Semi-Axiomatic Sequent Calculus

Henry DeYoung
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA
hdeyoung@andrew.cmu.edu

Frank Pfenning
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA
fp@cs.cmu.edu

Klaas Pruiksma
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA
kpruiksm@andrew.cmu.edu

Abstract

We present the semi-axiomatic sequent calculus (SAX) that blends features of Gentzen's sequent calculus with an axiomatic formulation of intuitionistic logic. We develop and prove a suitable analogue to cut elimination and then show that a natural computational interpretation of SAX provides a simple form of shared memory concurrency.

2012 ACM Subject Classification Theory of computation → Proof theory; Computing methodologies → Concurrent programming languages

Keywords and phrases Sequent calculus, Curry-Howard isomorphism, shared memory concurrency

Digital Object Identifier 10.4230/LIPIcs.FSCD.2020.29

Funding This material is based upon work supported by the National Science Foundation under Grant No. 1718276.

Acknowledgements We would like to thank Siva Somayyajula and the anonymous reviewers for suggestions on an earlier version of this paper.

1 Introduction

The celebrated Curry-Howard isomorphism [8, 14] establishes a close connection between logic and computation. There are three interrelated components that define this correspondence: (1) propositions in the logic are interpreted as types, (2) proofs in the logic are interpreted as programs, and (3) proof reduction is interpreted as computation. Parts (2) and (3) mean that we must pay attention to the specific formulation of the logic. Curry used combinatory logic [9] which derives from an axiomatic formulation of inference and arrives at combinatory reduction. Howard [14] established the close connection between natural deduction and the simply-typed λ-calculus. As a further example, Herbelin [13] introduced LJT, a formulation of the sequent calculus with a stoup whose computational interpretation uses explicit substitutions. In all three of these cases, the logic is simply intuitionistic logic but its computational interpretation is quite different.

In this paper we add another entry to this list of correspondences. We introduce the semi-axiomatic sequent calculus (Sax) which blends features of the sequent calculus with axiomatic presentations of intuitionistic logic. We show that Sax satisfies a version of cut elimination, where cut-free proofs have the subformula property, thereby establishing the basics of the proof theory for Sax.

As for other inference systems, the proof of cut elimination contains the seeds of its operational interpretation. For example, normalization for natural deduction is based on a substitution operation that corresponds to β-reduction, and cut reduction on LJT corresponds to $\overline{\lambda}$-reduction in a calculus of explicit substitution. In Sax, a cut disappears entirely during

a principal cut reduction, which is only possible because certain special forms of analytic cuts we call *snips* are allowed to remain in cut-free proofs. This disappearing act can be explained when we think of the cut reduction itself as reading from or writing to memory, which are atomic actions. Under this view, we recognize that one operational interpretation of Sax equates proofs with processes where some actively compute and others are passive, thereby representing shared memory cells. We formalize the concurrent reduction strategy (that is, the evaluation relation) using techniques from substructural operational semantics [27, 5, 6], which can be mapped back directly to proofs (with a loss of readability). As a concurrent programming language, the interpretation of Sax then has the desired properties such as type preservation, progress, confluence, and termination.

Our results are yet another illustration of the flexibility of the sequent calculus as a foundation for computation at a high level of abstraction. Sax is remarkably simple: it requires no stoup or other structural devices, just the initial leap of faith to replace noninvertible rules by axioms. It also provides a path towards understanding a simple, logically grounded form of shared memory concurrency under a Curry-Howard interpretation.

In Section 2, we provide an overview of the standard sequent calculus G_3 for background, before relating it to Sax in Section 3. This section contains our first set of contributions: Sax itself, translations between G_3 and Sax, and a cut elimination result for Sax. We briefly discuss some example Sax proofs in Section 4, which also serve as examples of computations later on. Section 5 contains our second major set of contributions. We provide an operational interpretation of Sax using *shared memory* and prove the basic theorems of progress, preservation, and (because the semantics are nondeterministic) confluence. A termination theorem, analagous to normalization for natural deduction, is more involved, and is delayed to its own section (Section 6).

In prior work we have given a different interpretation of Sax (without investigating its metatheory from a logical and proof-theoretic perspective) using session types and message passing, starting with purely linear intuitionistic logic and generalizing all the way to adjoint logic, combining structural and substructural intuitionistic logics [24]. This in turn built on the Curry-Howard interpretation of linear logic as session-typed processes using *message-passing* concurrency [3, 29, 4]. (Modeling asynchronous communication in a Curry-Howard interpretation of linear logic was the original motivation behind Sax.) Starting from Herbelin's seminal work, others have also given computational interpretations of intuitionistic and classical sequent calculi at various level of abstraction (for example, with multiple conclusions and stoups [7] or with de Bruijn indices [2]). These are, however, quite different from the interpretation presented here.

2 Ordinary Sequent Calculus

For reference, we provide the standard sequent calculus G_3 [15] in Figure 1 so we can explicitly relate it to Sax. In G_3 all structural rules remain implicit, and antecedents Γ of a sequent $\Gamma \vdash A$ are propagated to all premises. For brevity, we omit the logical constants falsehood ($\bot$) and truth ($\top$) since they are not particularly interesting, especially from the computational perspective. We do, however, present two forms of conjunction: $A \otimes B$ and $A \mathbin{\&} B$, which are distinguished here by their left rules extrapolated from linear logic. Although these conjunctions are logically equivalent (that is, $A \otimes B \dashv\vdash A \mathbin{\&} B$), they are computationally distinct: functionally, $A \otimes B$ corresponds to eager pairs whereas $A \mathbin{\&} B$ corresponds to lazy pairs (see, for example, call-by-push-value [16]).

$$\frac{}{\Gamma, A \vdash A}\ \mathsf{Id}_A \qquad \frac{\Gamma \vdash A \quad \Gamma, A \vdash C}{\Gamma \vdash C}\ \mathsf{Cut}_A$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B}\ \supset R \qquad \frac{\Gamma, A \supset B \vdash A \quad \Gamma, A \supset B, B \vdash C}{\Gamma, A \supset B \vdash C}\ \supset L$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \otimes B}\ \otimes R \qquad \frac{\Gamma, A \otimes B, A, B \vdash C}{\Gamma, A \otimes B \vdash C}\ \otimes L$$

$$\frac{}{\Gamma \vdash \mathbf{1}}\ \mathbf{1}R \qquad \frac{\Gamma, \mathbf{1} \vdash C}{\Gamma, \mathbf{1} \vdash C}\ \mathbf{1}L$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ \vee R_1 \quad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ \vee R_2 \qquad \frac{\Gamma, A \vee B, A \vdash C \quad \Gamma, A \vee B, B \vdash C}{\Gamma, A \vee B \vdash C}\ \vee L$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \,\&\, B}\ \& R \qquad \frac{\Gamma, A \,\&\, B, A \vdash C}{\Gamma, A \,\&\, B \vdash C}\ \&L_1 \quad \frac{\Gamma, A \,\&\, B, B \vdash C}{\Gamma, A \,\&\, B \vdash C}\ \&L_2$$

Figure 1 The Sequent Calculus (G_3).

We also include $\mathbf{1}$, the unit of $A \otimes B$, which can be thought of as the nullary form of the binary $A \otimes B$. The $\mathbf{1}L$ rule may look a bit surprising: we might expect the antecedent $\mathbf{1}$ to be deleted in its premise. But since we have implicit weakening and contraction, the principal formula of every left rule, $\mathbf{1}L$ included, must be preserved. Preserving the antecedents turns out to be computationally significant in the shared memory interpretation that we present in Section 5.

We have the following standard theorems with standard proofs. Gentzen's original sequent calculus had explicit structural rules and used an intermediate system with a rule called Mix in the proof of cut elimination [11]. We sketch the proofs of admissibility of cut and cut elimination [21] that are more closely related in structure to the proof in Sax in Section 3.

▶ **Theorem 1** (Admissibility of Cut in G_3 [11, 21]). *If there are cut-free derivations $\Gamma \vdash A$ and $\Gamma, A \vdash C$ then there is a cut-free derivation of $\Gamma \vdash C$.*

Proof. By nested induction on the structure of A and then simultaneously on the derivations of $\Gamma \vdash A$ and $\Gamma, A \vdash C$. ◀

▶ **Theorem 2** (Cut Elimination [11, 21]). *If $\Gamma \vdash A$ then there is a cut-free derivation of $\Gamma \vdash A$.*

Proof. By induction on the structure of the given derivation, using Theorem 1 for cut. ◀

It is evident that the cut-free sequent calculus has the subformula property since, except for cut, the premises of all rules are subformulas of the propositions in the conclusions. This allows us to view the right- and left-rules for the logical connectives in G_3 as a compositional explanation for the meaning of propositions [10, 19]. Another component of such an interpretation is identity expansion (we require the identity rule Id_A only for atomic propositions), which is significant from the foundational perspective but not computationally interesting since identity expansions correspond to extensional equalities.

$$\frac{}{\Gamma, A \vdash A}\ \mathsf{Id}_A \qquad \frac{\Gamma \vdash A \quad \Gamma, A \vdash C}{\Gamma \vdash C}\ \mathsf{Cut}_A$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B}\ \supset R \qquad \frac{}{\Gamma, A, A \supset B \vdash B}\ \supset L^0$$

$$\frac{}{\Gamma, A, B \vdash A \otimes B}\ \otimes R^0 \qquad \frac{\Gamma, A \otimes B, A, B \vdash C}{\Gamma, A \otimes B \vdash C}\ \otimes L$$

$$\frac{}{\Gamma \vdash \mathbf{1}}\ \mathbf{1}R^0 \qquad \frac{\Gamma, \mathbf{1} \vdash C}{\Gamma, \mathbf{1} \vdash C}\ \mathbf{1}L$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \,\&\, B}\ \&R \qquad \frac{}{\Gamma, A \,\&\, B \vdash A}\ \&L_1^0 \qquad \frac{}{\Gamma, A \,\&\, B \vdash B}\ \&L_1^0$$

$$\frac{}{\Gamma, A \vdash A \vee B}\ \vee R_1^0 \qquad \frac{}{\Gamma, B \vdash A \vee B}\ \vee R_2^0 \qquad \frac{\Gamma, A \vee B, A \vdash C \quad \Gamma, A \vee B, B \vdash C}{\Gamma, A \vee B \vdash C}\ \vee L$$

Figure 2 The Semi-Axiomatic Sequent Calculus (Sax).

