Session Types in Abelian Logic

Yoichi Hirai*
National Institute of Advanced Industrial Science and Technology
y-hirai@aist.go.jp

There was a PhD student who says "I found a pair of wooden shoes. I put a coin in the left and a key in the right. Next morning, I found those objects in the opposite shoes." We do not claim existence of such shoes, but propose a similar programming abstraction in the context of typed lambda calculi. The result, which we call the Amida calculus, extends Abramsky's linear lambda calculus LF and characterizes Abelian logic.

1 Introduction

We propose a way to unify ML-style programming languages [29, 23] and π-calculus [28]. "Well-typed expressions do not go wrong," said Milner [27]. However, when communication is involved, how to maintain such a typing principle is not yet settled. For example, Haskell, which has types similar to the ML-style types, allows different threads to communicate using a kind of shared data store called an MVar `mv` of type `MVar a`, with commands `putMVar mv` of type `a -> IO ()` and `takeMVar mv` of type `IO a`. The former command consumes an argument of type `a` and the consumed argument appears from the latter command. However, if programmers make mistakes, these commands can cause a deadlock during execution even after the program passes type checking.

In order to prevent this kind of mistakes, a type system can force the programmer to use both the sender and the receiver each once. For doing this, we use the technique of linear types. Linear types are refinements of the ML-style intuitionistic types. Differently from intuitionistic types, linear types can specify a portion of program to be used just once. Linear types are used by Wadler [35] and Caires and Pfenning [9] to encode session types, but our type system can type processes that Wadler and Pfenning's system cannot.

As intuitionistic types are based on intuitionistic logic, linear types are based on linear logic. There are classical and intuitionistic variants of linear logics. From the intuitionistic linear logic, our only addition is the Amida axiom $(\varphi \multimap \psi) \otimes (\psi \multimap \varphi)$. We will see that the resulting logic is identical to Abelian logic [10] up to provability of formulae. In the Amida calculus, we can express π-calculus-like processes as macros. From the viewpoint of typed lambda calculi, a natural way to add the axiom $(\varphi \multimap \psi) \otimes (\psi \multimap \varphi)$ is to add a pair of primitives c and $\bar{c}$ so that $\cdots ct \cdots \bar{c}u \cdots$ reduces to $\cdots u \cdots t \cdots$: in words, c returns $\bar{c}$'s argument and vice versa. In the axiom, we can substitute the singleton type $\mathbf{1}$ for the general ψ to obtain an axiom standing for the send-receive communication primitive pair $(\varphi \multimap \mathbf{1}) \otimes (\mathbf{1} \multimap \varphi)$; the left hand side c of type $\varphi \multimap \mathbf{1}$ is the sending primitive and the right hand side $\bar{c}$ of type $\mathbf{1} \multimap \varphi$ is the receiving primitive. The sending primitive consumes a data of type φ and produces a meaningless data of unit type $\mathbf{1}$. The receiving primitive takes the meaningless data of type $\mathbf{1}$ and produces a data of type φ.

*When this work was presented at the workshop, the author was a student at the University of Tokyo and a JSPS fellow supported by Grant-in-Aid for JSPS Fellows 23-6978.

N. Yoshida, W. Vanderbauwhede (Eds.): Programming Language Approaches to Concurrency- and Communication-cEntric Software 2013 (PLACES'13)
EPTCS 137, 2013, pp. 33–52, doi:10.4204/EPTCS.137.4

When we want to use these primitives in lambda terms, there is one problem: what happens to $\overline{c}(ct)$? In this case, we do not know the output of c because the output of c comes from $\overline{c}$'s input, which is the output of c. Fortunately, we just want to know the output of $\overline{c}$, which is the input of c, that is, t. In a more complicated case $\overline{c}(\overline{d}(c(dt)))$, we can reason the output of $\overline{c}$ as the input of c as the output of d as the input of $\overline{d}$ as the output of c as the input of $\overline{c}$ as the output of $\overline{d}$ as the input of d, which is t.

Our first contribution is encoding of session types into a linear type system. Although the approach is similar to that of Caires and Pfenning [9] and Wadler [35], the Amida calculus has an additional axiom so that it can type some processes that Caires-Pfenning or Wadler's type systems cannot. In essence, the axiom allows two processes to wait for one another and then exchange information.

