remarkably, the right and left rules in the sequent calculus defining the logical connectives are the same for each mode and are complemented by the permissible structural rules.

2.1 Judgmental and structural rules

The rules for adjoint logic can be found in Figure 1. We begin with the judgmental rules of identity and cut, which express the connection between antecedents and succedents. Identity says that if we assume A_m we are allowed to conclude A_m. Cut says the opposite: if we can conclude A_m we are allowed to assume A_m *as long as the declaration of independence is respected.*

As is common for the sequent calculus, we read the rules in the direction of bottom-up proof construction. This is also the direction of type checking, once we have assigned process expressions to the judgments. For the cut rule, this means we should assume that the conclusion $\Psi\ \Psi' \vdash C_k$ is well-formed and, in particular, that $\Psi \geq k$ and $\Psi' \geq k$. Therefore, if we check that $m \geq k$, then we know that the second premise, $(x : A_m)\ \Psi' \vdash C_k$, will also be well-formed. For the first premise to be well-formed, we need to check outright that $\Psi \geq m$.

The structural rules of weakening and contraction just need to verify that the mode of the principal formula permits the rule.

2.2 Additive and multiplicative connectives

The logical rules defining the additive and multiplicative connectives are simply the linear rules for all modes, since we have separated out the structural rules. Except in one case, $\multimap L$, the well-formedness of the conclusion implies the well-formedness of all premises.

As for $\multimap L$, we know from the well-formedness of the conclusion that $\Psi \geq k$, $\Psi' \geq k$, and $m \geq k$. These facts by themselves already imply the well-formedness of the second premise, but we need to check that $\Psi' \geq m$ in order for the first premise to be well-formed.

2.3 Shifts

The shifts represent the most interesting aspects of the rules. Recall that in $\uparrow^m_k A_k$ and $\downarrow^m_k A_m$ we require that $m \geq k$. We first consider the two rules for $\uparrow$. We know from the conclusion of the right rule that $\Psi \geq m$ and from the requirement of the shift that $m \geq k$. Therefore, as $\geq$ is transitive, $\Psi \geq k$ and the premise is always well-formed. This also means (although we do not prove it here) that this rule is *invertible*.

From the conclusion of the left rule, we know $\Psi \geq \ell$, $m \geq \ell$, and $m \geq k$. This does not imply that $k \geq \ell$, which we need for the premise to be well-formed and thus needs to be checked. Therefore, this rule is non-invertible.

The downshift rules are constructed analogously, taking only the declaration of independence and properties of the preorder $\leq$ as guidance. Note that in this case the left rule is always applicable (that is, invertible), while the right rule is non-invertible.

2.4 Logic Examples

We now describe how adjoint logic can be used to embed various other logics, and provide some examples to justify our presentation.

Example 2.1 (Linear logic). We obtain intuitionistic linear logic [Barber 1996; Girard 1987] by using two modes, U (for *structural*) and L (for *linear*) with U > L. Moreover, $\sigma(\mathsf{U}) = \{\mathsf{W}, \mathsf{C}\}$ and $\sigma(\mathsf{L}) = \{\,\}$, and the structural layer contains only the shifted proposition.

$$\begin{array}{rcl} A_\mathsf{U} & ::= & \uparrow^\mathsf{U}_\mathsf{L} A_\mathsf{L} \\ A_\mathsf{L}, B_\mathsf{L} & ::= & p_\mathsf{L} \mid A_\mathsf{L} \multimap B_\mathsf{L} \mid A_\mathsf{L} \otimes B_\mathsf{L} \mid \mathbf{1} \mid \bigoplus_{i \in I} A^i_\mathsf{L} \mid \&_{i \in I} A^i_\mathsf{L} \mid \downarrow^\mathsf{U}_\mathsf{L} A_\mathsf{U} \end{array}$$

In this representation the exponential modality is decomposed into shift modalities $!A_\mathsf{L} = \downarrow^\mathsf{U}_\mathsf{L} \uparrow^\mathsf{U}_\mathsf{L} A_\mathsf{L}$. Unlike Chang et al. [2003], our sequent calculus employs explicit structural rules of weakening and contraction on unrestricted propositions A_U. We do not state an explicit correctness theorem because it follows from the embedding of LNL (Theorem 2.3) and Benton's results [Benton 1994].

Example 2.2 (LNL). We obtain LNL [Benton 1994] just like linear logic with two modes U > L, but we populate the unrestricted layer with additional propositions, where we write $\times = \otimes_\mathsf{U}$ and $\to = \multimap_\mathsf{U}$.

$$\begin{array}{rcl} A_\mathsf{U}, B_\mathsf{U} & ::= & p_\mathsf{U} \mid A_\mathsf{U} \to B_\mathsf{U} \mid A_\mathsf{U} \times B_\mathsf{U} \mid 1_\mathsf{U} \mid \uparrow^\mathsf{U}_\mathsf{L} A_\mathsf{L} \\ A_\mathsf{L}, B_\mathsf{L} & ::= & p_\mathsf{L} \mid A_\mathsf{L} \multimap B_\mathsf{L} \mid A_\mathsf{L} \otimes B_\mathsf{L} \mid \mathbf{1}_\mathsf{L} \mid \downarrow^\mathsf{U}_\mathsf{L} A_\mathsf{U} \end{array}$$

Benton's notation for shifts is $F = \downarrow^\mathsf{U}_\mathsf{L}$ and $G = \uparrow^\mathsf{U}_\mathsf{L}$. Our formulation then combines the various versions of the rules by combining

$$\frac{}{(x : A_m) \vdash A_m}\ \mathsf{id} \qquad \frac{\Psi \geq m \geq k \quad \Psi \vdash A_m \quad (x : A_m)\ \Psi' \vdash C_k}{\Psi\ \Psi' \vdash C_k}\ \mathsf{cut}$$

$$\frac{\mathsf{W} \in \sigma(m) \quad \Psi \vdash C_k}{\Psi\ (x : A_m) \vdash C_k}\ \mathsf{weaken} \qquad \frac{\mathsf{C} \in \sigma(m) \quad \Psi\ (y : A_m)\ (z : A_m) \vdash C_k}{\Psi\ (x : A_m) \vdash C_k}\ \mathsf{contract}$$

$$\frac{\ell \in I \quad \Psi \vdash A^\ell_m}{\Psi \vdash \bigoplus_{i \in I} A^i_m}\ \oplus R_\ell \qquad \frac{\Psi\ (y : A^i_m) \vdash C_k \text{ for each } i \in I}{\Psi\ (x : \bigoplus_{i \in I} A^i_m) \vdash C_k}\ \oplus L \qquad \frac{\Psi \vdash A^i_m \text{ for each } i \in I}{\Psi \vdash \mathop{\&}_{i \in I} A^i_m}\ \& R \qquad \frac{\ell \in I \quad \Psi\ (y : A^\ell_m) \vdash C_k}{\Psi\ (x : \mathop{\&}_{i \in I} A^i_m) \vdash C_k}\ \& L_\ell$$

$$\frac{\Psi \vdash A_m \quad \Psi' \vdash B_m}{\Psi\ \Psi' \vdash A_m \otimes B_m}\ \otimes R \qquad \frac{\Psi\ (y : A_m)\ (z : B_m) \vdash C_k}{\Psi\ (x : A_m \otimes B_m) \vdash C_k}\ \otimes L \qquad \frac{}{\cdot \vdash \mathbf{1}_m}\ \mathbf{1}R \qquad \frac{\Psi \vdash C_k}{\Psi\ (x : \mathbf{1}_m) \vdash C_k}\ \mathbf{1}L$$

$$\frac{(x : A_m)\ \Psi \vdash B_m}{\Psi \vdash A_m \multimap B_m}\ \multimap R \qquad \frac{\Psi' \geq m \quad \Psi' \vdash A_m \quad \Psi\ (y : B_m) \vdash C_k}{\Psi\ \Psi'\ (x : A_m \multimap B_m) \vdash C_k}\ \multimap L$$

$$\frac{\Psi \vdash A_k}{\Psi \vdash \uparrow^m_k A_k}\ \uparrow R \qquad \frac{k \geq \ell \quad \Psi\ (y : A_k) \vdash C_\ell}{\Psi\ (x : \uparrow^m_k A_k) \vdash C_\ell}\ \uparrow L \qquad \frac{\Psi \geq m \quad \Psi \vdash A_m}{\Psi \vdash \downarrow^m_k A_m}\ \downarrow R \qquad \frac{\Psi\ (y : A_m) \vdash C_\ell}{\Psi\ (x : \downarrow^m_k A_m) \vdash C_\ell}\ \downarrow L$$

Figure 1: Rules of Adjoint Logic

the two contexts, using the declaration of independence instead to force that unrestricted succedents depend only on unrestricted antecedents. A small difference arises only in the $\times$-*left* rules where our version has both components in the premise, which is of course logically equivalent to LNL in the presence of weakening and contraction.

Theorem 2.3. *If we let τ embed propositions of LNL into the instance of adjoint logic described above, then*

(a) $\Theta \vdash_C X$ in LNL iff $\tau(\Theta) \vdash \tau(X)$ in adjoint logic.
(b) $\Theta; \Gamma \vdash_{\mathcal{L}} A$ in LNL iff $\tau(\Theta), \tau(\Gamma) \vdash \tau(A)$ in adjoint logic.

Example 2.4 (Counterexample for independence). Consider linear logic or LNL and consider the following **faulty(!)** "proof" showing that contraction for linear propositions is derivable:

$$\dfrac{\dfrac{}{(x : A_\mathsf{L}) \vdash A_\mathsf{L}}\ \mathsf{id}}{\dfrac{(x : A_\mathsf{L}) \vdash^{??} \uparrow^\mathsf{U}_\mathsf{L} A_\mathsf{L}}{(x : A_\mathsf{L}) \vdash C_\mathsf{L}}\ \mathsf{cut}}\ \uparrow R \qquad \dfrac{\dfrac{\dfrac{(y : A_\mathsf{L})\ (z : A_\mathsf{L}) \vdash C_\mathsf{L}}{(y : A_\mathsf{L})\ (w : \uparrow^\mathsf{U}_\mathsf{L} A_\mathsf{L}) \vdash C_\mathsf{L}}\ \uparrow L}{(v : \uparrow^\mathsf{U}_\mathsf{L} A_\mathsf{L})\ (w : \uparrow^\mathsf{U}_\mathsf{L} A_\mathsf{L}) \vdash C_\mathsf{L}}\ \uparrow L}{(u : \uparrow^\mathsf{U}_\mathsf{L} A_\mathsf{L}) \vdash C_\mathsf{L}}\ \mathsf{contract}$$

The fallacy lies with the sequent marked $\vdash^{??}$ because it violates our declaration of independence: the succedent $\uparrow^\mathsf{U}_\mathsf{L} A_\mathsf{L}$ of mode U depends on an antecedent of mode L, and $\mathsf{L} \not\geq \mathsf{U}$.

If we wanted to blame a particular inference, it would be either cut, viewed bottom-up, or $\uparrow R$, viewed top-down. In our case, the bottom-up construction of this proof would fail because the condition $(x : A_\mathsf{L}) \geq \mathsf{U}$ of the cut rule is violated.

It is an immediate corollary that cut elimination fails if the declaration of independence is not enforced. For example, using the above faulty reasoning, we could prove $A_\mathsf{L} \vdash A_\mathsf{L} \otimes A_\mathsf{L}$, which in general has no cut-free proof.

Example 2.5 (Judgmental S4). The ($\Diamond$-free portion of) judgmental modal logic S4 [Pfenning and Davies 2001] arises from two modes V (validity) and U (truth) with $\mathsf{V} > \mathsf{U}$. The declaration of independence here expresses that *validity is categorical with respect to truth*—that is, a proof of A_V may not depend on any hypotheses of the form B_U. Previously, this had been enforced by segregating the antecedents into two zones and managing their dependence accordingly.

$$\begin{array}{lcl} A_\mathsf{V} & ::= & \uparrow^\mathsf{V}_\mathsf{U} A_\mathsf{U} \\ A_\mathsf{U}, B_\mathsf{U} & ::= & p_\mathsf{U} \mid A_\mathsf{U} \multimap B_\mathsf{U} \mid A_\mathsf{U} \otimes B_\mathsf{U} \mid \mathbf{1} \mid \bigoplus_{i \in I} A^i_\mathsf{U} \mid \mathop{\&}_{i \in I} A^i_\mathsf{U} \mid \downarrow^\mathsf{V}_\mathsf{U} A_\mathsf{V} \end{array}$$

Analogous to the encoding of linear logic, we only need to allow $\uparrow^\mathsf{V}_\mathsf{U} A_\mathsf{U}$ in the validity layer. Under that interpretation, we encode $\Box A_\mathsf{U} = \downarrow^\mathsf{V}_\mathsf{U} \uparrow^\mathsf{V}_\mathsf{U} A_\mathsf{U}$, which is entirely analogous to the representation of $!A$ in linear logic.