3 The Semi-Axiomatic Sequent Calculus

The connectives in intuitionistic logic can be divided into positive ($A \otimes B$, $\mathbf{1}$, $A \vee B$) and negative ($A \supset B$, $A \,\&\, B$), where we have split conjunction into two. The right rules for negative connectives and the left rules for the positive connectives are asynchronous (in the terminology of Andreoli [1]) in the sense that a negative connective in the succedent and a positive connective in the antecedent can always be broken down eagerly with the corresponding rule when constructing a proof bottom-up. Conversely, the right rules for positive connectives and the left rules for negative connectives may have to be postponed until they can be applied. Even though in the formulation in G_3 this is not strictly accurate, we say that negative right and positive left rules are *invertible* while positive right and negative left rules are *noninvertible* (see, for example, the analysis by Liang and Miller [17]).

The semi-axiomatic sequent calculus Sax arises from G_3 by replacing all the noninvertible rules ($\supset L, \&L, \otimes R, \mathbf{1}R, \vee R$) by corresponding axioms while leaving all the invertible rules ($\supset R, \&R, \otimes L, \mathbf{1}L, \vee L$) unchanged. Identity and cut also remain unchanged. We annotate all the rules that are now axioms with the superscript 0, indicating the zero premises of the rule. A summary of the rules can be found in Figure 2.

First, we should convince ourselves that G_3 and Sax have the same derivable sequents.

▶ **Theorem 3** (Translations between G_3 and Sax)**.** *$\Gamma \vdash A$ in G_3 iff $\Gamma \vdash A$ in Sax.*

Proof. In one direction, we can derive each new Sax axiom using the corresponding rules of G_3 and the identity. For example, we can derive $\supset L^0$ in G_3 as follows:

$$\frac{\dfrac{}{\Gamma, A, A \supset B \vdash A}\ \mathsf{Id}_A \quad \dfrac{}{\Gamma, A, A \supset B, B \vdash B}\ \mathsf{Id}_B}{\Gamma, A, A \supset B \vdash B}\ \supset L$$

The other direction requires the uses of cut in Sax to derive the rules of G_3. For example,

we can derive $\supset L$ in Sax as follows:

$$\dfrac{\dfrac{\Gamma, A \supset B \vdash A \quad \overline{\Gamma, A, A \supset B \vdash B}\ \supset L^0}{\Gamma, A \supset B \vdash B}\ \mathsf{Cut} \qquad \Gamma, A \supset B, B \vdash C}{\Gamma, A \supset B \vdash C}\ \mathsf{Cut}$$

All other cases are similar. ◀

Sax does not satisfy the standard cut elimination theorem. For example, there is no cut-free proof of $A \supset B, B \supset C \vdash A \supset C$. Instead, we have the following proof (omitting some unused antecedents):

$$\dfrac{\dfrac{\overline{A, A \supset B \vdash B}\ \supset L^0 \quad \overline{B, B \supset C \vdash C}\ \supset L^0}{A, A \supset B, B \supset C \vdash C}\ \mathsf{Cut}_B}{A \supset B, B \supset C \vdash A \supset C}\ \supset R$$

Despite not being cut-free, this proof does exhibit the subformula property – its only cut is *analytic*. However, we find admitting arbitrary analytic cuts in the normal forms of proofs is too lenient and does not provide a good correspondence to operational behavior under the Curry-Howard interpretation. Instead, we would like to recognize the subformulas of the principal formula of each axiom as specific formulas that we may cut without losing the subformula property. We then have a restricted form of cut we call Snip which requires that the cut formula in one (or both) of the premises originates from an axiom in this way.

In order to make this precise we need to track specific formula occurrences, so we label each antecedent and the succedent of each sequent in a derivation with variables, where all variables in a sequent are distinct. As a representative example of an axiom, we will examine the $\supset L^0$ axiom. We call the variables $y : A$ and $z : B$ in

$$\overline{\Gamma, y : A, x : A \supset B \vdash z : B}\ \supset L^0$$

eligible for a Snip (or just *eligible*) and propagate this information through the derivation. In a rule with two premises a variable is eligible in the conclusion if it is eligible in at least one of the premises. If we assume or prove that an antecedent or succedent is eligible we mark it with $x^* : A$ (although the absence of a $*$ does not mean that it is ineligible). This mark is not part of the syntax of sequents, just expressing an assumed or known property of its derivation.

Since we are not in a linear logic, the cut elimination proof also seems to require that we track *irrelevant* antecedents in sequents, those that are never used. These are the (ineligible) side formulas of axioms and Id, which then propagate through the derivation. In a rule with two premises, a variable is only irrelevant in the conclusion if it is irrelevant in both premises. If we assume or prove that an antecedent is irrelevant, we mark it with $x^0 : A$ (although again the absence of the mark does not mean it is relevant). We also write Γ^0 if all variables in Γ are irrelevant. Again, the superscript 0 is not part of the syntax of sequents, just expressing an assumed or known property of its derivation.

We refer to the *status* of variables as *irrelevant* or *relevant* and *eligible* or *ineligible*, where, by its definition, an eligible variable is automatically relevant.

In the normal form of derivations we now permit cuts where at least one of the two occurrences of the cut formula is eligible for a snip. The rules can be found in Figure 3.

$$\frac{}{\Gamma^0, x : A \vdash y : A}\ \mathsf{Id}_A \qquad \frac{\Gamma \vdash x : A \quad \Gamma, x : A \vdash z : C}{\Gamma \vdash z : C}\ \mathsf{Cut}_A$$

$$\frac{\Gamma \vdash x^* : A \quad \Gamma, x : A \vdash z : C}{\Gamma \vdash z : C}\ \mathsf{Snip}^1_A \qquad \frac{\Gamma \vdash x : A \quad \Gamma, x^* : A \vdash z : C}{\Gamma \vdash z : C}\ \mathsf{Snip}^2_A$$

$$\frac{\Gamma, y : A \vdash z : B}{\Gamma \vdash x : A \supset B}\ \supset R \qquad \frac{}{\Gamma^0, y^* : A, x : A \supset B \vdash z^* : B}\ \supset L^0$$

$$\frac{}{\Gamma^0, y^* : A, z^* : B \vdash x : A \otimes B}\ \otimes R^0 \qquad \frac{\Gamma, x : A \otimes B, y : A, z : B \vdash w : C}{\Gamma, x : A \otimes B \vdash w : C}\ \otimes L$$

$$\frac{}{\Gamma^0 \vdash x : \mathbf{1}}\ \mathbf{1}R^0 \qquad \frac{\Gamma, x : \mathbf{1} \vdash w : C}{\Gamma, x : \mathbf{1} \vdash w : C}\ \mathbf{1}L$$

$$\frac{\Gamma \vdash y : A \quad \Gamma \vdash z : B}{\Gamma \vdash x : A \,\&\, B}\ \&R \qquad \frac{}{\Gamma^0, x : A \,\&\, B \vdash y^* : A}\ \&L^0_1 \qquad \frac{}{\Gamma^0, x : A \,\&\, B \vdash z^* : B}\ \&L^0_2$$

$$\frac{}{\Gamma^0, x^* : A \vdash z : A \vee B}\ \vee R^0_1 \qquad \frac{}{\Gamma^0, y^* : B \vdash z : A \vee B}\ \vee R^0_2 \qquad \frac{\Gamma, z : A \vee B, x : A \vdash w : C \quad \Gamma, z : A \vee B, y : B \vdash w : C}{\Gamma, z : A \vee B \vdash w : C}\ \vee L$$

Figure 3 Sax in Labeled Form.

The implementation of $\supset L$ from $\supset L^0$ has two uses of cuts, but both of them are on eligible formulas and are therefore snips.

$$\dfrac{\dfrac{\Gamma, y : A \supset B \vdash x : A \quad \dfrac{}{\Gamma, x^* : A, y : A \supset B \vdash z^* : B}\ \supset L^0}{\Gamma, y : A \supset B \vdash z^* : B}\ \mathsf{Snip}^2_A \quad \Gamma, y : A \supset B, z : B \vdash w : C}{\Gamma, y : A \supset B \vdash w : C}\ \mathsf{Snip}^1_B$$

From now on when we say "cut-free" we mean that a derivation may not use Cut, but is allowed to use Snip in both of its forms. In its most pedantic version, the cut-free proof of our example would be

$$\dfrac{\dfrac{\dfrac{}{u^* : A, x : A \supset B, y^0 : B \supset C \vdash t^* : B}\ \supset L^0 \quad \dfrac{}{u^0 : A, x^0 : A \supset B, t^* : B, y : B \supset C \vdash w^* : C}\ \supset L^0}{u^* : A, x : A \supset B, y : B \supset C \vdash w^* : C}\ \mathsf{Snip}_B}{x : A \supset B, y : B \supset C \vdash z : A \supset C}\ \supset R$$

It so happens that in this example, the Snip is an instance of both Snip^1 and Snip^2. Also, in the final conclusion none of the variables are eligible or irrelevant.

The translations between Sax and G_3 give us one way to prove cut elimination for Sax.

▶ **Theorem 4** (Cut Elimination in Sax, v1). *If $\Gamma \vdash x : A$ in Sax then there is a cut-free derivation of $\Gamma \vdash x : A$ in Sax.*

Proof. We translate the given derivation from Sax to G_3. This introduces additional uses of the identity but mostly preserves the structure of the derivation. Now we apply cut elimination (Theorem 2) to obtain a cut-free derivation in G_3. The backwards translation (see the proof of Theorem 3) of the result to Sax introduces some snips but no cuts. The result is therefore cut-free in Sax. ◀

This proof induces a simple algorithm for cut elimination but it does so in an indirect way, via two translations. We are instead interested in understanding the computational behavior of Sax directly, so we look for a direct algorithm for cut elimination. This proof (and the algorithm it embodies) is somewhat more complex for Sax than for G_3 because it needs to allow snips but not general cuts.

As with G_3, we proceed in two steps: first, we show the admissibility of cut in the cut-free calculus, and then we prove cut elimination using the admissibility of cut.