Our second contribution is a side effect of our first contribution. The type system we developed is a previously unknown proof system for Abelian logic [10]. In this paper, we introduce the axioms of the form $(\varphi \multimap \psi) \otimes (\psi \multimap \varphi)$ on top of IMALL, intuitionistic multiplicative additive linear logic.

Our third contribution is the use of conjunctive hypersequents. Hypersequents have been around since Avron [3], but in all cases, different components in a hypersequent were interpreted disjunctively. In our formalization of Abelian logic, we use conjunctive hypersequents, where different components are interpreted conjunctively. This is the first application of such conjunctive hypersequents.

Later in this paper, we address some issues about consistency (Theorem 4.11), complicated protocols (Section 4) and encoding process calculi (Section 4).

2 Definitions

Types We assume a countably infinite set of *propositional variables*, for which we use letters X, Y and so on. We define a type φ by BNF: $\varphi ::= \mathbf{1} \mid X \mid \varphi \otimes \varphi \mid \varphi \multimap \varphi \mid \varphi \oplus \varphi \mid \varphi \,\&\, \varphi$. A *formula* is a type. As the typing rules (Figure 1) reveal, $\otimes$ is the multiplicative conjunction, $\multimap$ is the multiplicative implication, $\oplus$ is the additive disjunction and & is the additive conjunction.

Terms and Free Variables We assume countably infinitely many variables $x, y, z, \ldots$. Before defining terms, following Abramsky's linear lambda calculus LF [1], we define *patterns* binding sets of variables:

- $*$ is a pattern binding $\emptyset$,
- $\langle x, \_\rangle$ and $\langle \_, x\rangle$ are patterns binding $\{x\}$,
- $x \otimes y$ is a pattern binding $\{x, y\}$.

All patterns are from Abramsky's LF [1]. Using patterns, we inductively define a *term* t with *free variables* S. We assume countably infinitely many *channels* with involution satisfying $\overline{c} \neq c$ and $\overline{\overline{c}} = c$.

- $*$ is a term with free variables $\emptyset$,
- a variable x is a term with free variables $\{x\}$,
- if t is a term with free variables S, u is a term with free variables S', and moreover S and S' are disjoint, then $t \otimes u$ and tu are terms with free variables $S \cup S'$,
- if t and u are terms with free variables S, then $\langle t, u\rangle$ is a term with free variables S,
- if t is a term with free variables S, then $\mathsf{inl}(t)$ and $\mathsf{inr}(t)$ are terms with free variables S,
- if t is a term with free variables $S \cup \{x\}$ and x is not in S, then $\lambda x.t$ is a term with free variables S,
- if t is a term with free variables S, p is a pattern binding S', u is a term with free variables $S' \cup S''$ and equalities $S \cap S'' = S' \cap S'' = \emptyset$ hold, then, $\mathsf{let}\, t\, \mathsf{be}\, p\, \mathsf{in}\, u$ is a term with free variables $S \cup S''$,

- if t is a term with free variables S, u is a term with free variables $S'' \cup \{x\}$, v is a term with free variables $S'' \cup \{y\}$, $x, y \notin S''$ and $S \cap S'' = \emptyset$ hold, then

$$\mathsf{match}\, t\, \mathsf{of}\, \mathsf{inl}(x).u/\mathsf{inr}(y).v$$

  is a term with free variables $S \cup S''$, and

- if t is a term with free variables S, then ct is also a term with free variables S for any channel c.

Only the last clause is original, introducing channels, which are our communication primitives. Note that a term with free variables S is not a term with free variables S' when S and S' are different (even if S is a subset of S'). In other words, the set of free variables $FV(t)$ is uniquely defined for a term t. We introduce an abbreviation

$$\mathsf{ign}\, \varepsilon\, \mathsf{in}\, t \equiv t$$
$$\mathsf{ign}\, s_0, \overrightarrow{s}\, \mathsf{in}\, t \equiv \mathsf{let}\, s_0\, \mathsf{be} * \mathsf{in}\, (\mathsf{ign}\, \overrightarrow{s}\, \mathsf{in}\, t)$$

inductively for a sequence of terms $\overrightarrow{s}$. Here ε stands for the empty sequence. The symbol ign is intended to be pronounced "ignore."