The adjoint reconstruction now gives rise to a richer logic where additional connectives speaking about validity can be decomposed directly via their left and right rules.

Theorem 2.6. *If we let τ embed propositions of judgmental S4 into the instance of adjoint logic described above, then*

(a) $\Delta; \Gamma \vdash A$ in judgmental S4 iff $\uparrow^\mathsf{V}_\mathsf{U} \tau(\Delta), \tau(\Gamma) \vdash \tau(A)$ in adjoint logic.
(b) $\Delta; \cdot \vdash A$ in judgmental S4 iff $\uparrow^\mathsf{V}_\mathsf{U} \tau(\Delta) \vdash \uparrow^\mathsf{V}_\mathsf{U} \tau(A)$ in adjoint logic.

Example 2.7 (Lax logic). Lax logic [Fairtlough and Mendler 1997; Pfenning and Davies 2001] encodes a weaker form of truth called *lax truth*. We can represent it as a substructural adjoint logic with two modes, $\mathsf{U} > \mathsf{X}$, where both modes satisfy weakening and contraction. We restrict the lax layer to a single connective and omit additive connectives for simplicity.

$$\begin{array}{lcl} A_\mathsf{U}, B_\mathsf{U} & ::= & p_\mathsf{U} \mid A_\mathsf{U} \to B_\mathsf{U} \mid A_\mathsf{U} \times B_\mathsf{U} \mid 1_\mathsf{U} \mid \uparrow^\mathsf{U}_\mathsf{X} A_\mathsf{X} \\ A_\mathsf{X} & ::= & \downarrow^\mathsf{U}_\mathsf{X} A_\mathsf{U} \end{array}$$

Now the lax modality is defined as $\bigcirc A_\mathsf{U} = \uparrow^\mathsf{U}_\mathsf{X} \downarrow^\mathsf{U}_\mathsf{X} A_\mathsf{U}$.

We can now add further connectives directly operating on the lax layer and obtain consistent left and right rules for them.

Theorem 2.8. *If we let τ embed propositions of lax logic into the instance of adjoint logic described above, then*

(a) $\Gamma \vdash A$ true in lax logic iff $\tau(\Gamma) \vdash \tau(A)$ in adjoint logic.
(b) $\Gamma, \Gamma' \vdash A$ lax in lax logic iff $\tau(\Gamma), \downarrow^\mathsf{U}_\mathsf{X} \tau(\Gamma') \vdash \downarrow^\mathsf{U}_\mathsf{X} \tau(A)$ in adjoint logic.

2.5 Multicut

Because we have an explicit rule of contraction, cut elimination does not follow by a simple structural induction. However, we can follow Gentzen [1935] and allow multiple copies of the same proposition to be removed by the cut, which then allows a structural induction argument. In anticipation of the operational interpretation, we have labeled our antecedents with unique variables, so the generalized form of cut called *multicut* (see, for example, Negri and von Plato [2001]) can remove $n \geq 0$ copies. Of course, such cuts are only legal if the propositions that are removed satisfy the necessary structural rules. For $n = 0$, we require that the mode m support weakening.

$$\frac{\Psi \geq m \geq k \quad \mathsf{W} \in \sigma(m) \quad \Psi \vdash A_m \quad \Psi' \vdash C_k}{\Psi\ \Psi' \vdash C_k}\ \mathsf{cut}(\emptyset)$$

For $n = 1$, we obtain the usual cut rule and no special requirements are needed.

$$\frac{\Psi \geq m \geq k \quad \Psi \vdash A_m \quad (x : A_m)\ \Psi' \vdash C_k}{\Psi\ \Psi' \vdash C_k}\ \mathsf{cut}(\{x\})$$

For $n \geq 2$, the mode of the cut formula must admit contraction.

$$\frac{\begin{array}{l}\mathsf{C} \in \sigma(m) \\ \Psi \geq m \geq k \quad \Psi \vdash A_m \quad (S \cup \{x, y\} : A_m)\ \Psi' \vdash C_k\end{array}}{\Psi\ \Psi' \vdash C_k}\ \mathsf{cut}(S \cup \{x, y\})$$

Here, we have used the abbreviation $(\{x_1, \ldots, x_n\} : A_m)$ to stand for $(x_1 : A_m) \ldots (x_n : A_m)$.

Note that each of these rules has a side condition that can be interpreted informally as stating that the number of antecedents cut must be compatible with the mode m: if there are no antecedents removed, m must admit weakening, and if we remove two or more, m must admit contraction. We write this as $|S| \sim m$ where $0 \sim m$ if $\mathsf{W} \in \sigma(m)$, $1 \sim m$ always, and $k \sim m$ for $k \geq 2$ if $\mathsf{C} \in \sigma(m)$.

This allows us to write down a single rule encompassing all three of the above cases for multicut:

$$\frac{\Psi \geq m \geq k \quad |S| \sim m \quad \Psi \vdash A_m \quad (S : A_m)\ \Psi' \vdash C_k}{\Psi\ \Psi' \vdash C_k}\ \mathsf{cut}(S)$$

Note that the standard cut rule is the instance of the multicut rule where $|S| = 1$, and so proving multicut elimination for adjoint logic also yields cut elimination for the standard cut rule.

2.6 Identity Expansion and Cut Elimination

We present standard identity expansion and cut elimination results as evidence for the correctness of the sequent calculus as capturing the meaning of the logical connectives via their inference rules. Cut-free proofs will always decompose propositions when read from conclusion to premise and thus yield a conservative extension result. Finally, the fine detail of the proof is significant because (a) the cut reductions, which constitute the essence of the proof, are the basis for the operational semantics, and (b) cut reductions define a proof equivalence under which the adjunction property for the shifts can be verified (see Section 2.7).

Theorem 2.9 (Identity Expansion). *If $\Psi \vdash A_m$, then there exists a proof that $\Psi \vdash A_m$ using identity rules only at atomic propositions, which is cut-free if the original proof is.*

Proof. We begin by proving that for any formula A_m, there is a cut-free proof that $(x : A_m) \vdash A_m$ using identity rules only at atomic propositions. This follows easily from an induction on A_m.

Now, we arrive at the theorem by induction over the structure of the given proof that $\Psi \vdash A_m$. □

We use $\Psi \Vdash A_m$ to stand for the statement that there is a cut-free proof of A_m from Ψ.

Theorem 2.10 (Cut Elimination). *If $\Psi \vdash A_m$, then also $\Psi \Vdash A_m$.*

Proof. This proof follows the structure of many cut-elimination results. First we prove admissibility of multicut in the cut-free system: if $\mathcal{D}$ is a proof of $\Psi \Vdash A_m$ and $\mathcal{E}$ is a proof of $(S : A_m)\ \Psi' \Vdash C_k$, then we can construct a proof of $\Psi\ \Psi' \Vdash C_k$. This is established by a straightforward nested induction, first on the proposition A_m and then simultaneously on the structure of the deductions $\mathcal{D}$ and $\mathcal{E}$. This is followed by a simple structural induction to prove cut elimination, using the admissibility of cut when it is encountered. If we ignore the modes, this proof is very similar to the original proof of Gentzen [1935]. Some sample cases are provided in Appendix A.2. □

Corollary 2.11. *Adjoint logic is a conservative extension of each of the logics at a fixed mode. That is, if $\Psi \vdash A_m$ is a sequent purely at mode m (in that every type in Ψ is at mode m and neither A_m nor the types in Ψ make use of shifts), then $\Psi \vdash A_m$ is provable using the rules of adjoint logic iff it is provable using the rules which define the logic at mode m.*

Proof. By cut elimination, we have that if $\Psi \vdash A_m$ is provable in the adjoint logic, then so is $\Psi \Vdash A_m$. By the subformula property of cut-free proofs, this cannot leave mode m. □

2.7 Adjunction properties

As yet, we have not discussed the meaning of the name "*adjoint logic*". This can be justified by showing that for fixed $k \leq m$, $\downarrow_k^m$ and $\uparrow_k^m$ yield an adjoint pair of functors $\downarrow_k^m \dashv \uparrow_k^m$. Since prior results (see Benton [1994] and Licata et al. [2017]) already establish this property and we have little new to contribute here, a sketch of this construction is relegated to Appendix A.3.

3 ASYNCHRONOUS ADJOINT LOGIC

As has been observed before, intuitionistic and classical linear logics can be put into a Curry–Howard correspondence with session-typed communicating processes [Caires and Pfenning 2010; Caires et al. 2016; Wadler 2012]. A linear logical proposition corresponds to a session type, and a sequent proof to a process expression. The transition rules of the operational semantics derive from the cut reductions.

Under the intuitionistic interpretation a sequent proof[1] of

$$(x_1 : A_{\mathsf{L}}^1) \cdots (x_n : A_{\mathsf{L}}^n) \vdash (x : A_{\mathsf{L}})$$

corresponds to a process P that *provides* channel x and uses channels x_i. The types of the channels prescribe the pattern of communication: in the succedent, positive types ($\oplus$, $\otimes$, $\mathbf{1}$) will send and negative types ($\&$, $\multimap$) will receive. In the antecedent, the roles are reversed.

[1] for now on the linear fragment, and also labeling the succedent with a fresh variable

Cut corresponds to parallel composition of two processes, with a private channel between them, while identity simply equates two channels.

3.1 Enforcing Asynchronous Communication

Under this interpretation, a cut of a right rule against a matching left rule allows computation to proceed by mimicking the cut reduction from the proof of Theorem 2.10. For example, a cut at type $\bigoplus_{i\in I} A^i_{\mathsf{L}}$ is replaced by a cut at type A^ℓ_{L} for some $\ell \in I$. This corresponds to passing a message ('ℓ') from the process *providing* $x : \bigoplus_{i\in I} A^i_{\mathsf{L}}$ to the process *using* x. By its very nature, this form of cut reduction is *synchronous*: both provider and client proceed simultaneously because the channel $x : A_\ell$ connects the two process continuations.

For realistic languages, and also for the paradigm to smoothly extend to the case of adjoint logic where some modes permit weakening and contraction, we would like to prescribe *asynchronous communication* instead. Is this possible while remaining true to the Curry–Howard interpretation whereby computation proceeds by cut reduction? The answer is "yes", but we need to reformulate the sequent calculus.

We replace the right rules for the positive connectives $(\oplus, \otimes, \mathbf{1})$ and the left rules for the negative connectives $(\&, \multimap)$ with new zero-premise rules. In the restructuring we preserve provability, but change the nature of cut reduction.

Consider the binary case of internal choice, $A \oplus B$. Omitting extraneous antecedents, one of two usual cut reductions is

$$\dfrac{\dfrac{\dfrac{\mathcal{D}}{\vdash A}}{\vdash A \oplus B}\,\oplus R_1 \quad \dfrac{\dfrac{\mathcal{E}_1}{A \vdash C} \quad \dfrac{\mathcal{E}_2}{B \vdash C}}{A \oplus B \vdash C}\,\oplus L}{\vdash C}\,\mathsf{cut}_{A\oplus B} \quad\longrightarrow\quad \dfrac{\dfrac{\mathcal{D}}{\vdash A} \quad \dfrac{\mathcal{E}_1}{A \vdash C}}{\vdash C}\,\mathsf{cut}_A$$

Now we replace the two right rules for disjunction with two zero-premise rules ("axioms"):

$$\dfrac{}{A \vdash A \oplus B}\,\oplus R^0_1 \qquad \dfrac{}{B \vdash A \oplus B}\,\oplus R^0_2$$

Then the cut reduction above is transformed into the following:

$$\dfrac{\dfrac{}{A \vdash A \oplus B}\,\oplus R^0_1 \quad \dfrac{\dfrac{\mathcal{E}_1}{A \vdash C} \quad \dfrac{\mathcal{E}_2}{B \vdash C}}{A \oplus B \vdash C}\,\oplus L}{A \vdash C}\,\mathsf{cut}_{A\oplus B} \quad\longrightarrow\quad \dfrac{\mathcal{E}_1}{A \vdash C}$$

We see that the process representing the proof of $A \vdash A \oplus B$ acts like a message (perhaps 'inl') and the cut reduction absorbs the message, in effect entirely eliminating the cut. To actually *send* such a message we have to use a cut with the new zero-premise rule. Since a cut always proceeds by spawning a new process, this makes sending a message in effect *asynchronous*. The old rules are trivially derivable using the new ones using this extra cut.

$$\dfrac{\dfrac{\mathcal{D}}{\Psi \vdash A}}{\Psi \vdash A \oplus B}\,\oplus R_1 \quad = \quad \dfrac{\dfrac{\mathcal{D}}{\Psi \vdash A} \quad \dfrac{}{A \vdash A \oplus B}\,\oplus R^0_1}{\Psi \vdash A \oplus B}\,\mathsf{cut}_{A\oplus B}$$

In this restructured calculus (see Figure 2, ignoring for now the process terms) cut elimination fails. For example, there cannot be a cut-free proof of $\vdash \mathbf{1} \oplus B$ because no rule except cut actually applies. With cut we obtain

$$\dfrac{\dfrac{}{\vdash \mathbf{1}}\,\mathbf{1}R \quad \dfrac{}{\mathbf{1} \vdash \mathbf{1} \oplus B}\,\oplus R^0_1}{\vdash \mathbf{1} \oplus B}\,\mathsf{cut}_{\mathbf{1}}$$

This proof in fact corresponds to the parallel composition of two messages: one '$\langle\rangle$' to terminate communcation and one 'inl' to select the first alternative of the internal choice.

