Compiling adjoint natural deduction to the semi-axiomatic sequent calculus

JOANNA BOYLAND, Carnegie Mellon University, USA
FRANK PFENNING, Carnegie Mellon University, USA

1 INTRODUCTION

The semi-axiomatic sequent calculus (Sax) [DeYoung et al., 2020], a modification of the sequent calculus where the noninvertible rules are turned into axioms, was introduced as a logical system that had a natural computational interpretation for shared memory (futures) [Pruiksma and Pfenning, 2022] and message-passing [Pfenning and Pruiksma, 2023]. Sax satisfies only a weaker version of cut-elimination, where certain kinds of cuts, called snips, are preserved because their inclusion does not break the subformula property. Somayyajula [2024] provides an alternative formulation for Sax via a bidirectional typechecker.

In this work-in-progress we generalize from separate linear and nonlinear versions of Sax [Pfenning and Pruiksma, 2023] to an adjoint formulation that also integrates these and other modes like affine or strict. Unlike the adjoint Sax presented in unpublished lecture notes [Pfenning, 2025], the typing rules presented here are bidirectional. Consequently, our adjoint Sax with only snips will automatically satisfy the subformula property.

We further provide a compilation from (bidirectional) adjoint natural deduction [Jang et al., 2024] to bidirectional adjoint Sax, which amounts to a translation to destination-passing style employing metalevel continuations. We prove that bidirectional typing is preserved by the compilation. This translation is at the heart of a prototype compiler for adjoint functional programs currently under development.

2 BACKGROUND

After linear logic was introduced by Girard [1987], where the structural rules of weakening and contraction are controlled by the exponential modality, other substructural logics arose, such as Benton's linear/non-linear system [Benton, 1994]. LNL has linear and non-linear formulae, and linear and non-linear contexts. Adjoint logic [Pruiksma et al., 2018] goes beyond this: instead of just two possible modes, linear and non-linear, there can be an arbitrary pre-order $\leq$ of modes m, each with its own set of allowed structural properties $\sigma(m)$. The principle of independence requires that in a sequent $\Gamma \vdash A_r$ the mode m of each formula in Γ satisfies $m \geq r$. As in LNL, adjoint logic decomposes the exponential modality into two explicit shifts between modes: the downshift $\downarrow_m^k A_k$ for $k \geq m$ and the upshift $\uparrow_n^m A_n$ for $m \geq n$. Modes m can admit or not admit weakening and admit or not admit contraction. We use $\mathsf{W} \in \sigma(m)$ to mean weakening is allowed at mode m and likewise $\mathsf{C} \in \sigma(m)$ for contraction. Adjoint logic was first formulated in the sequent calculus, and then extended to natural deduction in Jang et al. [2024]. Since adjoint logic is intuitionistic, we'll freely go back and forth between propositions and types.

We present now the collection of valid types we use for both natural deduction (ND) and Sax. For the positive types, we have the eager pair $A_m \otimes B_m$ (both elements in the pair must have the same mode m), the unit type $\mathbf{1}_m$, the (labelled) sum type $\oplus\{\ell : A_m^\ell\}_{\ell \in L}$ with labels ℓ corresponding to the type A^ℓ (all options must have the same mode m, and L must be nonempty and finite), and the downshift $\downarrow_m^k A_k$. The downshift to m of some type of mode k is a type of mode m, assuming $k \geq m$.

Authors' addresses: Joanna Boyland, joannabo@andrew.cmu.edu, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA; Frank Pfenning, fp@cs.cmu.edu, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA.