▶ **Theorem 5** (Admissibility of Cut in Sax)**.** *If there are cut-free derivations* $\Gamma \vdash x : A$ *and* $\Gamma, x : A \vdash z : C$ *then there exists a cut-free derivation of* $\Gamma \vdash z : C$*.*

Proof. For readability, we express the construction of $\mathcal{F}$ from $\mathcal{D}$ and $\mathcal{E}$ in the form

$$\dfrac{\overset{\mathcal{D}}{\Gamma \vdash x : A} \quad \overset{\mathcal{E}}{\Gamma, x : A \vdash z : C}}{\Gamma \vdash z : C}\ \mathsf{Cut}_A \qquad \Longrightarrow \qquad \overset{\mathcal{F}}{\Gamma \vdash z : C}$$

The proof proceeds by a nested induction, first on the structure of A and then on the structure of the first and second given derivations, $\mathcal{D}$ and $\mathcal{E}$. We exploit that adding or subtracting an irrelevant antecedent ($\mathcal{D} + \{y^0 : B\}$ and $\mathcal{D} - \{y^0 : B\}$) does not change the structure of a derivation.

However, a direct proof does not work – we need to generalize our induction hypothesis. We observe that in the resulting derivation $\mathcal{F}$, the status of variables in Γ and $z : C$ may be different from their status in $\mathcal{D}$ and $\mathcal{E}$. If a variable becomes *irrelevant* we impose no condition. If a variable $y : B$ transitions from *eligible* in $\mathcal{D}$ or $\mathcal{E}$ to *ineligible but relevant* in $\mathcal{F}$, then we demand that $B < A$, that is, B be a strict subformula of A. This will allow us to apply the induction hypothesis to B in these cases, but it also requires that this condition is preserved in each case of the proof. See Appendix A for additional proof details. ◀

▶ **Theorem 6** (Cut Elimination in Sax, v2)**.** *If there is a derivation of* $\Gamma \vdash x : A$ *then there is a cut-free derivation of the same sequent.*

Proof. By a standard induction on the structure of the deduction, appealing to the admissibility of cut in the case of a cut. A cut in the original derivation could turn into a snip or be eliminated entirely, based on the eligibility of the cut formula in the two premises after appeals to the induction hypothesis. ◀

▶ **Theorem 7** (Subformula Property in Sax)**.** *All formulas in a cut-free derivation of* $\Gamma \vdash x : A$ *are subformulas of formulas in* Γ *or* A*.*

Proof. We generalize the induction hypothesis to include:

> For any eligible antecedent or succedent $y^* : B$ in any sequent in the derivation, B is a subformula of at least one of the remaining (ineligible) formulas in the sequent.

Then we proceed by induction over the structure of the derivation. For a snip, we use the eligibility requirement and the second part of the induction hypothesis to conclude that the cut formula is a subformula in both premises. Also, the second part of the induction hypothesis is manifestly true in all axioms and propagated by all the rules. ◀

4 Some Example Derivations

As an example, we show that $A \otimes B \dashv\vdash A \,\&\, B$. These proofs will have some interesting computational content examined in Appendix B. The first proof has three axioms and two uses of snip.

$$\dfrac{\dfrac{\dfrac{}{x : A \,\&\, B \vdash y^* : A}\,\&L_1^0 \quad \dfrac{}{x : A \,\&\, B \vdash z^* : B}\,\&L_2^0}{x : A \,\&\, B, y^* : A \vdash w : A \otimes B}\,\mathsf{Snip} \quad \dfrac{}{x^0 : A \,\&\, B, y^* : A, z^* : B \vdash w : A \otimes B}\,\otimes R^0}{x : A \,\&\, B \vdash w : A \otimes B}\,\mathsf{Snip}$$

We have annotated the proof with eligibility information and notice that in both snips it so happens the variables are eligible on both sides. This proof is cut-free according to our criterion since it only contains snips and not general cuts. The following proof for the other direction is also cut-free, but contains only rules from the usual sequent calculus.

$$\dfrac{\dfrac{\dfrac{}{x^0 : A \otimes B, y : A, z^0 : B \vdash u : A}\,\mathsf{Id}_A \quad \dfrac{}{x^0 : A \otimes B, y^0 : A, z : B \vdash v : B}\,\mathsf{Id}_B}{x^0 : A \otimes B, y : A, z : B \vdash w : A \,\&\, B}\,\&R}{x : A \otimes B \vdash w : A \,\&\, B}\,\otimes L$$

As a final example, consider (this portion of) the proof of $(A \supset C) \,\&\, (B \supset C) \vdash (A \vee B) \supset C$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{p : (A \supset C) \,\&\, (B \supset C) \vdash r^* : A \supset C}\,\&L_1^0 \quad \dfrac{}{r : A \supset C, x^* : A \vdash z^* : C}\,\supset L^0}{p : (A \supset C) \,\&\, (B \supset C), x^* : A \vdash z^* : C}\,\mathsf{Snip} \quad \vdots}{p : (A \supset C) \,\&\, (B \supset C), s : A \vee B \vdash z^* : C}\,\vee L}{p : (A \supset C) \,\&\, (B \supset C) \vdash q : (A \vee B) \supset C}\,\supset R$$

5 A Shared Memory Interpretation

The key idea behind the shared memory interpretation is that at runtime, variables will be substituted by *addresses in shared memory*. Moreover, a sequent

$$x_1 : A_1, \ldots, x_n : A_n \vdash z : C$$

defines the interface to a process P that *reads* from addresses $x_1, \ldots, x_n$ and *writes* to address z. The types of the variables define the shape of the contents of memory at the given address. Once z has been written to, the process P terminates because it has completed its task. We sometimes refer to $x_1, \ldots, x_n$ as the *sources* and z as the *destination* for P.

True to the Curry-Howard interpretation, P can be read off from the derivation of the sequent. We incorporate a process expression P into the judgment by writing

$$x_1 : A_1, \ldots, x_n : A_n \vdash P :: (z : C)$$

Now, the sequent can be seen as a typing judgment for the process expression P.

Cut. To understand the operational behavior of the processes assigned to sequents, we have to study the cut reductions. We begin with the rule of cut itself, without distinguishing snip as a special case. Cut is a first-class rule (unlike in the proof of admissibility of cut), so it has a corresponding process. This is because cut reduction corresponds to computation, so if we did not have cut we would have no computation.

$$\dfrac{\Gamma \vdash P :: (x : A) \quad \Gamma, x : A \vdash Q :: (z : C)}{\Gamma \vdash (x \leftarrow P \,;\, Q) :: (z : C)}\,\mathsf{Cut}_A$$

A process executing $x \leftarrow P\ ;\ Q$ will allocate a new cell in memory with address a, then spawn a new process $[a/x]P$ (which will write a), and continue as $[a/x]Q$ (which may read from a). Reading from a will be an act of *synchronization*, because $[a/x]Q$ cannot read from a until the value has been written by $[a/x]P$.

This is the only point in the dynamics where a new memory cell is allocated. Initially, it is shared between two processes, $[a/x]P$ and $[a/x]Q$. However, we also notice that Γ, in accordance with the usual presentation of (nonlinear) intuitionistic sequent calculus, is propagated to both premises. Dynamically, this means any cell with an address in Γ is accessible to both new processes. On the other hand, the succedent of a sequent is always a singleton, which leads us to the conclusion:

> *A cell with address a will be written by one distinguished process and may be read by many different processes.*

This observation will have consequences when we consider other rules.

Identity. The rule of identity has a straightforward operational interpretation.

$$\frac{}{\Gamma, x : A \vdash (y^W \leftarrow x^R) :: (y : A)}\ \mathsf{Id}_A$$

The process we assign reads from x and writes to y which amounts to just copying the value at address x to memory at address y. After it has written to y it terminates. The superscripts W and R would presumably not be part of the concrete syntax of a language, but remind us that this process reads from x and writes to y.

If we examine the cut reductions $\_/\mathsf{Id}$ and $\mathsf{Id}/\_$ in the proof of Theorem 5 (shown in Appendix A) we see that this is a considerable restriction of the general reduction rules. This exemplifies a common phenomenon when we relate pure proof theory to computation: some rules of cut reduction may be entirely dropped (such as the so-called permutative reductions), while others are restricted to superimpose a particular strategy on the general notion of reduction.

Logical Rules. The general interpretation of the left and right rules is:

> *Process expressions assigned to right rules will* write *to memory while expressions assigned to left rules will* read *from memory.*

The question in each case is what to write to or read from memory, and how to subsequently continue execution. We will examine this for each connective in turn.

Positive Conjunction. We start with the positive conjunction $A \otimes B$ because it is a little easier to understand than implication. As one might expect given the general guideline, the right rule should write a pair to memory, and it does!

$$\frac{}{\Gamma^0, x^* : A, y^* : B \vdash z^W.\langle x, y\rangle :: (z : A \otimes B)}\ \otimes R^0$$

The expression $z^W.\langle x, y\rangle$ writes the pair $\langle x, y\rangle$ to the cell at location z and terminates. The superscript W is there to remind us that we write to the cell z. No other cell is written to or read from. It may also be helpful to directly think of $z^W.\langle x, y\rangle$ as a representation of the memory cell z with contents $\langle x, y\rangle$. Note that the value $\langle x, y\rangle$ itself just contains two addresses x and y, not complex terms. The cells with these addresses may still be empty when we write the pair to z, which allows for a high degree of parallelism.

Conversely, the expression assigned to the rule $\otimes L$

$$\frac{\Gamma, z : A \otimes B, x : A, y : B \vdash P :: (w : C)}{\Gamma, z : A \otimes B \vdash \mathbf{case}\ z^R\ (\langle x, y\rangle \Rightarrow P) :: (w : C)}\ \otimes L$$

reads such a pair from memory at location z, matches, it against $\langle x, y\rangle$ to extract the components (say addresses a and b) and continues with $[a/x, b/y]P$. The cell z is persistent, so it may be read again later, either by this process or by another one. Again, the cells at addresses x and y may not yet have been filled, but we can nevertheless extract and manipulate their addresses.

The principal cut reduction of $\otimes R^0$ against $\otimes L$, expressed on processes, becomes

$$z \leftarrow (z^W.\langle a, b\rangle)\ ;\ \mathbf{case}\ z^R\ (\langle x, y\rangle \Rightarrow P) \quad \longrightarrow \quad [a/x, b/y]P$$

which is precisely the intended operational semantics.