Fortunately, the lack of cut elimination is not troublesome, since computation rarely (if ever) corresponds to full normalization or cut elimination. Functional programming languages, for example, do not evaluate under λ-abstractions, and concurrent languages (even those proposed for the Curry–Howard interpretation [Caires and Pfenning 2010]) do not reduce under an input prefix. So we fall back on the usual progress and preservation theorems which, in the end, do derive from cut reductions, and carefully analyze the structure of irreducible configurations.

3.2 Eliminating Weakening and Contraction

We have introduced multicut entirely with the standard motivation of providing a simple proof of the admissibility of cut using structural induction. Surprisingly, we can streamline the system further by using multicut to eliminate weakening and contraction from the logic altogether.

Consider a mode m with $\mathsf{C} \in \sigma(m)$. Then contraction is a simple instance of multicut with an instance of the identity rule.

$$\dfrac{\dfrac{}{(x : A_m) \vdash A_m}\,\mathsf{id} \quad \Psi\,(y : A_m)\,(z : A_m) \vdash C_k}{\Psi\,(x : A_m) \vdash C_k}\,\mathsf{cut}(\{y, z\})$$

Similarly, for a mode m with $\mathsf{W} \in \sigma(m)$, weakening is also an instance of multicut.

$$\dfrac{\dfrac{}{(x : A_m) \vdash A_m}\,\mathsf{id} \quad \Psi \vdash C_k}{\Psi\,(x : A_m) \vdash C_k}\,\mathsf{cut}(\emptyset)$$

Cut reductions in the presence of contraction entail many residual contractions, as is evident already from Gentzen's original proof. Under our interpretation of contraction above, these residual contractions simply become multicuts with the identity. The operational interpretation of identities then plays three related roles: with one client, an identity achieves a renaming, redirecting communication; with two or more clients, an identity implements copying; with zero clients, its effect is garbage collection. The central role of identities can be seen in full detail in Figure 3, once we have introduced our notation for processes and process configurations.

4 OPERATIONAL SEMANTICS

The pattern of communication along private channels (see Section 3) is disturbed by the exponential modality $!A$ of linear logic, which requires a *shared channel* with multiple clients since it admits weakening and contraction. The only operation supported is for the client to obtain a fresh copy by sending it a fresh *linear channel* for communication at type A. This was analyzed from the perspective of adjoint logic with a fixed three-point partial order (structural,

$$\frac{}{(a : A_m) \vdash c \leftarrow a :: (c : A_m)}\ \mathsf{id} \qquad \frac{\Psi \geq m \geq k \quad |S| \sim m \quad \Psi \vdash P :: (x : A_m) \quad (S : A_m)\ \Psi' \vdash Q :: (c : C_k)}{\Psi\ \Psi' \vdash S \leftarrow (x.P); Q :: (c : C_k)}\ \mathsf{cut}(S)$$

$$\frac{\ell \in I}{(a : A^\ell_m) \vdash \mathsf{send}\ c\ \ell(a) :: (c : \oplus_{i \in I} A^i_m)}\ \oplus R^0_\ell \qquad \frac{\Psi\ (x_i : A^i_m) \vdash P_i :: (c : C_k)\ \text{for each } i \in I}{\Psi\ (a : \oplus_{i \in I} A^i_m) \vdash \mathsf{case}\ a\ \{i(x_i) \Rightarrow P_i\}_{i \in I} :: (c : C_k)}\ \oplus L$$

$$\frac{\Psi \vdash P_i :: (x_i : A^i_m)\ \text{for each } i \in I}{\Psi \vdash \mathsf{case}\ c\ \{i(x_i) \Rightarrow P_i\}_{i \in I} :: (c : \&_{i \in I} A^j_m)}\ \& R \qquad \frac{\ell \in I}{(a : \&_{i \in I} A^i_m) \vdash \mathsf{send}\ a\ \ell(c) :: (c : A^\ell_m)}\ \& L^0_\ell$$

$$\frac{}{(a : A_m)\ (b : B_m) \vdash \mathsf{send}\ c\ \langle a, b\rangle :: (c : A_m \otimes B_m)}\ \otimes R^0 \qquad \frac{\Psi\ (x : A_m)\ (y : B_m) \vdash P :: (c : C_k)}{\Psi\ (a : A_m \otimes B_m) \vdash \langle x, y\rangle \leftarrow \mathsf{recv}\ a; P :: (c : C_k)}\ \otimes L$$

$$\frac{}{\cdot \vdash \mathsf{close}\ c :: (c : \mathbf{1}_m)}\ \mathbf{1}R \qquad \frac{\Psi \vdash P :: (c : C_k)}{\Psi\ (a : \mathbf{1}_m) \vdash \mathsf{wait}\ a; P :: (c : C_k)}\ \mathbf{1}L$$

$$\frac{(x : A_m)\ \Psi \vdash P :: (y : B_m)}{\Psi \vdash \langle x, y\rangle \leftarrow \mathsf{recv}\ c; P :: (c : A_m \multimap B_m)}\ \multimap R \qquad \frac{}{(a : A_m)\ (c : A_m \multimap B_m) \vdash \mathsf{send}\ c\ \langle a, b\rangle :: (b : B_m)}\ \multimap L^0$$

$$\frac{\Psi \vdash P :: (x : A_k)}{\Psi \vdash \mathsf{shift}(x) \leftarrow \mathsf{recv}\ c; P :: (c : \uparrow^m_k A_k)}\ \uparrow R \qquad \frac{}{(a : \uparrow^m_k A_k) \vdash \mathsf{send}\ a\ \mathsf{shift}(c) :: (c : A_k)}\ \uparrow L^0$$

$$\frac{}{(a : A_m) \vdash \mathsf{send}\ c\ \mathsf{shift}(a) :: (c : \downarrow^m_k A_m)}\ \downarrow R^0 \qquad \frac{\Psi\ (x : A_m) \vdash P :: (c : C_\ell)}{\Psi\ (a : \downarrow^m_k A_m) \vdash \mathsf{shift}(c) \leftarrow \mathsf{recv}\ a; P :: (c : C_\ell)}\ \downarrow L$$

Figure 2: Process Assignment for Asynchronous Adjoint Logic

affine, and linear) by Pfenning and Griffith [2015]. No other structural connectives were supported, and a consistent operational interpretation of those was left as an open question.

In this section, we propose an answer to this question, which required a reformulation of adjoint logic by making the structural rules explicit and using multicut instead of the ordinary cut. Remarkably, we obtain two new operational phenomena: (1) a form of *multicast communication* and (2) a logically justified form of explicit garbage collection.

4.1 Static Semantics

We begin by providing proof terms for the rules in our sequent calculus, as shown in Figure 2. We can then interpret the proof terms as process expressions, and these rules are used to give the typing judgment for such processes. Table 1 gives the informal meaning of each such process term. Note that the process terms for shifts are a special case of the process terms for $\oplus$ and $\&$, and so are combined in the figure. In general, the process syntax represents an intermediate point between a programmer-friendly syntax and a notation in which it is easy to describe the operational semantics and prove progress and preservation. When compared to, say, SILL [Toninho et al. 2013], the main revisions are that (1) we make channel continuations explicit in order to facilitate asynchronous communication while preserving message order [DeYoung et al. 2012], and (2) we distinguish between an *internal name* for the channel provided by a process and *external names* connecting it to multiple clients.

Process term	Meaning
$a \leftarrow c$	Identify channels a and c.
$S \leftarrow (x.P); Q$	Spawn a new process P providing channels S to be used by Q. Here, x is the *internal name* in P for the channel offered by P, and S is the set of *external names* of the same channel.
$\mathsf{send}\ c\ \ell(a)$	Send the label ℓ and the channel a along c.
$\mathsf{case}\ c\{i(x_i) \Rightarrow P_i\}_{i \in I}$	Receive a label i and a channel x_i from c, continue as P_i.
$\mathsf{send}\ c\ \langle a, b\rangle$	Send the channels a and b along c.
$\langle x, y\rangle \leftarrow \mathsf{recv}\ c; P$	Receive channels x and y from c to be used in P.
$\mathsf{close}\ c$	End communication over c.
$\mathsf{wait}\ c; P$	Wait for c to be closed, continue as P.

Table 1: Informal Meanings of Process Terms

4.2 Some Simple Examples

At this point, we can write down some actual processes. We provide here some small examples, along with their type information.

First, we have a process that can be written at any mode m, which witnesses that $\otimes_m$ is commutative.

$(x : A_m \otimes B_m) \vdash \langle y, x'\rangle \leftarrow \mathsf{recv}\ x; \mathsf{send}\ z\ \langle x', y\rangle :: (z : B_m \otimes A_m)$

If m is a mode that admits contraction, we can write the following process, which witnesses that $A_m\ \&\ B_m$ proves $A_m \otimes B_m$ in the

presence of contraction. '%' starts a comment, and $a.(P)$ binds an internal name a for the channel provided by P.

$$
\begin{array}{ll}
(p : A_m \mathbin{\&} B_m) \vdash \{p_1, p_2\} \leftarrow q.(q \leftarrow p); & \% \ \{p_1, p_2\} \leftarrow \mathsf{copy}\ p \\
\qquad x \leftarrow a.(\mathsf{send}\ p_1\ \pi_1(a)); & \\
\qquad y \leftarrow b.(\mathsf{send}\ p_2\ \pi_2(b)); & \\
\qquad \mathsf{send}\ z\ \langle x, y\rangle & \\
:: (z : A_m \otimes B_m) &
\end{array}
$$

If m is a mode that admits weakening, we can write the following process, which witnesses that $A_m \otimes B_m$ proves $A_m \mathbin{\&} B_m$ in the presence of weakening.

$$
\begin{array}{ll}
(x : A \otimes B) \vdash \mathsf{case}\ p\ \ \{\, \pi_1(p_1) \Rightarrow \langle y, z\rangle \leftarrow \mathsf{recv}\ x; & \\
\qquad\qquad \emptyset \leftarrow a.(a \leftarrow z); & \% \ \mathsf{drop}\ z \\
\qquad\qquad p_1 \leftarrow y & \\
\qquad \mid \pi_2(p_2) \Rightarrow \langle y, z\rangle \leftarrow \mathsf{recv}\ x; & \\
\qquad\qquad \emptyset \leftarrow a.(a \leftarrow y); & \% \ \mathsf{drop}\ y \\
\qquad\qquad p_2 \leftarrow z\,\} & \\
:: (p : A \mathbin{\&} B) &
\end{array}
$$

Three further examples involving a recursive type can be found in Section 5.3.

4.3 Dynamic Semantics

In order to describe the computational behavior of process expressions, we need to first give some syntax for the computational artifacts, which are running processes $\mathsf{proc}(\ldots)$.

In this notation, $\mathsf{proc}(S, \Delta, a.P)$ represents a process P which provides to clients S along a channel that it knows internally as a, using channels in Δ. That is, Δ consists of the free channels in P. We will write $\overline{\Psi}$ for the set of variables declared in Ψ.

A *process configuration* is a multiset of processes:

$$C \quad ::= \quad (\cdot) \mid C\ \mathsf{proc}(S, \Delta, a.P)$$

where we require that all the channels provided by the processes $\mathsf{proc}(S, \Delta, a.P)$ are distinct, i.e., given objects $\mathsf{proc}(S, \Delta_1, a.P)$ and $\mathsf{proc}(T, \Delta_2, b.Q)$ in the same process configuration, S and T are disjoint. We will specify the operational semantics in the form of *multiset rewriting rules* [Cervesato and Scedrov 2009]. That means we show how to rewrite some subset of the configuration while leaving the remainder untouched. This form provides some assurance of the locality of the rules.

It simplifies the description of the operational semantics, if for any process $\mathsf{proc}(S, \Delta, a.P)$, Δ consists of exactly the free channels in P. This requires that we restrict the labeled internal and external choices, $\bigoplus_{i \in I} A^i_m$ and $\mathop{\&}_{i \in I} A^i_m$ to the case where $I \neq \emptyset$. Since a channel of empty choice type can never carry any messages, this is not a significant restriction in practice.

4.3.1 Configuration typing. In order to understand the rules of the operational semantics, it will be helpful to understand the typing of configurations. The judgment has the form $\Psi \vDash C :: \Psi'$ which expresses that using the channels in Ψ, configuration C provides the channels in Ψ'. This allows a channel that is not mentioned at all in C to appear in both Ψ and Ψ'—we think of such a channel as being "passed through" the configuration.