For the negative types, we have the function type $A_m \multimap B_m$, the lazy record type $\&\{\ell : A^\ell_m\}_{\ell \in L}$ (again with the requirement that L is finite and nonempty), and the upshift $\uparrow^m_n A_n$ where $m \geq n$, which is a dual notion to the downshift. The zero type is not a valid type here.

$$\begin{aligned} A_m, B_m ::= & A_m \otimes B_m \mid \mathbf{1}_m \mid \oplus\{\ell : A^\ell_m\}_{\ell \in L} \mid \downarrow^k_m A_k && (k \geq m) \\ & \mid A_m \multimap B_m \mid \&\{\ell : A^\ell_m\}_{\ell \in L} \mid \uparrow^m_n A_n && (m \geq n) \end{aligned}$$

As an exercise, the reader can use the $\otimes$ introduction and elimination rules and variable rule [Jang et al., 2024] to show that the following adjoint ND expression swaps the elements of a term of type $A_m \otimes B_m$.

$$p : A_m \otimes B_m \vdash \text{match } p \text{ with } (x, y) \hookrightarrow (y, x) \Leftarrow B_m \otimes A_m.$$

Natural deduction and the sequent calculus are known to prove the same theorems. A third proof system can be added to the mix: the semi-axiomatic sequent calculus, a modification of the sequent calculus where the non-invertible rules become axioms. A natural computational model of Sax is an imperative language whose processes write to destination addresses that will be read by other processes [DeYoung et al., 2020].

Lecture notes [Pfenning, 2025] provide an adjoint type system for Sax, but it is not bidirectional in the sense explained in detail by Dunfield and Krishnaswami [2022]. Cuts do not have to be snips, so the system does not satisfy the subformula property. On the other hand, Somayyajula [2024] gives a bidirectional type system, but only for structural Sax. Bidirectional typing for natural deduction applies only to the succedent—we assume the types for all variables in scope are known (written $x \Rightarrow A$). In Sax, not only can the succedent either synthesize or be checked against, but each antecedent can also either synthesize or be checked against. In the computational model of typechecking, the latter means the variables in the context either have an a priori known type, or their type can be synthesized during the typechecking process. Instead of the standard cut, we have snips, which synthesize the type of the allocated address either in the first or the second premise. On the right of the turnstile, $z \Leftarrow A$ will be used for checking destination z against a known type A, and $z \Rightarrow A$ for algorithmically synthesizing A for z. On the left, $x \Rightarrow A$ will be used to denote that x has known type A, while $x \Leftarrow A$ stands for a type A we have to algorithmically synthesize. The symbol $\div$ separates the process and the address it writes to.

$$\frac{\Gamma \vdash P(x) \div (x \Leftarrow A) \qquad \Gamma, x \Leftarrow A \vdash Q(x) \div d \Leftarrow D}{\Gamma \vdash x \leftarrow_R P(x)\,;\, Q(x) \div d \Leftarrow D}\ \text{snipR}$$

$$\frac{\Gamma \vdash P(x) \div (x \Rightarrow A) \qquad \Gamma, x \Rightarrow A \vdash Q(x) \div d \Leftarrow D}{\Gamma \vdash x \leftarrow_L P(x)\,;\, Q(x) \div d \Leftarrow D}\ \text{snipL}$$

Sax with just these snips (and not the general cut) *does* satisfy the subformula property. However, the rules in Somayyajula [2024] are not deterministic, that is, given some Sax expression and a type to synthesize or check against, it is not deterministic which rule we must apply. In this paper we'll be providing rules that are directly implementable for a language that is adjoint (rather than just structural).

3 BIDIRECTIONAL SAX

We will be focusing on adjoint Sax, so we'll omit the typing rules for adjoint ND, which can be found in Jang et al. [2024]. We'll use e to denote an arbitrary expression of ND and s to denote a synthesizing expression, that is,

$$s ::= x \mid s\ e \mid s\,.\,k \mid s\,.\,\text{force} \mid (e : A)$$

We'll use P, Q to denote processes using the read/write vocabulary of (shared) memory. Because we are not interested in the dynamics in this paper, we do not formally distinguish between variables that stand for addresses, and addresses as runtime objects. We tend to use a, b, c, d for variables free in a sequent, and x, y, z, w for variables bound in an expression like a process. We further use t for metavariables, A, B, C, D for types, and k, m, n, r for modes. Our contexts Γ, Ξ will be formed as

$$\Gamma ::= \cdot \mid \Gamma, (x \Leftarrow A_m) \mid \Gamma, (x \Rightarrow A_m)$$

For our bidirectional typing rules for adjoint Sax, we use the "additive" system, where the context in which we type a process will be the collection of all variables in scope, which will generally be denoted Γ. Then, the typing judgment returns the subset Ξ of Γ of the variables used in the formation of the typing of that process.