Typing Derivations On top of Abramsky's linear lambda calculus LF [1], we add a rule to make a closed term of type $(\varphi \multimap \psi) \otimes (\psi \multimap \varphi)$. A *context* Γ is a possibly empty sequence of variables associated with types where the same variable appears at most once. A context $x : X, y : Y$ is allowed, but $x : X, x : Y$ or $x : X, x : X$ is not a context. A *hypersequent* is inductively defined as $\mathscr{O} ::= \varepsilon \mid (\Gamma \vdash t : \varphi \,\|\, \mathscr{O})$ where Γ is a context. Each $\Gamma \vdash t : \varphi$ is called a *component* of a hypersequent. In this paper, we interpret the components conjunctively. Differently from the previous papers [3, 5, 4], here, the hypersequent $\Gamma \vdash \varphi \,\|\, \Delta \vdash \psi$ is interpreted as the conjunction of components: $(\bigotimes \Gamma \multimap \varphi) \otimes (\bigotimes \Delta \multimap \psi)$ where $\bigotimes \Gamma$ stands for the $\otimes$-conjunction of elements of Γ. The conjunctive treatment is our original invention, and finding an application of such a treatment is one of our contributions. We name this technique the *conjunctive hypersequent*. We have to note that, for Abelian logic, there is an ordinary disjunctive hypersequent system [24] that enjoys cut-elimination. We still claim that the conjunctive hypersequents reflect some computational intuition on concurrently running multiple processes, all of which are supposed to succeed (as opposed to the disjunctive interpretation where at least one of which is supposed succeed, e.g. Hirai's calculus for Gödel-Dummett logic [15]).

The typing rules of the *Amida calculus* are in Figure 1. Most rules are straightforward modification of Abramsky's rules [1]. The Sync rule is original. Rules &R and $\oplus$L are only applicable to singleton hypersequents. When $\vdash t : \varphi$ is derivable, the type φ is *inhabited*.

Example 2.1 (Derivation of the Amida axiom)**.** *The type $(\varphi \multimap \psi) \otimes (\psi \multimap \varphi)$ is inhabited by the following derivation.*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x : \varphi \vdash x : \varphi}\,Ax \quad \dfrac{}{y : \psi \vdash y : \psi}\,Ax}{x : \varphi \vdash x : \varphi \,\|\, y : \psi \vdash y : \psi}\,Merge}{x : \varphi \vdash cx : \psi \,\|\, y : \psi \vdash \bar{c}y : \varphi}\,Sync}{\vdash \lambda x.cx : \varphi \multimap \psi \,\|\, y : \psi \vdash \bar{c}y : \varphi}\,{\multimap}R}{\vdash \lambda x.cx : \varphi \multimap \psi \,\|\, \vdash \lambda y.\bar{c}y : \psi \multimap \varphi}\,{\multimap}R}{\vdash (\lambda x.cx) \otimes (\lambda y.\bar{c}y) : (\varphi \multimap \psi) \otimes (\psi \multimap \varphi)}\,{\otimes}R$$

Another example shows how we can type the term $\bar{c}(cx)$.

$$\text{Ax}\ \frac{}{x:\varphi\vdash x:\varphi}\qquad \text{Merge}\ \frac{\mathscr{O}\qquad \mathscr{O}'}{\mathscr{O}\,\|\,\mathscr{O}'}\qquad \text{Cut}\ \frac{\mathscr{O}\,\|\,\Gamma\vdash t:\varphi\,\|\,x:\varphi,\Delta\vdash u:\psi}{\mathscr{O}\,\|\,\Gamma,\Delta\vdash u[t/x]:\psi}$$

$$\text{IE}\ \frac{\mathscr{O}\,\|\,\Gamma,x:\varphi,y:\psi,\Delta\vdash t:\theta}{\mathscr{O}\,\|\,\Gamma,y:\psi,x:\varphi,\Delta\vdash t:\theta}\qquad \text{EE}\ \frac{\mathscr{O}\,\|\,\Gamma\vdash t:\varphi\,\|\,\Delta\vdash u:\psi\,\|\,\mathscr{O}'}{\mathscr{O}\,\|\,\Delta\vdash u:\psi\,\|\,\Gamma\vdash t:\varphi\,\|\,\mathscr{O}'}\qquad \mathbf{1}\text{R}\ \frac{}{\vdash *:\mathbf{1}}$$