Note that while the configuration typing rules induce an ordering on a configuration, the configuration itself is not inherently ordered. The key rule is the first: for any formula $\mathsf{proc}(S, \Delta, a.P)$, a is the *internal name* of the channel provided by P while S is the set of all clients. An important restriction is that the number of clients must be compatible with the mode m of the offered channel, which is exactly that $|S| \sim m$, as defined in Section 2.5.

$$\frac{|S| \sim m \quad \Psi' \vdash P :: (a : A_m)}{\Psi\ \Psi' \vDash \mathsf{proc}(S, \overline{\Psi'}, a.P) :: \Psi\ (S : A_m)}$$

$$\frac{}{\Psi \vDash (\cdot) :: \Psi} \qquad \frac{\Psi \vDash C :: \Psi' \quad \Psi' \vDash C' :: \Psi''}{\Psi \vDash C\ C' :: \Psi''}$$

The identity and composition rules are straightforward. The empty context $(\cdot)$ provides Ψ if given Ψ, since it does not use any channels in Ψ or provide any additional channels. Composition just connects configurations with compatible interfaces: what is provided by C is used by C'.

The computation rules we discuss below can be found in Figure 3.

4.3.2 Judgmental rules. The identity rule (written as $\overset{\mathsf{id}}{\Longrightarrow}$) describes how an identity process $\mathsf{proc}(S, \{c\}, a.(a \leftarrow c))$ can interact with other processes. We think of such a process as connecting the provider of c to clients in S, and therefore sometimes call it a *forwarding process*. A forwarding process interacts with the provider of c, telling it to replace c with S in its set of clients. In adding S to the set of clients, the forwarding process accomplishes its goal of connecting the provider of c to S, and so it can terminate.

The cut rule steps by spawning a new process which offers along a fresh set of channels S', all of which are used in Q, the continuation of the original process. Here we write Δ_P and Δ_Q for the set of free channels in P and Q, respectively.

4.3.3 Structural rules. As can be seen from the proof of cut elimination (Appendix A.2), a principal multicut reduction creates several new cuts: one with a smaller proposition (these are embodied in rules $(*)C$ for each connective $(*)$), possibly some with a smaller proof, and possibly followed by some contractions.

We refactor these into several smaller steps. First, if the multicut eliminates more than one copy of a proposition, we divide it into two multicuts followed by some contractions. This way we isolate the single proposition that is actually eliminated into a multicut on the same proposition and a smaller proof, followed by a singleton cut. The residual contractions are of course then again implemented by cuts with the identity. The critical step here is the following, assuming that $\mathsf{C} \in \sigma(m)$ and S and T nonempty:

$$\frac{\Psi \vdash A_m \quad (S \cup T : A_m)\ \Psi' \vdash C}{\Psi\ \Psi' \vdash C}\ \mathsf{cut}(S \cup T)$$

$$\Longrightarrow \quad \frac{\Psi \vdash A_m \quad \dfrac{\Psi \vdash A_m \quad (S \cup T : A_m)\ \Psi' \vdash C}{(T : A_m)\ \Psi\ \Psi' \vdash C}\ \mathsf{cut}(S)}{\dfrac{\Psi\ \Psi\ \Psi' \vdash C}{\Psi\ \Psi' \vdash C}\ \mathsf{contract}^*}\ \mathsf{cut}(T)$$

This is embodied in the rule $\overset{\mathsf{copy}}{\Longrightarrow}$ where $\Delta = \overline{\Psi}$. The identity processes here implement the residual contractions. We require P not to be an identity in order to prevent circular reductions. The transformation is entirely parametric in the client(s), which we therefore do not need to make explicit.

$$
\begin{array}{rcl}
\mathsf{proc}(T \cup \{c\}, \Delta, a.P) \\ \mathsf{proc}(S, \{c\}, a.(a \leftarrow c)) & \overset{\mathsf{id}}{\Longrightarrow} & \mathsf{proc}(T \cup S, \Delta, a.P) \\[1ex]
\mathsf{proc}(T, \Delta_P \cup \Delta_Q, a.(S \leftarrow (x.P); Q)) & \overset{\mathsf{cut}(S)}{\Longrightarrow} & \mathsf{proc}(S', \Delta_P, x.P) \qquad (S' \text{ a fresh set of channels matching } S) \\ & & \mathsf{proc}(T, \Delta_Q \cup \{S'\}, a.Q[S'/S]) \\[1ex]
\mathsf{proc}(\emptyset, \Delta, a.P) & \overset{\mathsf{drop}}{\Longrightarrow} & \mathsf{proc}(\emptyset, \{b\}, a.(a \leftarrow b))_{b \in \Delta} \qquad (P \text{ not an identity}) \\[1ex]
\mathsf{proc}(S \cup T, \Delta, a.P) & \overset{\mathsf{copy}}{\Longrightarrow} & \mathsf{proc}(\{b', b''\}, \{b\}, a.(a \leftarrow b))_{b \in \Delta} \\ & & \mathsf{proc}(S, \{b'\}_{b \in \Delta}, a.P) \qquad (P \text{ not an identity and } S, T \text{ non-empty}) \\ & & \mathsf{proc}(T, \{b''\}_{b \in \Delta}, a.P) \\[1ex]
\mathsf{proc}(\{b\}, \{c\}, a.(\mathsf{send}\ a\ \ell(c))) \\ \mathsf{proc}(S, \Delta \cup \{b\}, a.(\mathsf{case}\ b\ \{i(d_i) \Rightarrow P_i\}_{i \in I})) & \overset{\oplus C}{\Longrightarrow} & \mathsf{proc}(S, \Delta \cup \{c\}, a.P_\ell[c/d_\ell]) \\[1ex]
\mathsf{proc}(\{b\}, \Delta, a.(\mathsf{case}\ a\ \{i(d_i) \Rightarrow P_i\}_{i \in I})) \\ \mathsf{proc}(\{c\}, \{b\}, a.(\mathsf{send}\ b\ \ell(a))) & \overset{\& C}{\Longrightarrow} & \mathsf{proc}(\{c\}, \Delta, a.P_\ell[a/d_\ell]) \\[1ex]
\mathsf{proc}(\{b\}, \{c, d\}, a.(\mathsf{send}\ a\ \langle c, d\rangle)) \\ \mathsf{proc}(S, \Delta \cup \{b\}, a.(\langle x, y\rangle \leftarrow \mathsf{recv}\ b; P)) & \overset{\otimes C}{\Longrightarrow} & \mathsf{proc}(S, \Delta \cup \{c, d\}, a.P[c/x, d/y]) \\[1ex]
\mathsf{proc}(\{b\}, \Delta, a.(\langle x, y\rangle \leftarrow \mathsf{recv}\ a;\ P)) \\ \mathsf{proc}(\{c\}, \{b, d\}, a.(\mathsf{send}\ b\ \langle d, a\rangle)) & \overset{\multimap C}{\Longrightarrow} & \mathsf{proc}(\{c\}, \Delta \cup \{d\}, a.P[d/x, a/y]) \\[1ex]
\mathsf{proc}(\{b\}, \emptyset, a.(\mathsf{close}\ a)) \\ \mathsf{proc}(S, \Delta \cup \{b\}, a.(\mathsf{wait}\ b;\ P)) & \overset{\mathbf{1}\, C}{\Longrightarrow} & \mathsf{proc}(S, \Delta, a.P) \\[1ex]
\mathsf{proc}(\{b\}, \Delta, a.(\mathsf{shift}(x) \leftarrow \mathsf{recv}\ a; P)) \\ \mathsf{proc}(\{c\}, \{b\}, a.(\mathsf{send}\ b\ \mathsf{shift}(a))) & \overset{\uparrow C}{\Longrightarrow} & \mathsf{proc}(\{c\}, \Delta, a.(P[a/x])) \\[1ex]
\mathsf{proc}(\{b\}, \{c\}, a.(\mathsf{send}\ a\ \mathsf{shift}(c))) \\ \mathsf{proc}(S, \Delta \cup \{b\}, a.(\mathsf{shift}(x) \leftarrow \mathsf{recv}\ b;\ P)) & \overset{\downarrow C}{\Longrightarrow} & \mathsf{proc}(S, \Delta \cup \{c\}, a.P[c/x])
\end{array}
$$

Figure 3: Computation Rules for Asynchronous Adjoint Logic

A similar consideration in the case where a multicut eliminates zero copies of a proposition justifies the $\overset{\mathsf{drop}}{\Longrightarrow}$ rule of computation.

In the copy rule, we use a few conventions for simplicity. First, we implicitly assume that b' and b'' are fresh for each channel b. We also use the shorthand Δ' for $\{b' \mid b \in \Delta\}$, and similarly for Δ''. The substitution $P[\Delta'/\Delta]$ is the obvious pointwise substitution of each b' for the corresponding b.

4.3.4 Additive and Multiplicative connectives. In the computation rule for $\oplus$, the process $\mathsf{proc}(\{b\}, \{c\}, a.(\mathsf{send}\ a\ \ell(c)))$ represents the message 'label ℓ with continuation c'. After this message has been received, the process terminates since b was its only client. The recipient selects the appropriate branch of the case construct and also substitutes the continuation channel c for the continuation variable d_ℓ.

The $\&$ computation rule is largely similar to that for $\oplus$, except that communication proceeds in the opposite direction—messages are sent to providers from clients, rather than from providers to clients as in the case of $\oplus$.

The multiplicative connectives $\otimes$ and $\multimap$ behave similarly to their additive counterparts, except that rather than sending and receiving labels, they send and receive channels together with a continuation, and so an extra substitution is required when receiving messages.

The rule for $\mathbf{1}$ behaves as a nullary $\otimes$, allowing us to signal that no more communication is forthcoming along a channel, and to wait for such a signal before continuing to compute.

4.3.5 Shifts. Operationally, $\uparrow$ behaves essentially the same as unary $\&$, while $\downarrow$ behaves as unary $\oplus$, and so these rules are similar to the computation rules for those connectives. Their significance lies in the *mode shift* of the continuation channel that is transmitted, which is required for the configuration to remain well-typed.

5 PRESERVATION AND PROGRESS

While in the logic we can prove cut elimination, in a programming language we would support recursive types and recursive processes which may not terminate. Moreover, from a programmer's perspective we are not even interested in eliminating all cuts (which would correspond to reducing under λ-abstractions in a functional language) but we block when waiting to receive a message, analogous to a λ-abstraction waiting for input before it can reduce.

What we prove instead are the typical progress and preservation properties, adjusted to adjoint logic and to our specific operational semantics. Type preservation is usually referred to as session fidelity and progress as deadlock freedom.

5.1 Preservation (Session Fidelity)

Theorem 5.1 (Preservation). *If $\Psi \vDash C :: \Psi'$ and $C \Rightarrow C'$, then $\Psi \vDash C' :: \Psi'$.*

Proof. This proceeds by a case analysis on the computation rule used to get that $C \Rightarrow C'$.

The general structure of each case is to use Lemma A.18 to break C down into the processes on which the computation rule acts and some additional collections of processes which are irrelevant to the computation. Once we have done this, we build a proof that $\Psi \vDash C' :: \Psi'$ from these pieces. □

5.2 Global Progress (Freedom from Deadlocks)

The progress theorem for a functional language states that an expression is either a value or it can make a step. Here we do not have values, but there is nevertheless a clear analogue between, say, a value $\lambda x.e$ that waits for an argument, and a process $y \leftarrow \text{recv } x\ ;\ P$ that waits for an input. We formalize this in the definition below.

Definition 5.2. We say that a process $\text{proc}(S, \Delta, a.P)$ is *poised* on a if:

(1) it is a process $\text{proc}(S, \Delta, a.P)$ that sends on a — that is, P is of the form (send a _) or (close a), or
(2) it is a process $\text{proc}(S, \Delta, a.P)$ that receives on a — that is, P is of the form (case a _), (wait a ; _), or (_ ← recv a ; _).

Intuitively, $\text{proc}(S, \Delta, a.P)$ is poised on a if it is blocked trying to communicate along a. This definition allows us to state the following progress theorem:

Theorem 5.3 (Progress). *If $(\cdot) \vDash C :: \Psi$, then exactly one of the following holds:*

(1) There is a C' such that $C \Rightarrow C'$.
(2) Every $\text{proc}(S, \Delta, a.P)$ in C is poised on a.

Proof. This follows from an induction on the derivation of $(\cdot) \vDash C :: \Psi$, using the $\vDash'$ rules defined in Appendix A.4. Writing $C = C'\ \text{proc}(S, \overline{\Psi'}, a.P)$, we see that either C' can step, in which case so can C, or every process in C' is poised.