Implication. Implication represents somewhat of a challenge to intuition, and is perhaps the reason that this form of sequent calculus and its shared memory interpretation has been overlooked. We start with the left rule $\supset L^0$ which, according to our guiding principle, should read from memory.

$$\frac{}{\Gamma, y^* : A, x : A \supset B \vdash x^R.\langle y, z\rangle :: (z^* : B)}\ \supset L^0$$

The process expression $x^R.\langle y, z\rangle$ should read a value of type $A \supset B$ from location x and pass it the pair y and z. But what does this pair represent? y is the (usual) argument to the function, having type A. And z is the *destination for the result of the function*. As such, every function takes an additional argument. This is reminiscent of continuation-passing style [28] except that instead of passing a *continuation to accept the function's result* we pass a *destination address to store the function's result*.

Note that we have reused the syntax for pairs, except that the process assigned here *reads* from x. It is economical for the $\supset R$ rule to also reuse the same syntax to describe the augmented functions which we call *continuations*.

$$\frac{\Gamma, y : A \vdash P :: (z : B)}{\Gamma \vdash \mathbf{case}\ x^W\ (\langle y, z\rangle \Rightarrow P) :: (x : A \supset B)}\ \supset R$$

The process $\mathbf{case}\ c^W\ (\langle y, z\rangle \Rightarrow P)$ writes the continuation $(\langle y, z\rangle \Rightarrow P)$ to the cell at address c and terminates.

The cut reduction for $\supset R$ against $\supset L^0$, expressed directly on processes, is symmetric to the reduction for $\otimes R^0$ against $\otimes L$:

$$z \leftarrow (\mathbf{case}\ z^W\ (\langle x, y\rangle \Rightarrow P))\ ;\ z^R.\langle a, b\rangle \quad \longrightarrow \quad [a/x, b/y]P$$

Disjunction and Negative Conjunction. Just like implication and positive conjunction form a symmetric pair of expressions, reversing the role of read and write, so do disjunction and negative conjunction. The rules can be found in Figure 4, which summarizes all of the rules for Sax. We show here the reductions for $A \,\&\, B$, and the reductions for $A \vee B$ are symmetric, as they were for $A \supset B$ and $A \otimes B$.

$$\begin{array}{lcl} z \leftarrow (\mathbf{case}\ z^W\ (\pi_1(x) \Rightarrow P \mid \pi_2(y) \Rightarrow Q))\ ;\ z^R.\pi_1(a) & \longrightarrow & [a/x]P \\ z \leftarrow (\mathbf{case}\ z^W\ (\pi_1(x) \Rightarrow P \mid \pi_2(y) \Rightarrow Q))\ ;\ z^R.\pi_2(b) & \longrightarrow & [b/x]Q \end{array}$$

$$\frac{}{\Gamma, x : A \vdash (y^W \leftarrow x^R) :: (y : A)}\ \mathsf{Id}_A \qquad \frac{\Gamma \vdash P :: (x : A) \quad \Gamma, x : A \vdash Q :: (z : C)}{\Gamma \vdash (x \leftarrow P\ ;\ Q) :: (z : C)}\ \mathsf{Cut}_A$$

$$\frac{\Gamma, y : A \vdash P :: (z : B)}{\Gamma \vdash \mathbf{case}\ x^W\ (\langle y, z\rangle \Rightarrow P) :: (x : A \supset B)}\ \supset R \qquad \frac{}{\Gamma, y^* : A, x : A \supset B \vdash (x^R.\langle y, z\rangle) :: (z^* : B)}\ \supset L^0$$

$$\frac{}{\Gamma^0, y^* : A, z^* : B \vdash x^W.\langle y, z\rangle :: (x : A \otimes B)}\ \otimes R^0 \qquad \frac{\Gamma, x : A \otimes B, y : A, z : B \vdash P :: (w : C)}{\Gamma, x : A \otimes B \vdash \mathbf{case}\ x^R\ (\langle y, z\rangle \Rightarrow P) :: (w : C)}\ \otimes L$$

$$\frac{}{\Gamma^0 \vdash x^W.\langle\rangle :: (x : \mathbf{1})}\ \mathbf{1}R^0 \qquad \frac{\Gamma, x : \mathbf{1} \vdash P :: (w : C)}{\Gamma, x : \mathbf{1} \vdash \mathbf{case}\ x^R\ (\langle\rangle \Rightarrow P) :: (w : C)}\ \mathbf{1}L$$

$$\frac{\Gamma \vdash P :: (y : A) \quad \Gamma \vdash Q :: (z : B)}{\Gamma \vdash \mathbf{case}\ x^W\ (\pi_1(y) \Rightarrow P \mid \pi_2(z) \Rightarrow Q) :: (x : A \,\&\, B)}\ \&R \qquad \frac{}{\Gamma^0, x : A \,\&\, B \vdash x^R.\pi_1(y) :: (y^* : A)}\ \&L_1^0 \qquad \frac{}{\Gamma^0, x : A \,\&\, B \vdash x^R.\pi_2(z) :: (z^* : B)}\ \&L_2^0$$

$$\frac{}{\Gamma^0, x^* : A \vdash z^W.\pi_1(x) :: (z : A \vee B)}\ \vee R_1^0 \qquad \frac{}{\Gamma^0, y^* : B \vdash z^W.\pi_2(y) :: (z : A \vee B)}\ \vee R_2^0$$

$$\frac{\Gamma, z : A \vee B, x : A \vdash P :: (w : C) \quad \Gamma, z : A \vee B, y : B \vdash Q :: (w : C)}{\Gamma, z : A \vee B \vdash \mathbf{case}\ z^R\ (\pi_1(x) \Rightarrow P \mid \pi_2(y) \Rightarrow Q) :: (w : C)}\ \vee L$$

Figure 4 Process Expression Assignment for Sax.

Processes	P	$::= x \leftarrow P\ ;\ Q$	allocate a, spawn $[a/x]P$, continue as $[a/x]Q$
		$\mid\ x^W \leftarrow y^R$	copy contents of cell y into cell x
		$\mid\ x^W.V$	write V to cell x
		$\mid\ \mathbf{case}\ x^R\ K$	read value V from cell x and pass it to K
		$\mid\ \mathbf{case}\ x^W\ K$	write continuation K to cell x
		$\mid\ x^R.V$	read continuation K from x and pass V to it
Continuations	K	$::= (\langle x, y\rangle \Rightarrow P)$	$(\otimes L, \supset R)$
		$\mid\ (\langle\rangle \Rightarrow P)$	$(\mathbf{1}L)$
		$\mid\ (\pi_1(x) \Rightarrow P \mid \pi_2(x) \Rightarrow Q)$	$(\&R, \vee L)$
Values	V	$::= \langle x, y\rangle$	$(\otimes R^0, \supset L^0)$
		$\mid\ \langle\rangle$	$(\mathbf{1}R^0)$
		$\mid\ \pi_1(x) \mid \pi_2(x)$	$(\&L^0, \vee R^0)$
Cell Contents	W	$::= V \mid K$	

Figure 5 The Grammar for Sax Process Expressions.

5.1 Values and Continuations, Cells and Processes

The language of process expressions is now complete, but the immediate cut reductions do not yet fully capture the intended semantics. We first refactor the definition of our language slightly, by separating small values V from continuations K. Figure 5 also now shows what the new axioms of Sax represent: *values*. Meanwhile, the unchanged invertible rules represent *continuations*.

A key operation is in the semantics is passing a value to a continuation, which we write as $V \triangleright K$. It is defined by the following clauses:

$$\begin{array}{rcll} \langle a, b\rangle & \triangleright & (\langle x, y\rangle \Rightarrow P) & = & [a/x, b/y]P \\ \langle\,\rangle & \triangleright & (\langle\,\rangle \Rightarrow P) & = & P \\ \pi_1(a) & \triangleright & (\pi_1(x) \Rightarrow P \mid \pi_2(y) \Rightarrow Q) & = & [a/x]P \\ \pi_2(b) & \triangleright & (\pi_1(x) \Rightarrow P \mid \pi_2(y) \Rightarrow Q) & = & [b/y]Q \end{array}$$

The difficulty with the raw cut reductions in the presence of contraction (whether implicit as in G_3 and Sax, or with an explicit rule) is that some of them duplicate derivations. Instead, we would like them to be *shared*, which is exactly what the notion of shared memory allows us to do. One can feed this back into proof theory using the notion of multicut [24]. Here, we represent multiple cuts, and simultaneous cuts of one derivation against multiple others with the concept of a *configuration*. Such a configuration can be unravelled back into ordinary cuts (and therefore ordinary derivations), but at the cost of duplicating derivations.

A *configuration* consists of allocated memory cells (some of which may have been written to and some not) and executing processes. We present it as a substructural operational semantics (SSOS) [5, 27] in the form of multiset rewriting rules [6], using the following semantic objects.

$\mathsf{proc}\ c\ P$	process P with destination c
$\mathsf{cell}\ c\ \_$	memory cell c, allocated but not yet written
$!\mathsf{cell}\ c\ W$	cell c with contents W

In a configuration, a process $\mathsf{proc}\ c\ P$ is always paired with an allocated but not yet written $\mathsf{cell}\ c\ \_$. We also have cells $!\mathsf{cell}\ c\ W$ that have been written already. They are *persistent* (indicated by the exclamation mark) since they may be read multiple times but cannot be written again. In the multiset rewriting rules, a left-hand side of the form $!\phi$ will remain in the configuration, while objects ψ are removed and replaced by the objects on the right-hand side of the rule. All addresses c with objects $\mathsf{cell}\ c\ \_$ or $!\mathsf{cell}\ c\ W$ in a given configuration must be distinct, that is, no two cells in a configuration may share the same address.