$$\mathbf{1}\text{L}\ \frac{\mathscr{O}\,\|\,\Gamma\vdash t:\varphi}{\mathscr{O}\,\|\,\Gamma,z:\mathbf{1}\vdash \mathsf{ign}\,z\,\mathsf{in}\,t:\varphi}\qquad \otimes\text{R}\ \frac{\mathscr{O}\,\|\,\Gamma\vdash t:\varphi\,\|\,\Delta\vdash u:\psi}{\mathscr{O}\,\|\,\Gamma,\Delta\vdash t\otimes u:\varphi\otimes\psi}\qquad \text{Sync}\ \frac{\mathscr{O}\,\|\,\Gamma\vdash t:\varphi\,\|\,\Delta\vdash u:\psi}{\mathscr{O}\,\|\,\Gamma\vdash ct:\psi\,\|\,\Delta\vdash \bar{c}u:\varphi}$$

(*c* and $\bar{c}$ uniquely introduced here)

$$\otimes\text{L}\ \frac{\mathscr{O}\,\|\,\Gamma,x:\varphi,y:\psi\vdash t:\theta}{\mathscr{O}\,\|\,\Gamma,z:\varphi\otimes\psi\vdash \mathsf{let}\,z\,\mathsf{be}\,x\otimes y\,\mathsf{in}\,t:\theta}\qquad \multimap\text{R}\ \frac{\mathscr{O}\,\|\,\Gamma,x:\varphi\vdash t:\psi}{\mathscr{O}\,\|\,\Gamma\vdash\lambda x.t:\varphi\multimap\psi}\qquad \multimap\text{L}\ \frac{\mathscr{O}\,\|\,\Gamma\vdash t:\varphi\,\|\,x:\psi,\Delta\vdash u:\theta}{\mathscr{O}\,\|\,\Gamma,f:\varphi\multimap\psi,\Delta\vdash u[(ft)/x]:\theta}$$

$$\&\text{R}\ \frac{\Gamma\vdash t:\varphi\qquad \Gamma\vdash u:\psi}{\Gamma\vdash\langle t,u\rangle:\varphi\,\&\,\psi}\qquad \&\text{L}_0\ \frac{\mathscr{O}\,\|\,\Gamma,x:\varphi\vdash t:\theta}{\mathscr{O}\,\|\,\Gamma,z:\varphi\,\&\,\psi\vdash \mathsf{let}\,z\,\mathsf{be}\,\langle x,\_\rangle\,\mathsf{in}\,t:\theta}\qquad \&\text{L}_1\ \frac{\mathscr{O}\,\|\,\Gamma,y:\psi\vdash t:\theta}{\mathscr{O}\,\|\,\Gamma,z:\varphi\,\&\,\psi\vdash \mathsf{let}\,z\,\mathsf{be}\,\langle\_,y\rangle\,\mathsf{in}\,t:\theta}$$

$$\oplus\text{R}_0\ \frac{\mathscr{O}\,\|\,\Gamma\vdash t:\varphi}{\mathscr{O}\,\|\,\Gamma\vdash\mathsf{inl}(t):\varphi\oplus\psi}\qquad \oplus\text{R}_1\ \frac{\mathscr{O}\,\|\,\Gamma\vdash u:\psi}{\mathscr{O}\,\|\,\Gamma\vdash\mathsf{inr}(u):\varphi\oplus\psi}\qquad \oplus\text{L}\ \frac{\Gamma,x:\varphi\vdash u:\theta\qquad \Gamma,y:\psi\vdash v:\theta}{\Gamma,z:\varphi\oplus\psi\vdash\mathsf{match}\,z\,\mathsf{of}\,\mathsf{inl}(x).u/\mathsf{inr}(y).v:\theta}$$

Figure 1: The typing rules of the Amida calculus. $\mathscr{O}$ and $\mathscr{O}'$ stand for hypersequents.