Now we carefully distinguish cases on S (empty, singleton, or greater) and apply inversion to the typing of P to see that in each case the process either is poised, can take a step independently, or can interact with provider of a channel in $\overline{\Psi'}$. □

5.3 Garbage Collection

As we can see from the preservation theorem, the interface to a configuration never changes. While new processes may be spawned, they will have clients and are therefore not visible at the interface. That is in contrast to the semantics of shared channels in prior work (for example, in Caires and Pfenning [2010]) where shared channels may show up as newly provided channels. Therefore they may be left over at the end of a computation without any clients.

This cannot happen here. Initially, at the top level, we envision starting with

$$\cdot \vDash \text{proc}(\{c_0\}, \cdot, c.P_0) :: (c_0 : \mathbf{1})$$

Assuming this computation completes, by the progress property and the definition of *poised*, computation could only halt with

$$\cdot \vDash \text{proc}(\{c_0\}, \cdot, c.(\text{close } c)) :: (c_0 : \mathbf{1})$$

In other words: no garbage!

One can generalize this theorem to allow some nontrivial output by allowing any purely positive type (that is, one which only uses the fragment of the logic with connectives $\oplus$, $\otimes$, $\mathbf{1}$, and $\downarrow$), such as $\oplus\{\text{false} : \mathbf{1}, \text{true} : \mathbf{1}\}$.

Perhaps most interesting here is an extension, following SILL [Pfenning and Griffith 2015; Toninho et al. 2013] in a straightforward way, where we allow recursive types and recursive definition beyond the pure logic. For example, we can define binary numbers in "little endian" representation (least significant bit first) as

$$\text{bits}_m = \oplus\{\text{b0} : \text{bits}_m, \text{b1} : \text{bits}_m, \text{e} : \mathbf{1}_m\}$$

We do not specify here the mode m because the examples below will work for any mode, regardless of its structural properties! Then the number $5 = (101)_2$ is represented by the following process:

$$\begin{array}{l}\cdot \vdash \text{five}(x_0) :: (x_0 : \text{bits}) \\ \text{five}(x_0) = x_4 \leftarrow a.(\text{close } a); \\ \qquad\qquad x_3 \leftarrow a.(\text{send } a\ \text{e}(x_4)); \\ \qquad\qquad x_2 \leftarrow a.(\text{send } a\ \text{b1}(x_3)); \\ \qquad\qquad x_1 \leftarrow a.(\text{send } a\ \text{b0}(x_2)); \\ \qquad\qquad \text{send } x_0\ \text{b1}(x_1)\end{array}$$

which evolves into five processes: three representing the bits 1, 0, 1; one (e) for the end of the number; and one to close the channel. In fact, by the progress and preservation theorems and inversion on typing, we know that if any process $\cdot \vdash P :: (x_0 : \text{bits}_m)$ terminates, then P will represent a binary number with one process for each of its bits and two more to mark the end of the number and close the channel.

As a last example, the recursive process definition $\text{inc}(x, y)$ reads a stream of bits along channel x and sends an incremented stream along y.

$$\begin{array}{l}(x : \text{bits}) \vdash \text{inc}(x, y) :: (y : \text{bits}) \\ \text{inc}(x, y) = \text{case } x\ \{\ \text{b0}(x') \Rightarrow \text{send } y\ \text{b1}(x') \\ \qquad\qquad\qquad\qquad \mid \text{b1}(x') \Rightarrow y' \leftarrow a.(\text{inc}(x', a)); \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{send } y\ \text{b0}(y') \\ \qquad\qquad\qquad\qquad \mid \text{e}(x') \Rightarrow y' \leftarrow a.(\text{send } a\ \text{e}(x')) \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{send } y\ \text{b1}(y')\ \}\end{array}$$

We can obtain the representation of 6 by incrementing 5.

$$\begin{array}{l}\cdot \vdash \text{six}(x_0) :: (x_0 : \text{bits}) \\ \text{six}(x_0) = x_1 \leftarrow a.(\text{five}(a)); \\ \qquad\qquad \text{inc}(x_1, x_0)\end{array}$$

6 RELATED WORK

Various items of related work have already been mentioned in the preceding sections either in examples or technical cross-references.

The most closely related work and immediate inspiration comes from the unpublished Reed [2009] which introduces an arbitrary preorder on modes with a uniform logical language and sequent calculus rules. It uses the notation $F_{q \geq p} A_q = \downarrow_p^q A_q$ and $U_{q \geq p} A_q = \uparrow_q^p A_q$. It mostly stays in the realm of structural logics, but Section 4.4 sketches intuitionistic linear logic and LNL as examples. It does not use explicit weakening and contraction rules (which are incorporated into the other rules as is frequently done for sequent calculi), and, while it uniformly proves cut elimination in the case of structural modes, it does not provide an operational semantics.

Recently, Licata et al. [2017] have further generalized Reed's adjoint logic by uniquely labeling antecedents and then controlling their use through a resource annotation of a sequent. Their resource annotations are made in an expressive *mode theory* which allows a richer set of logics to be represented than in our system here, including non-associative, ordered, and bunched logics. In addition, Licata et al. view multiplicative connectives such as tensor ($\otimes$) or linear implication ($\multimap$) as instances of a new generalized form of the adjoint modal operators, which paves the way for yet additional operators to be represented. In particular, their system allows $?_a A$ of intuitionistic subexponential logic (see Section A.1.4) and $\diamond A$ of judgmental modal logic (see Example 2.5) to be encoded directly, which, as far as we can tell, requires at least a 4-point lattice and an additional distinguished atom a_r [Reed 2009, Section 4.5] in our setting.

This generality also comes at a price. The declaration of independence is no longer a fundamental notion, but a roughly corresponding *strengthening lemma* has to be proved and requires some complex conditions on the mode theory [Licata et al. 2017, Lemma 7.1]. Adequacy of encodings also becomes more complicated. Furthermore, Licata et al. do not provide an operational semantics; their interest (like Reed's) lies on the logical and categorical side.

In a different direction, there is some related work on the use of linearity for garbage collection. The work of Wadler [1990] presents a simple language which uses linearity for state change without a need for garbage collection. Roughly contemporaneous with this is work by Chirimar et al. [1992], which focuses more on the idea of implementing garbage collection with linear logic, allowing the programmer to "dispose" of variables which are not needed in a type-safe fashion. Again, however, this focuses on a functional language, with the intent to allow for easier reasoning about memory optimizations in functional programming. More recently, [Igarashi and Kobayashi 2000, 2002] deal with garbage collection (and the more general problem of resource management) using linearity. Yet again, this deals only with the case of sequential functional programs. A major difference between the prior work and our work, then, is that we work in a concurrent setting, and indeed our garbage collection is concurrent as well.

7 CONCLUSION

At this point, our formulation of adjoint logic and its operational semantics seem to fit well and provide a good explanation for multicast communication and distributed garbage collection. Moreover, if used linearly, the semantics coincides with the purely linear semantics developed in prior work. In future work, we plan to investigate if the logic also lends itelf to a shared memory implementation (as the example in Section 5.3 suggests), and if the declaration of independence is sufficient to allow a *modular* combination of different operational interpretations for different modes. Of particular interest here would be the sharing semantics [Balzer and Pfenning 2017]. We also have not yet explored the full range of examples suggested by instances of the adjoint logic framework, such as potential concurrent programming application of judgmental S4 (comonads) or lax logic (strong monads).

REFERENCES

Stephanie Balzer and Frank Pfenning. 2017. Manifest Sharing with Session Types. In *International Conference on Functional Programming (ICFP)*. ACM, 37:1–37:29.

Andrew Barber. 1996. *Dual Intuitionistic Linear Logic.* Technical Report ECS-LFCS-96-347. Department of Computer Science, University of Edinburgh.

Nick Benton. 1994. A Mixed Linear and Non-Linear Logic: Proofs, Terms and Models. In *Selected Papers from the 8th International Workshop on Computer Science Logic (CLS'94)*, Leszek Pacholski and Jerzy Tiuryn (Eds.). Springer LNCS 933, Kazimierz, Poland, 121–135. An extended version appears as Technical Report UCAM-CL-TR-352, University of Cambridge.

Luís Caires and Frank Pfenning. 2010. Session Types as Intuitionistic Linear Propositions. In *Proceedings of the 21st International Conference on Concurrency Theory (CONCUR 2010)*. Springer LNCS 6269, Paris, France, 222–236.

Luís Caires, Frank Pfenning, and Bernardo Toninho. 2016. Linear Logic Propositions as Session Types. *Mathematical Structures in Computer Science* 26, 3 (2016), 367–423.

Iliano Cervesato and Andre Scedrov. 2009. Relating State-Based and Process-Based Concurrency through Linear Logic. *Information and Computation* 207, 10 (Oct. 2009), 1044–1077.

Bor-Yuh Evan Chang, Kaustuv Chaudhuri, and Frank Pfenning. 2003. *A Judgmental Analysis of Linear Logic.* Technical Report CMU-CS-03-131R. Carnegie Mellon University, Department of Computer Science.

Kaustuv Chaudhuri. 2010. Classical and Intuitionistic Subexponential Logics are Equally Expressive. In *Computer Science Logic*. Springer LNCS 6247, 185–199.

Jawahar Chirimar, Carl A Gunter, and Jon G Riecke. 1992. Proving memory management invariants for a language based on linear logic. *ACM SIGPLAN Lisp Pointers* 1 (1992), 139–150.

Henry DeYoung, Luís Caires, Frank Pfenning, and Bernardo Toninho. 2012. Cut Reduction in Linear Logic as Asynchronous Session-Typed Communication. In *Proceedings of the 21st Conference on Computer Science Logic (CSL 2012)*, P. Cégielski and A. Durand (Eds.). 228–242.

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

M. Fairtlough and M.V. Mendler. 1997. Propositional Lax Logic. *Information and Computation* 137, 1 (Aug. 1997), 1–33.

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

Jean-Yves Girard. 1987. Linear Logic. *Theoretical Computer Science* 50 (1987), 1–102.

Atsushi Igarashi and Naoki Kobayashi. 2000. Garbage collection based on a linear type system. In *Preliminary Proceedings of the 3rd ACM SIGPLAN Workshop on Types in Compilation (TIC'00)*, Vol. 152.

Atsushi Igarashi and Naoki Kobayashi. 2002. Resource usage analysis. In *ACM SIGPLAN Notices*, Vol. 37. ACM, 331–342.

Daniel R. Licata, Michael Shulman, and Mitchell Riley. 2017. A Fibrational Framework for Substructural and Modal Logics. In *International Conference on Formal Structures for Computation and Deduction*. LIPIcs, Oxford.

Sara Negri and Jan von Plato. 2001. *Structural Proof Theory.* Cambridge University Press.

Vivek Nigam and Dale Miller. 2009. Algorithmic Specifications in Linear Logic with Subexponentials. In *Proceedings of the 11th International Conference on Principles and Practice of Declarative Programming (PPDP)*. ACM, Coimbra, Portugal, 129–140.

Frank Pfenning. 2016. Law and Order. (Sept. 2016). http://www.cs.cmu.edu/~fp/courses/15816-f16/lectures/08-lawandorder.pdf Lecture notes on *Substructural Logics*.

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

Frank Pfenning and Dennis Griffith. 2015. Polarized Substructural Session Types. In *Proceedings of the 18th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2015)*, A. Pitts (Ed.). Springer LNCS 9034, London, England, 3–22. Invited talk.

Jason Reed. 2009. A Judgmental Deconstruction of Modal Logic. (May 2009). http://www.cs.cmu.edu/~jcreed/papers/jdml2.pdf Unpublished manuscript.

Bernardo Toninho, Luís Caires, and Frank Pfenning. 2013. Higher-Order Processes, Functions, and Sessions: A Monadic Integration. In *Proceedings of the European Symposium on Programming (ESOP'13)*, M.Felleisen and P.Gardner (Eds.). Springer LNCS 7792, Rome, Italy, 350–369.

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

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

A APPENDIX

A.1 Logic Examples, Extended

Here, we provide more formal definitions of the embeddings τ from various logics into instances of adjoint logic, as well as proofs of their adequacy.

A.1.1 LNL. As described in Example 2.2, we will work with an adjoint logic with two modes $\mathsf{L} < \mathsf{U}$, where $\sigma(\mathsf{L}) = \emptyset$ and $\sigma(\mathsf{U}) = \{\mathsf{W}, \mathsf{C}\}$.