The judgment

$$\Gamma \vdash P \div (c \Leftarrow C_r) \ / \ \Xi$$

means that in the context Γ of all variables in scope, the process P will write to the destination c and will output the context Ξ of variables used in the typing, and if we check c against C_r, it will succeed. We will also have the synthesis judgment, which says that c will synthesize C_r in the used context Ξ:

$$\Gamma \vdash P \div (c \Rightarrow C_r) \ / \ \Xi.$$

Each right rule for a positive type in the sequent calculus becomes an axiom of Sax. We provide the Sax axiom rule for the eager product below. The used context returned will be $(a \Leftarrow A_m, b \Leftarrow B_m)$ if $a \neq b$ and $(a \Leftarrow A_m)$ if they are the same and m admits contraction. Otherwise, the judgment fails. We denote this by $(a \Leftarrow A_m; b \Leftarrow B_m)$.

$$\frac{a \Leftarrow A_m, b \Leftarrow B_m \in \Gamma}{\Gamma \vdash \text{write } c\,(a, b) \div (c \Leftarrow A_m \otimes B_m) \ / \ (a \Leftarrow A_m; b \Leftarrow B_m)}\ \otimes \text{X}$$

Note that algorithmically we would not already have A_m and B_m in Γ for a and b, but would synthesize them from $A_m \otimes B_m$ in the succedent.

We also provide the left (invertible) rule for the eager product. We'll use γ_r to mean some $d \Leftarrow D_r$ for mode r. By the premise, we have $\Xi \geq r$, but we'll also need $m \geq r$ for the rule to satisfy the principle of independence.

$$\frac{c \Rightarrow A_m \otimes B_m \in \Gamma \qquad \Gamma, x \Rightarrow A_m, y \Rightarrow B_m \vdash P(x, y) \div \gamma_r \ / \ \Xi \qquad m \geq r}{\Gamma \vdash \text{read } c\,((x, y) \hookrightarrow P(x, y)) \div \gamma_r \ / \ (\Xi \setminus x_m \setminus y_m; c \Rightarrow A_m \otimes B_m)}\ \otimes \text{L}$$

We used two context operators above, join (;) and removal (\), defined by Jang et al. [2024]. The join operation was described by example in the $\otimes X$ rule, and allows $x \Rightarrow A_m \ ; \ x \Rightarrow A_m$ only if mode m admits contraction (similarly for $\Leftarrow$). The removal operation $\Xi \setminus x_m$ removes x of mode m (we annotate the mode by writing x_m) from Ξ, where it must occur unless mode m admits weakening.

The following Sax process does not typecheck, because write $d\,(x, y)$ synthesizes the types of x and y, when we already have them in the context as $x \Rightarrow A_m$ and $y \Rightarrow B_m$.

$$p \Leftarrow A_m \otimes B_m \vdash \text{read } p\,((x, y) \hookrightarrow \text{write } d\,(x, y)) \div (d \Leftarrow A_m \otimes B_m) \ / \ p \Leftarrow A_m \otimes B_m$$

It *does* typecheck in Somayyajula [2024], because he allows the left subtyping rule ≤L that unfortunately ruins determinism.

$$\frac{A \leq B \qquad \Gamma, a \Leftarrow B \vdash P \div \gamma}{\Gamma, a \Leftarrow A \vdash P \div \gamma} \leq \mathrm{L}$$

Instead, to keep determinism, we will allow arrow-direction switch only in the id rule. The process id $a\ b$ copies the contents of b to a.

$$\frac{b \Rightarrow B_m \in \Gamma \qquad B_m \leq A_m}{\Gamma \vdash \mathrm{id}\ a\ b \div (a \Leftarrow A_m)\ /\ b \Rightarrow B_m}\ \mathrm{idSub}$$