The transitions in Figure 6 are multiset rewriting rules describing the dynamics of configurations. They can be applied to any subconfiguration, which induces a form of concurrency. However, the rules are confluent (see Theorem 10) so the result of reducing a configuration via these rules is ultimately deterministic, modulo the names of freshly introduced variables.

$$\begin{array}{lcll} \mathsf{proc}\ c\ (x \leftarrow P\ ;\ Q) & \longrightarrow & \mathsf{proc}\ a\ ([a/x]P), \mathsf{cell}\ a\ \_, \mathsf{proc}\ c\ ([a/x]Q) & (a \text{ fresh}) \\ !\mathsf{cell}\ b\ W, \mathsf{proc}\ a\ (a^W \leftarrow b^R), \mathsf{cell}\ a\ \_ & \longrightarrow & !\mathsf{cell}\ a\ W & \\ \mathsf{proc}\ a\ (\mathbf{case}\ a^W\ K), \mathsf{cell}\ a\ \_ & \longrightarrow & !\mathsf{cell}\ a\ K & \\ !\mathsf{cell}\ a\ K, \mathsf{proc}\ c\ (a^R.V) & \longrightarrow & \mathsf{proc}\ c\ (V \triangleright K) & \\ \\ !\mathsf{cell}\ a\ V, \mathsf{proc}\ a\ (\mathbf{case}\ a^R\ K) & \longrightarrow & \mathsf{proc}\ a\ (V \triangleright K) & \\ \mathsf{proc}\ a\ (a.V), \mathsf{cell}\ a\ \_ & \longrightarrow & !\mathsf{cell}\ a\ V & \end{array}$$

Figure 6 Dynamic Semantics of Configurations.

$$\frac{}{\Gamma \vdash (\cdot) :: \Gamma}\ \mathsf{Empty} \qquad \frac{\Gamma_0 \vdash \mathcal{C}_1 :: \Gamma_1 \quad \Gamma_1 \vdash \mathcal{C}_2 :: \Gamma_2}{\Gamma_0 \vdash (\mathcal{C}_1, \mathcal{C}_2) :: \Gamma_2}\ \mathsf{Join}$$

$$\frac{\Gamma \vdash P :: (a : A)}{\Gamma \vdash (\mathsf{proc}\ a\ P, \mathsf{cell}\ a\ \_) :: (\Gamma, a : A)}\ \mathsf{Proc}$$

$$\frac{\Gamma \vdash a^W.V :: (a : A)}{\Gamma \vdash \mathsf{!cell}\ a\ V :: (\Gamma, a : A)}\ \mathsf{Cell}_V \qquad \frac{\Gamma \vdash \mathbf{case}\ a^W\ K :: (a : A)}{\Gamma \vdash \mathsf{!cell}\ a\ K :: (\Gamma, a : A)}\ \mathsf{Cell}_K$$

Figure 7 Typing Rules for Configurations.

5.2 Typing Configurations

Configurations, like cells and processes, are runtime artifacts which can nevertheless be typed and also put into correspondence with the sequent calculus. First, typing. We have

Configurations	$\mathcal{C}$	::=	$\mathsf{proc}\ a\ P, \mathsf{cell}\ a\ \_$	process with destination a
		$\mid$	$\mathsf{!cell}\ a\ W$	cell with contents W
		$\mid$	$\cdot$	empty configuration
		$\mid$	$\mathcal{C}_1, \mathcal{C}_2$	joining configurations

The concatenation operator "," for configurations is commutative and associative with unit "$\cdot$", which makes it a suitable basis for multiset rewriting. However, a typing derivation for a configuration imposes an order by requiring that the writer of a cell a comes before all the readers of the cell. We include the rules for typing the contents of cells below those for typing configurations in Figure 7.

We can now state and prove several key results regarding our dynamics.

▶ **Theorem 8** (Preservation). *If* $\Gamma_0 \vdash \mathcal{C} :: \Gamma$ *and* $\mathcal{C} \longrightarrow \mathcal{C}'$ *then* $\Gamma_0 \vdash \mathcal{C} :: \Gamma'$ *for some* $\Gamma' \supseteq \Gamma$.

Proof. A first key property is that if $\Gamma_0 \vdash \mathcal{C} :: \Gamma$ then $\Gamma_0 \subseteq \Gamma$, which is easily proved by induction on the typing derivation. Therefore, a cell with address a and any process tasked with writing to a (which need not exist if the cell has already been filled) will always come to the left of any reader of a. This and the persistence of $\mathsf{!cell}\ a\ W$ easily yield preservation by induction on the typing derivation and inversion on the typing of processes and cell contents. ◀

We say a configuration is *final* if it consists only of cells $\mathsf{!cell}\ a\ W$. In other words, there are no longer any running processes. We prove progress for *closed* configurations ($\cdot \vdash \mathcal{C} :: \Gamma$) that do not depend on any undefined addresses.

Note that the typing derivation for a configuration is associative with respect to rule Join and unit Empty. This has two useful consequences. First, this provides a simple way to reconstruct a proof tree from a (well-typed) configuration – an empty configuration yields an empty tree, the Proc and Cell rules are base cases, each becoming the proof tree in the premise, and the Join rule simply cuts together the proof trees on the left and the right. Second, because of the associativity, we may perform induction where we isolate the rightmost cell or process and apply the inductive hypothesis to the remaining configuration to the left.

▶ **Theorem 9** (Progress). *If* $\cdot \vdash \mathcal{C} :: \Gamma$ *then either* $\mathcal{C}$ *is final or* $\mathcal{C} \longrightarrow \mathcal{C}'$ *for some* $\mathcal{C}'$.

Proof. By right-to-left induction over the typing derivation for $\mathcal{C}$.
Case: The rightmost rule is Cell, so $\mathcal{C} = (\mathcal{C}_1, \mathsf{!cell}\ a\ W)$. By induction hypothesis, either $\mathcal{C}_1$ is final (and so is then $\mathcal{C}$) or $\mathcal{C}_1 \longrightarrow \mathcal{C}_1'$ and therefore also $\mathcal{C} \longrightarrow \mathcal{C}_1', \mathsf{!cell}\ a\ W$.

Case: The rightmost rule is Proc, so $\mathcal{C} = (\mathcal{C}_1, \mathsf{proc}\ a\ P, \mathsf{cell}\ a\ \_)$. If $\mathcal{C}_1 \longrightarrow \mathcal{C}_1'$ then also $\mathcal{C} \longrightarrow (\mathcal{C}_1', \mathsf{proc}\ a\ P, \mathsf{cell}\ a\ \_)$. If $\mathcal{C}_1$ is final then we distinguish cases on P. If P is alloc/spawn, copy, or a process that writes, then $\mathcal{C} \longrightarrow \mathcal{C}'$ for some $\mathcal{C}'$. If P is a process that reads, then we apply inversion on the typing derivations of the cell a and the process P to show that a reduction is once again possible. ◀

We say $\mathcal{C}_1 \sim \mathcal{C}_2$ if there is a renaming ρ such that $\rho\,\mathcal{C}_1 = \mathcal{C}_2$.

▶ **Theorem 10** (Diamond Property). *Assume $\Delta \vdash \mathcal{C} :: \Gamma$. If $\mathcal{C} \longrightarrow \mathcal{C}_1$ and $\mathcal{C} \longrightarrow \mathcal{C}_2$ such that $\mathcal{C}_1 \not\sim \mathcal{C}_2$. Then there exist $\mathcal{C}_1'$ and $\mathcal{C}_2'$ such that $\mathcal{C}_1 \longrightarrow \mathcal{C}_1'$ and $\mathcal{C}_2 \longrightarrow \mathcal{C}_2'$ with $\mathcal{C}_1' \sim \mathcal{C}_2'$.*

Proof. The proof is straightforward by cases. There are no critical pairs involving ephemeral (that is, non-persistent) objects in the left-hand sides. ◀

As usual, confluence of multistep reduction follows by two standard inductions from the diamond property.

We now return to the distinction between general cuts and snips, which we introduced in order to establish cut elimination. First, we note that the transition rule for cut applies equally whether the cut $x \leftarrow P\ ;\ Q$ is a snip or not. In order to discuss the other rules, we define that an occurrence of an *address a is eligible* if it is eligible in the typing derivation according to the rules in Figure 7. This coincides with saying that the corresponding variable would be eligible if we unwound the configuration into a collection of proofs. We notice that in an object $!\mathsf{cell}\ a\ W$ the address a is never eligible because it labels the principal formula of an inference. Furthermore, none of the reductions besides cut involve an eligible address a, since they all label either an application of identity or the principal proposition of an inference.

It remains to characterize final configurations, that is, those consisting only of cells. We might at first suspect that all remaining cuts are snips, but that is not true because the dynamics does not reduce continuations K. This reflects a common difference between pure proof theory (where we show full normalization or cut elimination) and the dynamics of programming languages (where we do not evaluate under abstractions). We call addresses that occur in values V *observable* and those that occur in continuations K *hidden*.

▶ **Theorem 11** (Observable Addresses). *All observable addresses in a well-typed final configuration are eligible.*

Proof. By inversion on the typing of cells containing values V. ◀

From this, we obtain a simple corollary for *purely positive types*. In a functional language, *values* of purely positive type are *observable* in their entirety, without any functions or closures with hidden structure. Here, such values are allocated and distributed into memory cells, but nevertheless observable in their entirety.

Purely Positive Type	A^+	::=	$\mathbf{1} \mid A_1^+ \otimes A_2^+ \mid A_1^+ \vee A_2^+$
Purely Positive Context	Γ^+	::=	$\cdot \mid \Gamma^+, x : A^+$

We then have the following characterization.

▶ **Corollary 12** (Final Configurations of Purely Positive Type). *All cells in a final configuration $\cdot \vdash \mathcal{F} :: \Gamma^+$ have the form $!\mathsf{cell}\ a\ V$, and therefore all addresses in V are observable and eligible.*

This means if we reconstitute a final configuration into a collection of proofs (one for each $a : A^+$ in Γ^+) by introducing cuts, then all these cuts will be snips.

6 Termination

We prove termination by means of a logical relation (predicate), which we first define for closed configurations (with no free addresses), and then extend to open configurations. The definition and proof incorporate ideas from standard logical relations for natural deduction into those of Pérez et al. [20] in the context of synchronous message-passing concurrency.