For this section, we will use the following subset of LNL, where X^0, A^0 are base propositions. Connectives other than $\multimap$ and $\to$ can be translated similarly, and so in the interest of brevity, we omit them.

$$\begin{array}{lrcl} \text{Persistent Propositions} & X & ::= & X^0 \mid X_1 \to X_2 \mid GA \\ \text{Linear Propositions} & A & ::= & A^0 \mid A_1 \multimap A_2 \mid FX \\ \text{Persistent Hypotheses} & \Theta & ::= & \cdot \mid \Theta, X \\ \text{Linear Hypotheses} & \Gamma & ::= & \cdot \mid \Gamma, A \end{array}$$

Definition A.1. We define an embedding τ of LNL into adjoint logic as follows:

$$\begin{array}{rcl} \tau(X^0) & = & X_{\mathsf{U}} \\ \tau(X_1 \to X_2) & = & \tau(X_1) \multimap_{\mathsf{U}} \tau(X_2) \\ \tau(GA) & = & \uparrow^{\mathsf{U}}_{\mathsf{L}} \tau(A) \\ \\ \tau(A^0) & = & A_{\mathsf{L}} \\ \tau(A_1 \to A_2) & = & \tau(A_1) \multimap_{\mathsf{L}} \tau(A_2) \\ \tau(FX) & = & \downarrow^{\mathsf{U}}_{\mathsf{L}} \tau(X) \\ \\ \tau(\cdot) & = & \cdot \\ \tau(\Theta, X) & = & \tau(\Theta), (x : \tau(X)) \\ \\ \tau(\cdot) & = & \cdot \\ \tau(\Gamma, A) & = & \tau(\Gamma), (y : \tau(A)) \end{array}$$

Theorem A.2.

(a) $\Theta \vdash_{\mathcal{C}} X$ *in LNL iff* $\tau(\Theta) \vdash \tau(X)$ *in adjoint logic.*
(b) $\Theta; \Gamma \vdash_{\mathcal{L}} A$ *in LNL iff* $\tau(\Theta), \tau(\Gamma) \vdash \tau(A)$ *in adjoint logic.*

Proof. First, we take advantage of cut-elimination for adjoint logic to allow us to prove weaker implications, needing only to show that the existence of a cut-free proof in adjoint logic implies the existence of an LNL proof. Our proof is then reduced to proving the following:

(a.1) If $\tau(\Theta) \Vdash \tau(X)$ in adjoint logic, then $\Theta \vdash_{\mathcal{C}} X$ in LNL.
(a.2) If $\Theta \vdash_{\mathcal{C}} X$ in LNL, then $\tau(\Theta) \vdash \tau(X)$ in adjoint logic.
(b.1) If $\tau(\Theta), \tau(\Gamma) \Vdash \tau(A)$ in adjoint logic, then $\Theta; \Gamma \vdash_{\mathcal{L}} A$ in LNL.
(b.2) If $\Theta; \Gamma \vdash_{\mathcal{L}} A$ in LNL, then $\tau(\Theta), \tau(\Gamma) \vdash \tau(A)$ in adjoint logic.

We prove (a.1) and (b.1) by simultaneous induction on the derivation of $\Psi \Vdash C_m$.

We prove (a.2) and (b.2) by showing that (the translations of) the rules of LNL are derivable from the rules of adjoint logic. This is straightforward, and consists primarily of inserting the correct modes into rules of adjoint logic. □

A.1.2 Judgmental S4. As described in Example 2.5, we will work with an adjoint logic with two modes $\mathsf{U} < \mathsf{V}$, where $\sigma(\mathsf{U}) = \sigma(\mathsf{V}) = \{\mathsf{W}, \mathsf{C}\}$.

For this section, we will use the following subset of judgmental S4, where P are base propositions. Connectives other than $\supset$ can be translated similarly, and so are omitted.

$$\begin{array}{lrcl} \text{Propositions} & A & ::= & P \mid A_1 \supset A_2 \mid \Box A \\ \text{True Hypotheses} & \Gamma & ::= & \cdot \mid \Gamma, A\ \mathit{true} \\ \text{Valid Hypotheses} & \Delta & ::= & \cdot \mid \Delta, A\ \mathit{valid} \end{array}$$

Definition A.3. We define an embedding τ of judgmental S4 into adjoint logic as follows:

$$\begin{array}{rcl} \tau(P) & = & P_{\mathsf{U}} \\ \tau(A_1 \supset A_2) & = & \tau(A_1) \multimap_{\mathsf{U}} \tau(A_2) \\ \tau(\Box A) & = & \downarrow^{\mathsf{V}}_{\mathsf{U}} \uparrow^{\mathsf{V}}_{\mathsf{U}} \tau(A) \\ \\ \tau(\cdot) & = & \cdot \\ \tau(\Gamma, A\ \mathit{true}) & = & \tau(\Gamma), (x : \tau(A)) \\ \\ \tau(\cdot) & = & \cdot \\ \tau(\Delta, A\ \mathit{valid}) & = & \tau(\Delta), (y : \tau(A)) \end{array}$$

Theorem A.4.

(a) $\Delta; \Gamma \vdash A$ *in judgmental S4 iff* $\uparrow^{\mathsf{V}}_{\mathsf{U}} \tau(\Delta), \tau(\Gamma) \vdash \tau(A)$ *in adjoint logic.*
(b) $\Delta; \cdot \vdash A$ *in judgmental S4 iff* $\uparrow^{\mathsf{V}}_{\mathsf{U}} \tau(\Delta) \vdash \uparrow^{\mathsf{V}}_{\mathsf{U}} \tau(A)$ *in adjoint logic.*

Proof. We first note that in [Pfenning and Davies 2001], the inference system given by the rules above is stated to satisfy weakening and contraction for both true and valid hypotheses. As such, we may add weakening and contraction rules for judgmental S4 without changing the provability of judgments. We will write $\vdash^+$ for proofs in judgmental S4 using these structural rules in order to make them more obviously distinguished.

We will use this along with cut elimination for adjoint logic (in the same manner as in Section A.1.1) in order to reduce the result to proving the following implications:

(a.1) If $\uparrow^{\mathsf{V}}_{\mathsf{U}} \tau(\Delta), \tau(\Gamma) \Vdash \tau(A)$ in adjoint logic, then $\Delta; \Gamma \vdash^+ A$ in judgmental S4 augmented with structural rules.
(a.2) If $\Delta; \Gamma \vdash A$ in judgmental S4, then $\uparrow^{\mathsf{V}}_{\mathsf{U}} \tau(\Delta), \tau(\Gamma) \vdash \tau(A)$ in adjoint logic.
(b.1) If $\uparrow^{\mathsf{V}}_{\mathsf{U}} \tau(\Delta) \Vdash \uparrow^{\mathsf{V}}_{\mathsf{U}} \tau(A)$ in adjoint logic, then $\Delta; \cdot \vdash^+ A$ in judgmental S4 augmented with structural rules.
(b.2) If $\Delta; \cdot \vdash A$ in judgmental S4, then $\uparrow^{\mathsf{V}}_{\mathsf{U}} \tau(\Delta) \vdash \uparrow^{\mathsf{V}}_{\mathsf{U}} \tau(A)$ in linear logic

From here, the proof proceeds much as the proof for LNL.

We prove (a.1) and (b.1) by simultaneous induction on the derivation of $\Psi \Vdash C_m$.

We prove (a.2) and (b.2) by showing that (the translations of) the rules of judgmental S4 are derivable from the rules of adjoint logic. This is slightly less straightforward than the case of LNL, since the usual presentation of judgmental S4 is in terms of introduction and elimination rather than left and right rules, and so deriving the elimination rules involves using a cut. □

A.1.3 Lax Logic. As described in Example 2.7, we work here with two modes, $\mathsf{X} < \mathsf{U}$, where $\sigma(\mathsf{X}) = \sigma(\mathsf{U}) = \{\mathsf{W}, \mathsf{C}\}$. Interestingly, this is the same preorder (up to renaming) as that for judgmental S4.

We take the following as the syntax for lax logic, omitting connectives other than $\supset$ and $\bigcirc$ for brevity, as the proof extends naturally

to those cases.

$$\begin{array}{lrcl} \text{Propositions} & A & ::= & P \mid A_1 \supset A_2 \mid \bigcirc A \\ \text{Hypotheses} & \Gamma & ::= & \cdot \mid \Gamma, A \end{array}$$

Definition A.5. We define an embedding τ of lax logic into adjoint logic as follows:

$$\begin{array}{rcl} \tau(P) & = & P_{\mathsf{U}} \\ \tau(A_1 \supset A_2) & = & \tau(A_1) \multimap \tau(A_2) \\ \tau(\bigcirc A) & = & \uparrow^{\mathsf{U}}_{\mathsf{X}} \downarrow^{\mathsf{U}}_{\mathsf{X}} \tau(A) \\ \\ \tau(\cdot) & = & \cdot \\ \tau(\Gamma, A) & = & \tau(\Gamma), (x : \tau(A)) \end{array}$$

Theorem A.6.

(a) $\Gamma \vdash A$ *true in lax logic iff* $\tau(\Gamma) \vdash \tau(A)$ *in adjoint logic.*
(b) $\Gamma, \Gamma' \vdash A$ *lax in lax logic iff* $\tau(\Gamma), \downarrow^{\mathsf{U}}_{\mathsf{X}} \tau(\Gamma') \vdash \downarrow^{\mathsf{U}}_{\mathsf{X}} \tau(A)$ *in adjoint logic.*

Proof. Much like the proof for judgmental S4, we use cut elimination for adjoint logic along with the admissibility of weakening and contraction for lax logic to reduce our claims to the following:

(a.1) If $\tau(\Gamma) \Vdash \tau(A)$ in adjoint logic, then $\Gamma \vdash^{+} A$ *true* in lax logic augmented with structural rules.
(a.2) If $\Gamma \vdash A$ *true* in lax logic, then $\tau(\Gamma) \vdash \tau(A)$ in adjoint logic.
(b.1) If $\tau(\Gamma), \downarrow^{\mathsf{U}}_{\mathsf{X}} \tau(\Delta) \Vdash \downarrow^{\mathsf{U}}_{\mathsf{X}} \tau(A)$ in adjoint logic, then $\Gamma, \Delta \vdash^{+} A$ *lax* in lax logic augmented with structural rules.
(b.2) If $\Gamma, \Delta \vdash A$ *lax* in lax logic, then $\tau(\Gamma), \downarrow^{\mathsf{U}}_{\mathsf{X}} \tau(\Delta) \vdash \downarrow^{\mathsf{U}}_{\mathsf{X}} \tau(A)$ in adjoint logic.

As before, we prove (a.1) and (b.1) by simultaneous induction on the derivation of $\Psi \Vdash C_m$.

We prove (a.2) and (b.2) by showing that (the translations of) the rules of lax logic are derivable from the rules of adjoint logic. This proceeds in essentially the same way as for judgmental S4. □

A.1.4 Subexponential Linear Logic. We can represent a somewhat restricted form of intuitionistic subexponential linear logic (ISELL) [Chaudhuri 2010] as a fragment of adjoint logic. Subexponential labels of *zones* correspond to modes, and we preserve the preorder between labels as the preorder between modes. There is a *working zone* which corresponds to a distinguished mode L.

We require $z \geq \mathsf{L}$ for all modes $z \neq \mathsf{L}$ and define $!_z A = \downarrow^z_{\mathsf{L}} \uparrow^z_{\mathsf{L}} A$ for $z > \mathsf{L}$. We also work on the ?-free fragment, making this slightly less general than ISELL, which also includes $?_z A$ and allows labels $z < \mathsf{L}$. Indeed, the rules for the shifts under the obvious candidate representation $?_z A = \uparrow^{\mathsf{L}}_z \downarrow^{\mathsf{L}}_z A$ do not match the rules for $?_z A$ in ISELL. Fortunately, the modality $?_z$ is not in the image of the translation [Chaudhuri 2010, Section 4.1] from classical subexponential logic [Nigam and Miller 2009] into ISELL, so it does not appear essential to gauge its expressive power.

An instance of ISELL satisfying these requirements can then be seen as an instance of adjoint logic where all modes a other than L contain only propositions of the form $\uparrow^a_{\mathsf{L}} A_{\mathsf{L}}$.

Because subexponential logic is designed as a logical framework based on proof construction and focusing instead of proof reduction, the structural rules are integrated into the other rules rather than separated out. All other differences are cosmetic.

We also have a new opportunity, namely adding connectives that directly combine propositions of mode $z \neq \mathsf{L}$. The consequences warrant further proof-theoretic investigation, because the additional connectives may reduce the number of subexponential modalities in a logic representation. This in turn may streamline the focusing behavior of encodings since subexponential modalities interrupt focusing phases.

Formally, we fix a particular instance of ISELL (and a corresponding instance of adjoint logic), and take the following syntax for the fragment of ISELL which we consider, omitting connectives other than $\multimap$ and $!^a$ for brevity:

$$\begin{array}{lrcl} \text{Propositions} & A & ::= & P \mid A_1 \multimap A_2 \mid !^a A \\ \text{Hypotheses} & \Delta & ::= & \cdot \mid \Delta, A \end{array}$$

Definition A.7. We define an embedding τ from ISELL into adjoint logic as follows:

$$\begin{array}{rcl} \tau(P) & = & P_{\mathsf{L}} \\ \tau(A_1 \multimap A_2) & = & \tau(A_1) \multimap \tau(A_2) \\ \tau(!^a A) & = & \downarrow^a_{\mathsf{L}} \uparrow^a_{\mathsf{L}} \tau(A) \\ \\ \tau(\cdot) & = & \cdot \\ \tau(\Delta, A) & = & \tau(\Delta), (y : \tau(A)) \end{array}$$

Theorem A.8.