We next provide the axiom and left rules for the downshift.

$$\frac{a \Leftarrow A_k \in \Gamma}{\Gamma \vdash \mathrm{write}\ c\ \langle a \rangle \div c \Leftarrow \downarrow^k_m A_k\ /\ a \Leftarrow A_k} \downarrow \mathrm{X}$$

$$\frac{c \Rightarrow \downarrow^k_m A_k \in \Gamma \qquad \Gamma, x \Rightarrow A_k \vdash P(x) \div \gamma_r\ /\ \Xi \qquad m \geq r}{\Gamma \vdash \mathrm{read}\ c\ (\langle x \rangle \hookrightarrow P(x)) \div \gamma_r\ /\ (\Xi \setminus x \Rightarrow A_k); c \Rightarrow \downarrow^k_m A_k} \downarrow \mathrm{L}$$

For brevity, the last set of rules we'll include will be the left rule and the right axiom for functions.

$$\frac{\Gamma, x \Rightarrow A_m \vdash P(x, y) \div (y \Leftarrow B_m)\ /\ \Xi}{\Gamma \vdash \mathrm{write}\ c\ ((x, y) \hookrightarrow P(x, y)) \div (c \Leftarrow A_m \multimap B_m)\ /\ (\Xi \setminus x \Rightarrow A_m)} \multimap \mathrm{R}$$

$$\frac{c \Rightarrow A_m \multimap B_m, a \Leftarrow A_m \in \Gamma}{\Gamma \vdash \mathrm{read}\ c\ (a, b) \div (b \Rightarrow B_m)\ /\ (c \Rightarrow A_m \multimap B_m; a \Leftarrow A_m)} \multimap \mathrm{X}$$

4 COMPILATION TO SAX

Now, we sketch compilation of ND expressions into Sax processes. The compilation of a ND expression that is checked against a type is provided a destination, written as $[\![e]\!]d$. In brief, if $e \Leftarrow A_m$ then $[\![e]\!]d \div (d \Leftarrow A_m)$ (see Theorem 4.1). The compilation of a synthesizing ND expression will be defined relative to a meta continuation K, written as $[s]K$. If $s \Rightarrow A_m$ and $x \Rightarrow A_m \vdash K(x) \div (d \Leftarrow C_r)$ then $[s]K \div (d \Leftarrow C_r)$.

For example, the compilation of a variable will be as follows.

$$[x]K := K(x)$$

Next, we consider eager pairs, which inhabit the type $A_m \otimes B_m$.

$$[\![(e_1, e_2)]\!]d := x_1 \leftarrow_R [\![e_1]\!]x_1\ ;\ (x_2 \leftarrow_R [\![e_2]\!]x_2\ ;\ \mathrm{write}\ d\ (x_1, x_2))$$

$$[\![\mathrm{match}\ s\ \mathrm{with}\ (x_1, x_2)\ \hookrightarrow e]\!]d := [s](\underline{\lambda} t.\ (\mathrm{read}\ t\ ((x_1, x_2) \hookrightarrow [\![e]\!]d)))$$

Then, the compilation of the ND expression that swaps the elements of an eager pair $[\![\mathrm{match}\ p\ \mathrm{with}\ (x, y)\ \hookrightarrow (y, x)]\!]d$ will be

$$\mathrm{read}\ p\ ((x, y) \hookrightarrow w \leftarrow_R \mathrm{id}\ w\ x\ ;\ (z \leftarrow_R \mathrm{id}\ z\ y\ ;\ \mathrm{write}\ d\ (w, z)))\ .$$

Next, see the compilations of the ND introduction and elimination forms of downshift and functions.

$$[\![\langle e \rangle]\!]d := x \leftarrow_R [\![e]\!]x\ ;\ \mathrm{write}\ d\ \langle x \rangle$$

$$[\![\mathrm{match}\ s\ \mathrm{with}\ \langle x \rangle\ \hookrightarrow e]\!]d := [s](\underline{\lambda} t.\ \mathrm{read}\ t\ (\langle x \rangle \hookrightarrow [\![e]\!]d))$$

$$[\![\lambda x.\ e]\!]d := \mathrm{write}\ d\ ((x, y) \hookrightarrow [\![e]\!]y)$$

$$[s\ e]K := [s](\underline{\lambda} t.\ x \leftarrow_R [\![e]\!]x\ ;\ (y \leftarrow_L \mathrm{read}\ t\ (x, y)\ ;\ K(y)))$$

Now we can state our main theorem, making the intuition at the beginning of this section precise.