▶ **Definition 13.** *We define two predicates on configurations,* $[\![a : A]\!]$ *and* $[a : A]$*, by mutual induction on the structure of the type* A*.*

1. $\mathcal{C} \in [\![a : A]\!]$ *iff* $\mathcal{C} \longrightarrow^* \mathcal{F}$ *where* $\mathcal{F}$ *is final and* $\mathcal{F} \in [a : A]$*.*

2.a $\mathcal{F} \in [a : B \otimes C]$ *iff* $\mathcal{F} = \mathcal{F}', !\mathsf{cell}\ a\ \langle b, c\rangle$*,* $\mathcal{F}' \in [b : B]$*, and* $\mathcal{F}' \in [c : C]$*.*
2.b $\mathcal{F} \in [a : \mathbf{1}]$ *iff* $\mathcal{F} = \mathcal{F}', !\mathsf{cell}\ a\ \langle\rangle$*.*
2.c $\mathcal{F} \in [a : B \supset C]$ *iff for all* $\mathcal{F}'$ *such that* $\mathcal{F}, \mathcal{F}' \in [b : B]$*,*
we have $\mathcal{F}, \mathcal{F}', (\mathsf{proc}\ c\ (a.\langle b, c\rangle)), \mathsf{cell}\ c\ \_) \in [\![c : C]\!]$*.*
2.d $\mathcal{F} \in [a : B \,\&\, C]$ *iff both* $\mathcal{F}, (\mathsf{proc}\ b\ (a.\pi_1(b)), \mathsf{cell}\ b\ \_) \in [\![b : B]\!]$
and $\mathcal{F}, (\mathsf{proc}\ c\ (a.\pi_2(c)), \mathsf{cell}\ c\ \_) \in [\![c : C]\!]$*.*
2.e $\mathcal{F} \in [a : B \vee C]$ *iff either* $\mathcal{F} = \mathcal{F}_B, !\mathsf{cell}\ a\ (\pi_1(b))$ *with* $\mathcal{F}_B \in [b : B]$
or $\mathcal{F} = \mathcal{F}_C, !\mathsf{cell}\ a\ (\pi_2(c))$ *with* $\mathcal{F}_C \in [c : C]$*.*

We can then extend this definition to specify the behavior of a configuration providing more than one address.

▶ **Definition 14.** *We define* $\mathcal{C} \in [\![\Gamma]\!]$ *iff for all* $a : A \in \Gamma$*,* $\mathcal{C} \in [\![a : A]\!]$*.*

Finally, using the above definitions, we can extend the predicate to open configurations.

▶ **Definition 15.** *We define* $\mathcal{C} \in [\![\Gamma \vdash a : A]\!]$ *iff for all* $\mathcal{C}' \in [\![\Gamma]\!]$*,* $\mathcal{C}', \mathcal{C} \in [\![a : A]\!]$*. Note that* $\mathcal{C} \in [\![\cdot \vdash a : A]\!]$ *iff* $\mathcal{C} \in [\![a : A]\!]$*.*

Given these definitions, we have three key lemmas for the proof of termination.

▶ **Lemma 16** (Weakening)**.** *If* $\mathcal{C} \in [\![a : A]\!]$*, then for all* b, B*, and* $\mathcal{C}' \in [\![b : B]\!]$ *we have* $\mathcal{C}', \mathcal{C} \in [\![a : A]\!]$*.*

As a corollary, if $\mathcal{C} \in [\![a : A]\!]$ then $\mathcal{C} \in [\![\Gamma \vdash a : A]\!]$ for any Γ.

▶ **Lemma 17** (Closure)**.** *If* $\mathcal{C} \longrightarrow^* \mathcal{C}'$ *then* $\mathcal{C} \in [\![\Gamma \vdash a : A]\!]$ *iff* $\mathcal{C}' \in [\![\Gamma \vdash a : A]\!]$*.*

▶ **Lemma 18** (Inversion)**.** *If a final* $\mathcal{F} \in [a : A]$ *then* $\mathcal{F} = \mathcal{F}', !\mathsf{cell}\ a\ W$ *for some* $\mathcal{F}'$ *and* W*.*

Finally, using these lemmas, we can go on to prove the main theorem of this section, which state that all well-typed configurations satisfy the termination predicate. We can apply this to any of the succedents in the general typing judgment for configurations.

▶ **Theorem 19.** *If* $\Gamma \vdash \mathcal{C} :: \Delta$*, then* $\mathcal{C} \in [\![\Gamma \vdash a : A]\!]$ *for every* $(a : A) \in \Delta$*.*

Proof. This proof proceeds by induction on the multiset $\mathcal{M}$ of derivations $\Gamma' \vdash P :: (b : B)$ and $\Gamma' \vdash W : B$ used in the derivation of $\Gamma \vdash \mathcal{C} :: \Delta$. Derivations are ordered in the standard way, and we use the multiset ordering derived from this as the basis of our induction. As noted in the proof of Theorem 9, the typing derivation for a configuration induces an order on that configuration and we can examine it from right to left. ◀

A corollary of this theorem is that any closed well-typed configuration terminates in a final configuration.

7 Conclusion

We defined Sax, a new hybrid form of sequent calculus in which right rules for positive connectives and left rules for negative connectives are replaced by axioms, that is, inference rules with no premises. This calculus satisfies a modified cut elimination theorem in which certain analytic cuts which preserve the subformula property are allowed. We showed how to assign process expression to derivations in Sax and provided a simple shared memory semantics for them: cut allocates memory cells, identity copies contents from one cell to another, processes assigned to right rules write to cells and those assigned to left rules read from them. Cells may be written at most once, but read many times, which means that they provide synchronization points between concurrent processes. This seems quite similar to futures [12], an analogy we have substantiated in an unpublished report [25]. This report does not investigate the proof-theoretic foundations of Sax but further generalizes the type system to *adjoint logic* [26, 18, 23, 24], adds recursive types, and shows that futures can be embedded into an adjoint formulation of Sax. We further conjecture that functional programs can be compiled directly into Sax, where a particular schedule for the resulting concurrent programs corresponds to their sequential execution. This has been developed (without proof) in some unpublished lecture notes [22, Lectures 19–21].

References

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

2 Zena M. Ariola, Aaron Bohannon, and Amr Sabry. Sequent calculi and abstract machines. *ACM Transactions on Programming Languages and Systems*, 31(4):13:1–13:48, May 2009.

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

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

5 Iliano Cervesato, Frank Pfenning, David Walker, and Kevin Watkins. A concurrent logical framework II: Examples and applications. Technical Report CMU-CS-02-102, Department of Computer Science, Carnegie Mellon University, 2002. Revised May 2003.

6 Iliano Cervesato and Andre Scedrov. Relating state-based and process-based concurrency through linear logic. *Information and Computation*, 207(10):1044–1077, October 2009.

7 Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In *International Conference on Functional Programming (ICFP 2000)*, pages 233–243. ACM Press, September 2000.

8 H. B. Curry. Functionality in combinatory logic. *Proceedings of the National Academy of Sciences, U.S.A.*, 20:584–590, 1934.

9 H. B. Curry and R. Feys. *Combinatory Logic*. North-Holland, Amsterdam, 1958.

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

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

12 Robert H. Halstead. Multilisp: A language for parallel symbolic computation. *ACM Transactions on Programming Languages and Systems*, 7(4):501–539, October 1985.

13 Hugo Herbelin. A lambda-calculus structure isomorphic to Gentzen-style sequent calculus structure. In L. Pacholski and J. Tiuryn, editors, *8th International Workshop on Computer Science Logic*, pages 61–75, Kazimierz, Poland, September 1994. Springer LNCS 933.

14 W. A. Howard. The formulae-as-types notion of construction. Unpublished note. An annotated version appeared in: *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, 479–490, Academic Press (1980), 1969.

15 Stephen Cole Kleene. *Introduction to Metamathematics*. North-Holland, 1952.

16 Paul Blain Levy. Call-by-push-value: Decomposing call-by-value and call-by-name. *Higher-Order and Symbolic Computation*, 19(4):377–414, 2006.

17 Chuck Liang and Dale Miller. Focusing and polarization in linear, intuitionistic, and classical logics. *Theoretical Computer Science*, 410(46):4747–4768, November 2009.

18 Daniel R. Licata and Michael Shulman. Adjoint logic with a 2-category of modes. In *International Symposium on Logical Foundations of Computer Science (LFCS)*, pages 219–235. Springer LNCS 9537, January 2016.

19 Per Martin-Löf. On the meanings of the logical constants and the justifications of the logical laws. Notes for three lectures given in Siena, Italy. Published in *Nordic Journal of Philosophical Logic*, 1(1):11-60, 1996, April 1983.

20 Jorge A. Pérez, Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logical relations and observational equivalences for session-based concurrency. *Information and Computation*, 239:254–302, 2014.

21 Frank Pfenning. Structural cut elimination I. Intuitionistic and classical logic. *Information and Computation*, 157(1/2):84–141, March 2000.

22 Frank Pfenning. Types and programming languages. Lecture Notes, 2019. URL: `http://www.cs.cmu.edu/~fp/courses/15814-f19`.

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

24 Klaas Pruiksma and Frank Pfenning. A message-passing interpretation of adjoint logic. In F. Martins and D. Orchard, editors, *Workshop on Programming Language Approaches to Concurrency and Communication-Centric Software (PLACES)*, pages 60–79, Prague, Czech Republic, April 2019. EPTCS 291.

25 Klaas Pruiksma and Frank Pfenning. Back to futures. *CoRR*, abs/2002.04607, February 2020. URL: `http://arxiv.org/abs/2002.04607`.

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

27 Robert J. Simmons. *Substructural Logical Specifications*. PhD thesis, Carnegie Mellon University, November 2012. Available as Technical Report CMU-CS-12-142.

28 Gerald Jay Sussman and Guy L. Steele. Scheme: An intepreter for extended lambda calculus. *Higher-Order and Symbolic Computation*, 11(4):405–439, December 1998. Reprint of the MIT AI Memo 349:19, December 1975.

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

A Some Cases in the Proof of Cut Admissibility

Proof of Theorem 5. We consider various classes of cases. These cases are not mutually exclusive, which means that the algorithm for the construction of $\mathcal{F}$ from $\mathcal{D}$ and $\mathcal{E}$ induced by this constructive proof is naturally nondeterministic. We restrict ourselves to implication only, but show all the cases relevant to this fragment. The cases for other connectives follow similar patterns.

The first two cases show the admissibility of cut by building a snip when one (or both) of the cut formulas are eligible.