(a) $\Delta, !^{a_1} A_1, \ldots, !^{a_n} A_n \vdash B$ *in ISELL iff* $\tau(\Delta), \uparrow^{a_1}_{\mathsf{L}} \tau(A_1), \ldots, \uparrow^{a_n}_{\mathsf{L}} \tau(A_n) \vdash \tau(B)$ *in adjoint logic.*
(b) $!^{a_1} A_1, \ldots, !^{a_n} A_n \vdash !^b B$ *in ISELL iff* $\uparrow^{a_1}_{\mathsf{L}} \tau(A_1), \ldots, \uparrow^{a_n}_{\mathsf{L}} \tau(A_n) \vdash \uparrow^b_{\mathsf{L}} \tau(B)$ *in adjoint logic.*

Proof. As in the previous proofs, we use cut elimination for adjoint logic to reduce our claim to the weaker claims that:

(a.1) If $\tau(\Delta), \uparrow^{a_1}_{\mathsf{L}} \tau(A_1), \ldots, \uparrow^{a_n}_{\mathsf{L}} \tau(A_n) \Vdash \tau(B)$ in adjoint logic, then $\Delta, !^{a_1} A_1, \ldots, !^{a_n} A_n \vdash B$ in ISELL.
(a.2) If $\Delta, !^{a_1} A_1, \ldots, !^{a_n} A_n \vdash B$ in ISELL, then $\tau(\Delta), \uparrow^{a_1}_{\mathsf{L}} \tau(A_1), \ldots, \uparrow^{a_n}_{\mathsf{L}} \tau(A_n) \vdash \tau(B)$ in adjoint logic.
(b.1) If $\uparrow^{a_1}_{\mathsf{L}} \tau(A_1), \ldots, \uparrow^{a_n}_{\mathsf{L}} \tau(A_n) \Vdash \uparrow^b_{\mathsf{L}} \tau(B)$ in adjoint logic, then $!^{a_1} A_1, \ldots, !^{a_n} A_n \vdash !^b B$ in ISELL.
(b.2) If $!^{a_1} A_1, \ldots, !^{a_n} A_n \vdash !^b B$ in ISELL, then $\uparrow^{a_1}_{\mathsf{L}} \tau(A_1), \ldots, \uparrow^{a_n}_{\mathsf{L}} \tau(A_n) \vdash \uparrow^b_{\mathsf{L}} \tau(B)$ in adjoint logic.

Also as in the previous proofs, (a.1) and (b.1) may be proved by induction over the judgment $\Psi \Vdash C$, and (a.2) and (b.2) may be proved by showing that each of (the translations of) the rules of ISELL can be derived from the rules of adjoint logic. □

A.2 Cut Elimination

Theorem A.9. *The rule of multicut is admissible in the cut-free system.*

Proof. This proof follows the structure of many cut elimination results. First we prove admissibility of cut in the cut-free system: if $\mathcal{D}$ is a proof of $\Psi \Vdash A_m$ and $\mathcal{E}$ is a proof of $(S : A_m)\ \Psi' \Vdash C_k$, then we can construct a cut-free proof of $\Psi\ \Psi' \Vdash C_k$. This is proven by a straightforward induction on the triple $(A_m, \mathcal{D}, \mathcal{E})$, ordered lexicographically. This is followed by a simple structural induction to prove cut elimination, using the admissibility of cut when it is encountered. If we ignore the modes, this proof is very similar to the original proof of Gentzen [1935].

The particular cases of this are given by local cut reductions in a standard way. Because of their importance both for the operational semantics (Section 4) and equivalence proofs (Section 2.7), however, we show a representative sample of the reductions. Note that we will always identify proofs that are α-equivalent.

Example A.10 (An identity case). We begin by showing an identity case—specifically, the case where $\mathcal{D}$ is an identity, $\mathcal{E}$ is arbitrary, and $S = \emptyset$. This reduction has the form shown in the first line of Figure A.2. Note that the premise $\mathsf{W} \in \sigma(m)$ of the reduced proof follows from the premise $|\emptyset| \sim m$ of the original proof.

Example A.11 (A structural case). We now show a case where we cut with a structural rule—in particular, where $\mathcal{D}$ is arbitrary, $\mathcal{E}$ ends in a contraction rule, and $S = \{y\}$. This reduction is shown in the second line of Figure A.2. Note that the premise $|\{z, w\}| \sim m$ of the reduced proof follows from the premise $\mathsf{C} \in \sigma(m)$ of the original proof.

Example A.12 (A principal case). We now consider one of the principal cases for $\oplus$, as shown in the third line of Figure A.2. In particular, we consider the case where $S = T \cup \{x, y\}$. Here, the induction hypothesis is used at $(\bigoplus_{i \in I} A^i_m, \mathcal{D}, \mathcal{E}_\ell)$ in order to remove extra copies of $\bigoplus_{i \in I} A^i_m$, similarly to the standard cross-cut in cut elimination proofs for structural logic. In order to clean up the two copies of Ψ produced by the cross-cut and the second, more standard cut, we apply the contraction rule repeatedly, replacing $\Psi\ \Psi$ with Ψ. Note also that we omit the side condition $|\{y\}| \sim m$ of the second use of the induction hypothesis, as $1 \sim m$ always.

Example A.13 (A commutative case). Finally, we consider an example of a commutative case, where the reduction proceeds by pushing the cut further up in the proof. In particular, we consider the case where $\mathcal{D}$ is arbitrary, and $\mathcal{E}$ ends in the $\multimap R$ rule. This reduction proceeds as shown in the fourth line of Figure A.2.

While the details of the remaining cases are not identical, each remaining case is similar to one of the above. □

A.3 The Adjunction Property

In order to define functors, we first need to define the categories that will serve as their domain and codomain. For each mode m, there is a category (which we will also denote by m) defined as follows:

- The objects of m are the types A_m.
- A morphism $\mathcal{D} : A_m \to B_m$ is an equivalence class of proofs of $(x : A_m) \vdash (y : B_m)$, where two proofs are equivalent if they are related by cut reduction or identity expansion or if they are α-equivalent.
- The identity morphism at A_m is the equivalence class of

$$\frac{}{A_m \vdash A_m}\ \mathsf{id}_{A_m}$$

- Composition is given by cut—given $\mathcal{D} : A_m \to B_m$ and $\mathcal{E} : B_m \to C_m$, we define $\mathcal{E} \circ \mathcal{D}$ to be the equivalence class of

$$\frac{(x : A_m) \geq m \geq m \quad \overset{\mathcal{D}}{(x : A_m) \vdash B_m} \quad \overset{\mathcal{E}}{(y : B_m) \vdash C_m}}{(x : A_m) \vdash C_m}\ \mathsf{cut}(\{y\})$$

With this definition, the main theorem that we seek to prove is the following:

Theorem A.14. *$\downarrow^m_k$ and $\uparrow^m_k$ can be extended to functors $\downarrow^m_k : m \to k$ and $\uparrow^m_k : k \to m$ on the categories k and m as defined above. Moreover, these functors form an adjoint pair with $\downarrow^m_k \dashv \uparrow^m_k$.*

Proof. Throughout this proof, as m and k are fixed, we will omit them as subscripts and superscripts on shifts—we write $\downarrow$ for $\downarrow^m_k$ and $\uparrow$ for $\uparrow^m_k$.

Given $\mathcal{D} : A_m \vdash B_m$, we can define $\downarrow\mathcal{D} : \downarrow A_m \vdash \downarrow B_m$ as follows:

$$\frac{\dfrac{A_m \geq m \quad \overset{\mathcal{D}}{A_m \vdash B_m}}{A_m \vdash \downarrow B_m}\ \downarrow R}{\downarrow A_m \vdash \downarrow B_m}\ \downarrow L \qquad .$$

Similarly, given $\mathcal{D} : A_k \vdash B_k$, we can define $\uparrow\mathcal{D} : \uparrow A_k \vdash \uparrow B_k$ as follows:

$$\frac{\dfrac{A_k \geq k \quad \overset{\mathcal{D}}{A_k \vdash B_k}}{\uparrow A_k \vdash B_k}\ \uparrow L}{\uparrow A_k \vdash \uparrow B_k}\ \uparrow R \qquad .$$

It is easy to check that these are functorial—in particular, they preserve identities up to identity expansion and preserve composition up to cut reduction (using two commutative reductions and one principal reduction at either $\downarrow A_m$ or $\uparrow A_m$).

Now, in order to prove that $\downarrow \dashv \uparrow$, we will use the unit-counit formulation of adjunction and so we begin by defining the unit and counit. To show that there is an adjunction, it then only remains to show that these transformations are natural and that they satisfy the unit-counit laws.

Definition A.15. We define two transformations $\eta : \mathsf{id}_m \to \uparrow\downarrow$ and $\varepsilon : \downarrow\uparrow \to \mathsf{id}_k$ as follows:

Define $\eta_{A_m} : A_m \to \uparrow\downarrow A_m$ to be the equivalence class of

$$\frac{\dfrac{A_m \geq m \quad \dfrac{}{A_m \vdash A_m}\ \mathsf{id}_{A_m}}{A_m \vdash \downarrow A_m}\ \downarrow R}{A_m \vdash \uparrow\downarrow A_m}\ \uparrow R$$

Define $\varepsilon_{B_k} : \downarrow\uparrow B_k \to B_k$ to be the equivalence class of

$$\frac{\dfrac{B_k \geq k \quad \dfrac{}{B_k \vdash B_k}\ \mathsf{id}_{B_k}}{\uparrow B_k \vdash B_k}\ \uparrow L}{\downarrow\uparrow B_k \vdash B_k}\ \downarrow L$$

Proposition A.16. *η and ε are natural in their arguments.*

Proof. Suppose $\mathcal{D} : A_m \to B_m$, and consider the following square:

$$\begin{array}{ccc} A_m & \xrightarrow{\mathcal{D}} & B_m \\ \downarrow{\scriptstyle \eta_{A_m}} & & \downarrow{\scriptstyle \eta_{B_m}} \\ \uparrow\downarrow A_m & \xrightarrow{\uparrow\downarrow\mathcal{D}} & \uparrow\downarrow B_m \end{array}$$

We wish to show that this square commutes in that the proofs along each path are equivalent up to identity expansion and cut-reduction.

$$\dfrac{(x:A_m) \geq m \geq k \quad |\emptyset| \sim m \quad \dfrac{}{(x:A_m) \Vdash A_m}\,\mathsf{id}_{A_m} \quad \overset{\mathcal{E}}{\Psi' \Vdash C_k}}{(x:A_m)\,\Psi' \vdash C_k}\,\mathsf{cut}(\emptyset)$$

$$\Longrightarrow \quad \dfrac{\mathsf{W} \in \sigma(m) \quad \overset{\mathcal{E}}{\Psi' \vdash C_k}}{(x:A_m)\,\Psi' \vdash C_k}\,\mathsf{weaken}$$

$$\dfrac{\Psi \geq m \geq k \quad |\{y\}| \sim m \quad \overset{\mathcal{D}}{\Psi \Vdash A_m} \quad \dfrac{\mathsf{C} \in \sigma(m) \quad \overset{\mathcal{E}'}{\Psi'\,(z:A_m)\,(w:A_m) \Vdash C_k}}{\Psi'\,(y:A_m) \Vdash C_k}\,\mathsf{contract}}{\Psi\,\Psi' \vdash C_k}\,\mathsf{cut}(\{y\})$$

$$\Longrightarrow \quad \dfrac{\Psi'\,(z:A_m)\,(w:A_m) \geq m \geq k \quad |\{z,w\}| \sim m \quad \overset{\mathcal{D}}{\Psi \Vdash A_m} \quad \overset{\mathcal{E}'}{\Psi'\,(z:A_m)\,(w:A_m) \Vdash C_k}}{\Psi\,\Psi' \vdash C_k}\,\mathrm{i.h.}(A_m, \mathcal{D}, \mathcal{E}')(\{z,w\})$$

$$\dfrac{\Psi \geq m \geq k \quad |T \cup \{x,y\}| \sim m \quad \mathcal{D} = \dfrac{\overset{\mathcal{D}_1}{\ell \in I \quad \Psi \Vdash A^\ell_m}}{\Psi \Vdash \bigoplus\limits_{i\in I} A^i_m}\,\oplus R_\ell \quad \dfrac{\overset{\mathcal{E}_i}{\Psi'\,(T \cup \{x\} : \bigoplus\limits_{i\in I} A^i_m)\,(z:A^i_m) \Vdash C_k \text{ for each } i \in I}}{\Psi'\,(T \cup \{x,y\} : \bigoplus\limits_{i\in I} A^i_m) \Vdash C_k}\,\oplus L}{\Psi\Psi' \Vdash C_k}\,\mathsf{cut}(T \cup \{x,y\})$$