Theorem 4.1.

(i) *If* $\Gamma \vdash e \Leftarrow A_m \mid \Xi$ *then* $\Gamma \vdash [\![e]\!]d \div (d \Leftarrow A_m) \mid \Xi$ *(where* Γ *and* Ξ *consist only of a priori known* $y \Rightarrow B_k$*)*

(ii) *If* $\Gamma \vdash s \Rightarrow A_m \mid \Xi_1$ *and* $\Gamma, x \Rightarrow A_m \vdash K(x) \div (d \Leftarrow D_r) \mid (x \Rightarrow A_m \,;\, \Xi_2)$ *then* $\Gamma \vdash [s]K \div (d \Leftarrow D_r) \mid (\Xi_1 \,;\, \Xi_2)$

Proof. By rule induction on the given ND derivation. □

5 CONCLUSION

We provided a bidirectional type system for adjoint Sax. Then we defined a compilation from adjoint natural deduction that uses both destination passing style and a metalevel continuation. We proved this translation is type-preserving.

The continuations in the compilation are at the meta level, so we don't create extraneous redices, but the moving of the value from t to d in $[\![s]\!]d := [s](\underline{\lambda}t.\ \mathrm{id}\ d\ t)$ is at the object level. We do this to allow for subtyping and arrow direction switch. Is it possible to avoid some of these moves without breaking the underlying principles?

It's not immediately obvious with the limited number of rules provided, but the continuations all use their metavariable linearly. Do these intermediate variables also satisfy a stack discipline as those for translation to continuation-passing style [Danvy and Pfenning, 1995]? We conjecture that this is so.

With the context allowing both synthesis and checking, typing becomes highly nondeterministic unless we are strict about when we can change directions for antecedents. The version of Sax we present is in fact very strict. It means that many programs that might seem reasonable will not typecheck, and instead require some seemingly redundant snips. It seems possible to allow an (implicit) change from synthesis to checking with a corresponding application of subtyping in other rules besides the identity, but we have not yet developed the corresponding theory.

REFERENCES

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

Olivier Danvy and Frank Pfenning. The occurrence of continuation parameters in CPS terms. Technical Report CMU-CS-95-121, Department of Computer Science, Carnegie Mellon University, February 1995.

Henry DeYoung, Frank Pfenning, and Klaas Pruiksma. Semi-axiomatic sequent calculus. In Z. Ariola, editor, *5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020)*, pages 29:1–29:22, Paris, France, June 2020. LIPIcs 167.

Jana Dunfield and Neel Krishnaswami. Bidirectional typing. *ACM Computing Surveys*, 54(5):98:1–98:38, 2022.

Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

Junyoung Jang, Sophia Roshal, Frank Pfenning, and Brigitte Pientka. Adjoint natural deduction. In Jakob Rehof, editor, *9th International Conference on Formal Structures for Computation and Deduction (FSCD 2024)*, pages 15:1–15:23, Tallinn, Estonia, July 2024. LIPIcs 299. Extended version available as https://arxiv.org/abs/2402.01428.

Frank Pfenning. Lecture notes in higher order type compilation, February 2025. URL https://www.cs.cmu.edu/~fp/courses/15417-s25/lectures/14-adjsax.pdf.

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

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

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

Siva Kamesh Somayyajula. *Total Correctness Type Refinements for Communicating Processes.* Ph.D. thesis, Carnegie Mellon University, May 2024. Available as Technical Report CMU-CS-24-108.