Case: $*/\_$ (which also covers the cases $\supset L^0/\_$)

$$\dfrac{\dfrac{\mathcal{D}}{\Gamma \vdash x^* : A} \quad \dfrac{\mathcal{E}}{\Gamma, x : A \vdash z : C}}{\Gamma \vdash z : C}\ \mathsf{Cut}_A \quad \Longrightarrow \quad \dfrac{\dfrac{\mathcal{D}}{\Gamma \vdash x^* : A} \quad \dfrac{\mathcal{E}}{\Gamma, x : A \vdash z : C}}{\Gamma \vdash z : C}\ \mathsf{Snip}^1_A$$

Case: $\_/*$ (which also covers a subcase of $\_/\supset^0$)

$$\dfrac{\dfrac{\mathcal{D}}{\Gamma \vdash x : A} \quad \dfrac{\mathcal{E}}{\Gamma, x^* : A \vdash z : C}}{\Gamma \vdash z : C}\ \mathsf{Cut}_A \quad \Longrightarrow \quad \dfrac{\dfrac{\mathcal{D}}{\Gamma \vdash x : A} \quad \dfrac{\mathcal{E}}{\Gamma, x^* : A \vdash z : C}}{\Gamma \vdash z : C}\ \mathsf{Snip}^2_A$$

The next case covers a cut against an irrelevant antecedent. This case is not strictly necessary and could be replaced by several more specialized ones if desired.

Case: $\_/0$ (also covers $\_/\supset L^0$ and $\_/\mathsf{Id}$ where the cut formula is a side formula in $\mathcal{E}$)

$$\dfrac{\dfrac{\mathcal{D}}{\Gamma \vdash x : A} \quad \dfrac{\mathcal{E}}{\Gamma, x^0 : A \vdash z : C}}{\Gamma \vdash z : C}\ \mathsf{Cut}_A \quad \Longrightarrow \quad \begin{array}{c}\mathcal{E} - \{x^0 : A\} \\ \Gamma \vdash z : C\end{array}$$

In the next two cases the cut formula is the principal formula of an identity, either in $\mathcal{D}$ or $\mathcal{E}$.

Case: $\_/\mathsf{Id}$

$$\dfrac{\dfrac{\mathcal{D}}{\Gamma \vdash x : A} \quad \dfrac{}{\Gamma, x : A \vdash z : A}\ \mathsf{Id}}{\Gamma \vdash z : A}\ \mathsf{Cut}_A \quad \Longrightarrow \quad \begin{array}{c}[z/x]\mathcal{D} \\ \Gamma \vdash z : A\end{array}$$

Next is a case where $z : C$ may be eligible in $\mathcal{F}$ even if it is not in $\mathcal{E}$. Our proof is not concerned with variables that may gain eligibility.

Case: $\mathsf{Id}/\_$

$$\dfrac{\dfrac{}{\Gamma', y : A \vdash x : A}\ \mathsf{id} \quad \dfrac{\mathcal{E}}{\Gamma', y : A, x : A \vdash z : C}}{\Gamma', y : A \vdash z : C}\ \mathsf{Cut}_A \quad \Longrightarrow \quad \begin{array}{c}[y/x]\mathcal{E} \\ \Gamma', y : A \vdash z : C\end{array}$$

The next case is the principal case where the cut formula is inferred in the last inference of both premises of the cut.

Case: $\supset R/\supset L^0$

$$\dfrac{\dfrac{\dfrac{\mathcal{D}_0}{\Gamma', y : A_1, x_1 : A_1 \vdash x_2 : A_2}}{\Gamma', y : A_1 \vdash x : A_1 \supset A_2}\ \supset R \quad \dfrac{}{\Gamma', y^* : A_1, x : A_1 \supset A_2 \vdash z^* : A_2}\ \supset L^0}{\Gamma', y : A_1 \vdash z : A_2}\ \mathsf{Cut}_{A_1 \supset A_2} \Longrightarrow \begin{array}{c}[y/x_1, z/x_2]\mathcal{D}_0 \\ \Gamma', y : A_1 \vdash z : A_2\end{array}$$

Note that in this case $y : A_1$ and $z : A_2$ may lose eligibility. However, $A_1 < A_1 \supset A_2$ and $A_2 < A_1 \supset A_2$ so our requirements are satisfied in case they remain relevant.

In the next group of cases, the first derivation $\mathcal{D}$ is arbitrary and the cut formula is a side formula of the last inference in $\mathcal{E}$. Because $\mathcal{E}$ is assumed to be cut-free, we get five cases $\_/\supset R$, $\_/\mathsf{Snip}^1$, $\_/\mathsf{Snip}^2$, $\_/\supset L^0$, and $\_/\mathsf{Id}$, where the last two are already covered by $\_/0$.

Case: $\_/\supset R$

$$\dfrac{\dfrac{\mathcal{D}}{\Gamma \vdash x : A} \quad \dfrac{\dfrac{\mathcal{E}_0}{\Gamma, x : A, z_1 : C_1 \vdash z_2 : C_2}}{\Gamma, x : A \vdash z : C_1 \supset C_2}\ \supset R}{\Gamma \vdash z : C_1 \supset C_2}\ \mathsf{Cut}_A \Longrightarrow \dfrac{\dfrac{\mathcal{D} + \{z_1^0 : C_1\}}{\Gamma, z_1^0 : C_1 \vdash x : A} \quad \dfrac{\mathcal{E}_0}{\Gamma, x : A, z_1 : C_1 \vdash z_2 : C_2}\ \mathsf{Cut}_A}{\dfrac{\Gamma, z_1 : C_1 \vdash z_2 : C_2}{\Gamma \vdash z : C_1 \supset C_2}\ \supset R}$$

Case: $\_/\mathsf{Snip}^1$ with two subcases.

$$\dfrac{\mathcal{D}\ \Gamma \vdash x : A \qquad \dfrac{\mathcal{E}_1\ \Gamma, x : A \vdash y^* : B \quad \mathcal{E}_2\ \Gamma, x : A, y : B \vdash z : C}{\Gamma, x : A \vdash z : C}\ \mathsf{Snip}^1_B}{\Gamma \vdash z : C}\ \mathsf{Cut}_A \quad \Longrightarrow \quad \mathcal{F}\ \Gamma \vdash z : C$$

where $\mathcal{F}$ is defined in each subcase below from

$$\dfrac{\mathcal{D}\ \Gamma \vdash x : A \quad \mathcal{E}_1\ \Gamma, x : A \vdash y^* : B}{\Gamma \vdash y : B}\ \mathsf{Cut}_A \quad \Longrightarrow \quad \mathcal{F}_1\ \Gamma \vdash y : B$$

and

$$\dfrac{\mathcal{D} + \{y : B\}\ \Gamma, y : B \vdash x : A \quad \mathcal{E}_2\ \Gamma, y : B, x : A \vdash z : C}{\Gamma, y : B \vdash z : C}\ \mathsf{Cut}_A \quad \Longrightarrow \quad \mathcal{F}_2\ \Gamma, y : B \vdash z : C$$

There are two subcases depending on properties of $\mathcal{F}_1$, obtained from the induction hypothesis.

Subcase: y is still eligible in $\mathcal{F}_1$.

Subcase: y is not eligible in $\mathcal{F}_1$ and $B < A$. Note that in this case we can apply the induction hypothesis once more.

These two subcases yield the following two definitions of $\mathcal{F}$, respectively:

$$\mathcal{F} = \dfrac{\mathcal{F}_1\ \Gamma \vdash y^* : B \quad \mathcal{F}_2\ \Gamma, y : B \vdash z : C}{\Gamma \vdash z : C}\ \mathsf{Snip}^1 \qquad \mathcal{F} = \dfrac{\mathcal{F}_1\ \Gamma \vdash y : B \quad \mathcal{F}_2\ \Gamma, y : B \vdash z : C}{\Gamma \vdash z : C}\ \mathsf{Cut}_B$$

Case: $\_/\mathsf{Snip}^2$ with three subcases.

$$\dfrac{\mathcal{D}\ \Gamma \vdash x : A \qquad \dfrac{\mathcal{E}_1\ \Gamma, x : A \vdash y : B \quad \mathcal{E}_2\ \Gamma, x : A, y^* : B \vdash z : C}{\Gamma, x : A \vdash z : C}\ \mathsf{Snip}^1_B}{\Gamma \vdash z : C}\ \mathsf{Cut}_A \quad \Longrightarrow \quad \mathcal{F}$$

We first apply the induction hypothesis twice on smaller derivations.

$$\dfrac{\mathcal{D}\ \Gamma \vdash x : A \quad \mathcal{E}_1\ \Gamma, x : A \vdash y : B}{\Gamma \vdash y : B}\ \mathsf{Cut}_A \quad \Longrightarrow \quad \mathcal{F}_1\ \Gamma \vdash y : B$$

and

$$\dfrac{\mathcal{D} + \{y^0 : B\}\ \Gamma, y^0 : B \vdash x : A \quad \mathcal{E}_2\ \Gamma, y^* : B, x : A \vdash z : C}{\Gamma, y : B \vdash z : C}\ \mathsf{Cut}_A \quad \Longrightarrow \quad \mathcal{F}_2\ \Gamma, y : B \vdash z : C$$

There are three subcases for the status of $y : B$ in $\mathcal{F}_2$.

Subcase: $y : B$ becomes irrelevant.

Subcase: $y : B$ remains eligible.

Subcase: $y : B$ is ineligible but relevant and $B < A$.

These subcases yield the following three definitions for $\mathcal{F}$, respectively:

$$\mathcal{F} = \dfrac{\mathcal{F}_2 - \{y^0 : B\}}{\Gamma \vdash z : C} \quad \mathcal{F} = \dfrac{\dfrac{\mathcal{F}_1}{\Gamma \vdash y : B} \quad \dfrac{\mathcal{F}_2}{\Gamma, y^* : B \vdash z : C}}{\Gamma \vdash z : C}\ \mathsf{Snip}^2_B \quad \mathcal{F} = \dfrac{\dfrac{\mathcal{F}_1}{\Gamma \vdash y : B} \quad \dfrac{\mathcal{F}_2}{\Gamma, y : B \vdash z : C}}{\Gamma \vdash z : C}\ \mathsf{Cut}_B$$

The next set of cases the cut formula is a side formula of the inference in $\mathcal{D}$ and $\mathcal{E}$ is arbitrary.