$$\Longrightarrow \quad \dfrac{\mathsf{C} \in \sigma(m) \quad \dfrac{\overset{\mathcal{D}_1}{\Psi \geq m \geq k \quad \Psi \Vdash A^\ell_m} \quad \dfrac{\Psi \geq m \geq k \quad |T \cup \{x\}| \sim m \quad \overset{\mathcal{D}}{\Psi \Vdash \bigoplus\limits_{i\in I} A^i_m} \quad \overset{\mathcal{E}_\ell}{\Psi'\,(T \cup \{x\} : \bigoplus\limits_{i\in I} A^i_m)(z:A^\ell_m) \Vdash C_k}}{\Psi\,\Psi'\,(z:A^\ell_m) \Vdash C_k}\,\mathrm{i.h.}(\ldots)}{\Psi\,\Psi\,\Psi' \Vdash C_k}\,\mathrm{i.h.}(A^\ell_m, \mathcal{D}_1, \ldots)(\{z\})}{\Psi\,\Psi' \Vdash C_k}\,\mathsf{contract}^*$$

$$\dfrac{\Psi \geq k \geq m \quad |S| \sim k \quad \overset{\mathcal{D}}{\Psi \Vdash C_k} \quad \dfrac{\overset{\mathcal{E}_1}{(x:A_m)\,(S:C_k)\,\Psi' \Vdash B_m}}{(S:C_k)\,\Psi' \Vdash A_m \multimap B_m}\,\multimap R}{\Psi\,\Psi' \Vdash A_m \multimap B_m}\,\mathsf{cut}(S)$$

$$\Longrightarrow \quad \dfrac{\Psi \geq k \geq m \quad |S| \sim k \quad \overset{\mathcal{D}}{\Psi \Vdash C_k} \quad \dfrac{\overset{\mathcal{E}_1}{(x:A_m)\,(S:C_k)\,\Psi' \Vdash B_m}}{(x:A_m)\,\Psi\,\Psi' \Vdash B_m}\,\mathrm{i.h.}(C_k, \mathcal{D}, \mathcal{E}_1)(S)}{\Psi\,\Psi' \Vdash A_m \multimap B_m}\,\multimap R$$

Figure 4: Sample Cut Reductions

Composing along the left-hand path, we get

$$\dfrac{\dfrac{A_m \geq m \quad \dfrac{}{A_m \vdash A_m}\,\mathsf{id}_{A_m}}{\dfrac{A_m \vdash \downarrow A_m}{A_m \vdash \uparrow\downarrow A_m}\,\uparrow R}\,\downarrow R \quad \dfrac{\downarrow A_m \geq k \quad \dfrac{\dfrac{\overset{\mathcal{D}}{A_m \geq m \quad A_m \vdash B_m}}{A_m \vdash \downarrow B_m}\,\downarrow L}{\downarrow A_m \vdash \downarrow B_m}\,\uparrow L}{\uparrow\downarrow A_m \vdash \downarrow B_m}\,\uparrow R}{A_m \vdash \uparrow\downarrow B_m}\,\mathsf{cut}_{\uparrow\downarrow A_m}\ .$$

Now, composing along the right-hand path, we get

$$\dfrac{\overset{\mathcal{D}}{A_m \vdash B_m} \quad \dfrac{\dfrac{B_m \geq m \quad \dfrac{}{B_m \vdash B_m}\,\mathsf{id}_{B_m}}{B_m \vdash \downarrow B_m}\,\downarrow R}{B_m \vdash \uparrow\downarrow B_m}\,\uparrow R}{A_m \vdash \uparrow\downarrow B_m}\,\mathsf{cut}_{B_m}\ .$$

Applying identity and cut reductions to both of these proofs until we have removed all instances of identity and of cut, we are

in both cases left with

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ A_m \geq m \quad A_m \vdash B_m\end{array}}{A_m \vdash \downarrow B_m}\downarrow R}{A_m \vdash \uparrow\downarrow B_m}\uparrow R$$

and so we indeed have that this square commutes.

That ε is natural follows from a similar argument. □

We now show that the unit-counit laws hold for η and ε, which will give us that $\downarrow \dashv \uparrow$.

That is, we show that the following diagrams commute:

$$\begin{array}{ccc} \downarrow & \xrightarrow{\downarrow\eta} & \downarrow\uparrow\downarrow \\ & {\scriptstyle \mathrm{id}_\downarrow} \searrow & \downarrow{\scriptstyle \varepsilon_\downarrow} \\ & & \downarrow \end{array} \qquad \text{and} \qquad \begin{array}{ccc} \uparrow & \xrightarrow{\eta\uparrow} & \uparrow\downarrow\uparrow \\ & {\scriptstyle \mathrm{id}_\uparrow} \searrow & \downarrow{\scriptstyle \uparrow\varepsilon} \\ & & \uparrow \end{array}$$

We consider first the former diagram. Let A_m be given, and note that it will suffice to show that $\mathrm{id}_{\downarrow A_m} = \varepsilon_{\downarrow A_m} \circ \downarrow\eta_{A_m}$.

This corresponds to showing that the following two proofs are equivalent up to cut reduction and identity expansion:

$$\dfrac{\dfrac{A_m \geq m \quad \dfrac{}{A_m \vdash A_m}\,\mathrm{id}_{A_m}}{A_m \vdash \downarrow A_m}\downarrow R}{\dfrac{A_m \vdash \uparrow\downarrow A_m}{\downarrow A_m \vdash \downarrow\uparrow\downarrow A_m}\downarrow L}\uparrow R \qquad \dfrac{\dfrac{\downarrow A_m \geq k \quad \dfrac{}{\downarrow A_m \vdash \downarrow A_m}\,\mathrm{id}_{\downarrow A_m}}{\uparrow\downarrow A_m \vdash \downarrow A_m}\uparrow L}{\downarrow\uparrow\downarrow A_m \vdash \downarrow A_m}\downarrow L \qquad \downarrow\uparrow\downarrow A_m \vdash \downarrow A_m}\downarrow R\ \mathrm{cut}$$

and

$$\dfrac{}{\downarrow A_m \vdash \downarrow A_m}\,\mathrm{id}_{\downarrow A_m}$$

As before, we can use cut reductions and identity expansions to show that both proofs are equivalent to

$$\dfrac{\dfrac{A_m \geq m \quad \dfrac{}{A_m \vdash A_m}\,\mathrm{id}_{A_m}}{A_m \vdash \downarrow A_m}\downarrow R}{\downarrow A_m \vdash \downarrow A_m}\downarrow L \quad ,$$

and so the triangle commutes, as desired.

The proof that the second triangle commutes is similar, and therefore omitted. □

A.4 Progress and Preservation

We present here several definitions and lemmas used in the proofs of progress and preservation.

For the progress theorem, it is convenient to have a second form of process typing which induces a list ordering, rather than a binary tree ordering, on configurations. Again, we require that the size of S be compatible with the mode of the providing channel. We will show that the two typing judgments are equivalent in Lemma A.17.

$$\dfrac{}{\Psi \vDash' (\cdot) :: \Psi} \qquad \dfrac{|S| \sim m \quad \Psi \vDash' C :: \Psi_1\Psi_2 \quad \Psi_2 \vdash P :: (a : A_m)}{\Psi \vDash' C\ \mathrm{proc}(S, \overline{\Psi_2}, a.P) :: \Psi_1(S : A_m)}$$

We now present some results about configuration typing, which will allow us to get useful information from the fact that a configuration is well-typed. As well-typedness of a configuration is the primary assumption for both progress and preservation, these lemmas are key in the proofs of those results.

Lemma A.17. $\Psi \vDash' C :: \Psi'$ *iff* $\Psi \vDash C :: \Psi'$.

Proof. We note that by applying the rules for $\vDash$, we can derive the following two proofs:

$$\dfrac{}{\Psi \vDash (\cdot) :: \Psi}$$

$$\dfrac{\Psi \vDash C :: \Psi_1\Psi_2 \quad \dfrac{|S| \sim m \quad \Psi_2 \vdash P :: (a : A_m)}{\Psi_1\Psi_2 \vDash \mathrm{proc}(S, \overline{\Psi_2}, a.P) :: \Psi_1(S : A_m)}}{\Psi \vDash C\ \mathrm{proc}(S, \overline{\Psi_2}, a.P) :: \Psi_1(S : A_m)}$$

Therefore, if we have that $\Psi \vDash' C :: \Psi'$, then the derived rules above can be used to show that $\Psi \vDash C :: \Psi'$.

Now, suppose that $\Psi \vDash C :: \Psi'$. By inducting on the derivation of $\Psi \vDash C :: \Psi'$, we can show that $\Psi \vDash' C :: \Psi'$. From the inductive step, we apply the inductive hypothesis to get proofs that (when $C = C_1C_2$) $\Psi \vDash' C_1 :: \Psi''$ and that $\Psi'' \vDash' C_2 :: \Psi'$. Since the rules for $\vDash'$ give only one possible construction for each of these proofs, it is possible to simply "glue" them together by replacing an instance of $\Psi'' \vdash (\cdot) :: \Psi'$ in the latter proof with the whole former proof to get the desired result. □

As this lemma justifies that $\vDash'$ and $\vDash$ are equivalent, we will use them interchangeably, simply writing $\vDash$ for either.

Lemma A.18. *If* $C = C_1\ C_2$ *and* $\Psi \vDash C :: \Psi''$*, then there is* Ψ' *such that* $\Psi \vDash C_1 :: \Psi'$ *and* $\Psi' \vDash C_2 :: \Psi''$.

Proof. It is easy to take a proof that $\Psi \vDash C_1\ C_2 :: \Psi''$ and break it into a proof that $\Psi \vDash C_1 :: \Psi'$ and $\Psi' \vDash C_2 :: \Psi''$ following the reverse of the "gluing" process used in the previous lemma. □

Lemma A.19. *If* $\Psi\ (x : A_m) \vDash C :: \Psi'\ (x : A_m)$*, then also* $\Psi \vDash C :: \Psi'$.

Proof. This follows from an induction on the derivation of $\Psi(x : A_m) \vDash C :: \Psi'(x : A_m)$, using uniqueness of channel names for the singleton case. □

Lemma A.20 (Configuration Inversion). *If*

$$(\cdot) \vDash C :: \Psi''(x : A_m),$$

then there are C_1, C_2 *and* T, Ψ', P *such that all of the following hold:*

(a) $C = C_1\ \mathrm{proc}(T \cup \{x\}, \overline{\Psi'}, b.P)\ C_2$.
(b) $(\cdot) \vDash C_1 :: \Psi\ \Psi'$.
(c) $\Psi' \vdash P :: (b : A_m)$.
(d) $\Psi\ (T \cup \{x\} : A_m) \vDash C_2 :: \Psi''\ (x : A_m)$.

Proof. This proof proceeds by induction on the derivation of $(\cdot) \vDash C :: \Psi''(x : A_m)$.

We first note that the last rule used cannot possibly be the rule for the empty configuration—this case fails to give the $(x : A_m)$ in the assumption. Similarly, if there is only one process in C, the derivation must have the form below:

$$\dfrac{\dfrac{}{(\cdot) \vDash (\cdot) :: (\cdot)} \quad \begin{array}{c}\mathcal{D}\\ |T \cup \{x\}| \sim m\end{array} \quad \begin{array}{c}\mathcal{E}\\ (\cdot) \vdash P :: (b : A_m)\end{array}}{(\cdot) \vDash \mathrm{proc}(T \cup \{x\}, \emptyset, b.P) :: (T \cup \{x\} : A_m)}$$

In this case, taking $C_1 = C_2 = (\cdot)$ gives the desired result.

Now, we are left with the cases where C contains more than one process, and so the derivation has the form

$$\frac{(\cdot) \vDash \overset{\mathcal{D}}{C'} :: \Psi\Psi' \quad \overset{\mathcal{E}}{|S|} \sim k \quad \overset{\mathcal{F}}{\Psi' \vdash P :: (c : B_k)}}{(\cdot) \vDash C' \ \mathrm{proc}(S, \overline{\Psi'}, c.P) :: \Psi(S : B_k)}$$

Note that $\Psi\ (S : B_k) = \Psi''\ (x : A_m)$, and so either $x \in S$ and $B_k = A_m$ or $(x : A_m)$ occurs in Ψ.

In the first case, we take $C_1 = C'$ and $C_2 = (\cdot)$, and $\mathrm{proc}(S, \overline{\Psi'}, c.P)$ is the desired process.

In the second case, we apply the inductive hypothesis to $\mathcal{D}$ to get C_1', C_2', T', Φ, Q satisfying (a)–(d). Taking $C_1 = C_1'$, $C_2 = C_2'\ \mathrm{proc}(\emptyset, \overline{\Psi'}, b.P)$, and $\mathrm{proc}(T' \cup \{x\}, \overline{\Phi}, a.Q)$ to be the desired process then gives the result. □