Case: $\mathsf{Snip}^1/\_$.

$$\dfrac{\dfrac{\mathcal{D}_1}{\Gamma \vdash y^* : B} \quad \dfrac{\mathcal{D}_2}{\Gamma, y : B \vdash x : A}}{\Gamma \vdash x : A}\ \mathsf{Snip}^1_B \quad \dfrac{\mathcal{E}}{\Gamma, x : A \vdash z : C}}{\Gamma \vdash z : C}\ \mathsf{Cut}_A$$

$$\Longrightarrow \quad \dfrac{\dfrac{\mathcal{D}_1}{\Gamma \vdash y^* : B} \quad \dfrac{\dfrac{\mathcal{D}_2}{\Gamma, y : B \vdash x : A} \quad \dfrac{\mathcal{E} + \{y^0 : B\}}{\Gamma, y^0 : B, x : A \vdash z : C}}{\Gamma, y : B \vdash z : C}\ \mathsf{Cut}_A}{\Gamma \vdash z : C}\ \mathsf{Snip}^1_B$$

Case: $\mathsf{Snip}^2/\_$

$$\dfrac{\dfrac{\mathcal{D}_1}{\Gamma \vdash y : B} \quad \dfrac{\mathcal{D}_2}{\Gamma, y^* : B \vdash x : A}}{\Gamma \vdash x : A}\ \mathsf{Snip}^2_B \quad \dfrac{\mathcal{E}}{\Gamma, x : A \vdash z : C}}{\Gamma \vdash z : C}\ \mathsf{Cut}_A \quad \Longrightarrow \quad \dfrac{\mathcal{F}}{\Gamma \vdash z : C}$$

where

$$\mathcal{F}_2 = \dfrac{\dfrac{\mathcal{D}_2}{\Gamma, y^* : B \vdash x : A} \quad \dfrac{\mathcal{E} + \{y^0 : B\}}{\Gamma, y^0 : B, x : A \vdash z : C}}{\Gamma, y : B \vdash z : C}\ \mathsf{Cut}_A$$

Again, there are three subcases.

Subcase: $y^0 : B$ is irrelevant in $\mathcal{F}_2$.

Subcase: $y^* : B$ is eligible in $\mathcal{F}_2$.

Subcase: $y : B$ is relevant but not eligible in $\mathcal{F}_2$ and $B < A$.

These three subcases yield the following three definitions for $\mathcal{F}$, respectively:

$$\mathcal{F} = \dfrac{\mathcal{F}_2 - \{y^0 : B\}}{\Gamma \vdash z : C} \quad \mathcal{F} = \dfrac{\dfrac{\mathcal{D}_1}{\Gamma \vdash y : B} \quad \dfrac{\mathcal{F}_2}{\Gamma, y^* : B \vdash z : C}}{\Gamma \vdash z : C}\ \mathsf{Snip}^2_B \quad \mathcal{F} = \dfrac{\dfrac{\mathcal{D}_1}{\Gamma \vdash y : B} \quad \dfrac{\mathcal{F}_2}{\Gamma, y : B \vdash z : C}}{\Gamma \vdash z : C}\ \mathsf{Cut}_B$$

◀

B Some Example Programs

The following are the process expressions assigned to the proofs in Section 3 and Section 4.

$$x : A \supset B, y : B \supset C \vdash z : A \supset C$$

$$\begin{array}{l}\textbf{case}\ z^W\ (\langle u, w\rangle \Rightarrow b \leftarrow x^R.\langle u, b\rangle\ ; \\ \qquad\qquad\qquad\quad y^R.\langle b, w\rangle)\end{array}$$

This process terminates immediately after writing a continuation to z. When this continuation is called with an argument u and destination w, it allocates a fresh cell b and passes u to x with instructions to place the result in b. While this continuation executes, it passes the address b (which may not be filled yet) to y and jumps to *that* continuation in order to write *its* answer into w. Note that two processes may execute concurrently here: intuitively, one executing the function x and the other executing the function y. They are connected via a common reference to a fresh cell b, to be written by x and read by y.

$$x : A \mathbin{\&} B \vdash w : A \otimes B$$

$$\begin{array}{l} y \leftarrow x^R.\pi_1(y)\ ; \\ z \leftarrow x^R.\pi_2(z)\ ; \\ w^W.\langle y, z\rangle \end{array}$$

This process reads the continuation at x twice, requesting that the first component of the pair be written to a freshly allocated cell y, the second to a freshly allocated cell z. While these processes execute, it writes the pair $\langle y, z\rangle$ to the required destination w and terminates. Note that both y and z have been allocated, but neither needs to have been filled by the time this process terminates, since the two processes executing with the destinations y and z can continue to run.

$$x : A \otimes B \vdash w : A \mathbin{\&} B$$

$$\begin{array}{l} \textbf{case}\ x^R\ (\langle y, z\rangle \Rightarrow \\ \qquad \textbf{case}\ w^W\ (\ \pi_1(u) \Rightarrow u^W \leftarrow y^R \\ \qquad\qquad \mid \pi_2(v) \Rightarrow v^W \leftarrow z^R\ )) \end{array}$$

This process reads the pair $\langle y, z\rangle$ from x and then terminates by writing a continuation to w. If this continuation is invoked by a reader requesting the first component of $A \mathbin{\&} B$ to be put into u, this is satisfied by copying the contents of y; if the second component is requested we copy the contents of z. This program is essentially sequential.

$$p : (A \supset C) \mathbin{\&} (B \supset C) \vdash q : (A \vee B) \supset C$$

$$\begin{array}{l} \textbf{case}\ q^W\ (\langle s, z\rangle \Rightarrow \\ \qquad \textbf{case}\ s^R\ (\ \pi_1(x) \Rightarrow r \leftarrow p^R.\pi_1(r)\ ;\ r^R.\langle x, z\rangle \\ \qquad\qquad \mid \pi_2(y) \Rightarrow t \leftarrow p^R.\pi_2(t)\ ;\ t^R.\langle y, z\rangle\ )) \end{array}$$

This process writes a continuation to q and terminates. This continuation represents a function with argument s and destination z. If it is invoked, it distinguishes two cases for the contents of s. If it is $\pi_1(x)$ for some address x, it obtains the first component of p (call it r) and invokes that with x, requesting the result to be put into z. If it is $\pi_2(y)$, it takes a symmetric action with the second component of p. This example has little parallelism: even though a fresh process is spawned to fill r and t in the two branches of the inner case, they are immediately read which will block until r and t have been filled, respectively.

C More Details in the Termination Proof

Proof Sketch of Theorem 19. The case of an empty configuration is straightforward, as is the case where the rightmost object in $\mathcal{C}$ provides an address $d \neq a$. In all remaining cases, we may therefore assume that $\mathcal{C}$ is non-empty, and that its rightmost part (either a cell !cell $a\ W$ or a process with its unfilled destination cell at a) provides a. We refer to this component as ϕ_a and the remainder of $\mathcal{C}$ as $\mathcal{C}'$.

If the typing derivation for ϕ_a ends with an id rule, then $\phi_a = (\mathsf{proc}\ a\ a \leftarrow b, \mathsf{cell}\ a\ \_)$ for some $b : B$ in Δ. We therefore have (by the inductive hypothesis) that $\mathcal{C}' \in [\![\Gamma \vdash b : A]\!]$. Let $\mathcal{C}'' \in [\![\Gamma]\!]$. Now, by definition, $(\mathcal{C}'', \mathcal{C}') \in [\![b : A]\!]$, and so $(\mathcal{C}'', \mathcal{C}') \longrightarrow^* \mathcal{F}$ for some $\mathcal{F}$ final and $\mathcal{F} \in [b : A]$. Lemma 18 gives us that $\mathcal{F}$ contains $!\mathsf{cell}\ b\ W$ for some W. We therefore have that $(\mathcal{C}'', \mathcal{C}) \longrightarrow^* \mathcal{F}, !\mathsf{cell}\ a\ W$. Now, split $\mathcal{C}'$ into $\mathcal{C}_1, \mathcal{C}_2$, where $\mathcal{C}_2$ contains exactly the objects which mention or provide b. We may now apply the inductive hypothesis to $\mathcal{C}_1, !\mathsf{cell}\ a\ W$ to get that $\mathcal{C}_1, !\mathsf{cell}\ a\ W \in [\![\Gamma \vdash a : A]\!]$. Lemma 16 then allows us to conclude that $\mathcal{C}', !\mathsf{cell}\ a\ W \in [\![\Gamma \vdash a : A]\!]$. Thus, $\mathcal{C}'', \mathcal{C}', !\mathsf{cell}\ a\ W \longrightarrow^* \mathcal{F}', !\mathsf{cell}\ a\ W$ and $\mathcal{F}', !\mathsf{cell}\ a\ W \in [a : A]$. Confluence (an easy consequence of Theorem 10) gives us that $\mathcal{F} = \mathcal{F}'$, and Lemma 17 completes this case.

If the typing derivation for ϕ_a ends with a cut rule, we may simply apply to the inductive hypothesis after taking a step.

If the typing derivation for ϕ_a ends with $\supset L$, $\& L$, $\mathbf{1}R$, $\otimes R$, or $\vee R$, then we simply take either zero or one steps (depending on exactly which rule it is and what form $\mathcal{C}$ has), invoke the inductive hypothesis up to two times on $\mathcal{C}'$, and then conclude using Lemma 17.

If the typing derivation for ϕ_a ends with $\mathbf{1}L$, $\otimes L$, or $\vee L$, we proceed much as in the case of id. We set up a configuration $\widehat{\mathcal{C}}$ which is our candidate for what $\mathcal{C}$ would look like if we were able to step in ϕ_a, and using the inductive hypothesis both on $\widehat{\mathcal{C}}$ and on $\mathcal{C}'$, we are able to show that given $\mathcal{C}'' \in [\![\Gamma]\!]$, both $(\mathcal{C}'', \mathcal{C})$ and $(\mathcal{C}'', \widehat{\mathcal{C}})$ reduce to the same configuration $\widetilde{\mathcal{C}}$ (taking advantage of confluence and Lemma 18 to do so). As in the identity case, Lemma 17 completes these cases.

If the typing derivation for ϕ_a ends with $\supset R$ or $\& R$, we proceed largely similarly, constructing a candidate $\widehat{\mathcal{C}}$ for the result of stepping ϕ_a in $\mathcal{C}$. The only difference is that here, we need to augment $\mathcal{C}$ with an additional process which reads from the cell a, rather than working with $\mathcal{C}$ on its own. ◀