$$\textit{Cut}\ \dfrac{\textit{Sync}\ \dfrac{\textit{Merge}\ \dfrac{\textit{Ax}\ \dfrac{}{x:\varphi\vdash x:\varphi}\qquad \textit{Ax}\ \dfrac{}{y:\psi\vdash y:\psi}}{x:\varphi\vdash x:\varphi\,\|\,y:\psi\vdash y:\psi}}{x:\varphi\vdash cx:\psi\,\|\,y:\psi\vdash\bar{c}y:\varphi}}{x:\varphi\vdash\bar{c}(cx):\varphi}$$

Evaluation As a programming language, the Amida calculus is equipped with an operational semantics that evaluates some closed hyper-terms into a sequence of canonical forms. The *canonical forms* are the same as those of Abramsky's LF [1]:

$$\langle t,u\rangle\qquad *\qquad v\otimes w\qquad \lambda x.t\qquad \mathsf{inl}(v)\qquad \mathsf{inr}(w)$$

where v and w are canonical forms and t and u are terms.

An *evaluation sequence* $\mathscr{E}$ is defined by the following grammar:

$$\mathscr{E} ::= \varepsilon \mid (t\Downarrow v\,\|\,\mathscr{E})$$

where t is a term and v is a canonical form. Now we define evaluation as a set of evaluation sequences (Figure 2). Though most rules are similar to those of Abramsky's LF [1], we add the semantics for channels. It is noteworthy that the results of evaluation are always canonical forms.

3 Type Safety

When we can evaluate a derivable hypersequent, the result is also derivable. Especially, this shows that, whenever a communicating term is used, the communicating term is used according to the types shown in the Sync rule occurrence introducing the communicating term.

$$\frac{}{* \Downarrow *} \qquad \frac{\mathscr{E} \parallel t \Downarrow * \parallel u \Downarrow v}{\mathscr{E} \parallel \mathsf{ign}\, t\, \mathsf{in}\, u \Downarrow v} \qquad \frac{\mathscr{E} \parallel t \Downarrow v \parallel u \Downarrow w}{\mathscr{E} \parallel t \otimes u \Downarrow v \otimes w} \qquad \frac{\mathscr{E} \parallel t \Downarrow v \otimes w \parallel u[v/x, w/y] \Downarrow v'}{\mathscr{E} \parallel \mathsf{let}\, t\, \mathsf{be}\, x \otimes y\, \mathsf{in}\, u \Downarrow v'}$$

$$\text{Merge}\ \frac{\mathscr{E} \qquad \mathscr{E}'}{\mathscr{E} \parallel \mathscr{E}'}\ \text{(For any channel } c\text{, it is not the case that } \mathscr{E} \text{ contains } c \text{ and } \mathscr{E}' \text{ contains } \bar{c}.)$$

$$\frac{}{\lambda x.t \Downarrow \lambda x.t}$$

$$\frac{\mathscr{E} \parallel t \Downarrow \lambda x.t' \parallel u \Downarrow v \parallel t'[v/x] \Downarrow w}{\mathscr{E} \parallel tu \Downarrow w} \qquad \frac{\mathscr{E} \parallel t \Downarrow v \parallel u \Downarrow w}{\mathscr{E} \parallel ct \Downarrow w \parallel \bar{c}u \Downarrow v}\ (\mathscr{E}, t \text{ and } u \text{ do not contain } c \text{ or } \bar{c}.)$$

$$\frac{\mathscr{E} \parallel t \Downarrow t' \parallel s \Downarrow s' \parallel \mathscr{E}'}{\mathscr{E} \parallel s \Downarrow s' \parallel t \Downarrow t' \parallel \mathscr{E}'} \qquad \frac{}{\langle t, u \rangle \Downarrow \langle t, u \rangle} \qquad \frac{\mathscr{E} \parallel t \Downarrow \langle t_0, t_1 \rangle \parallel u[t_0/x] \Downarrow w}{\mathscr{E} \parallel \mathsf{let}\, t\, \mathsf{be}\, \langle x, \_ \rangle\, \mathsf{in}\, u \Downarrow w}$$

$$\frac{\mathscr{E} \parallel t \Downarrow \langle t_0, t_1 \rangle \parallel u[t_1/y] \Downarrow w}{\mathscr{E} \parallel \mathsf{let}\, t\, \mathsf{be}\, \langle \_, y \rangle\, \mathsf{in}\, u \Downarrow w} \qquad \frac{\mathscr{E} \parallel t \Downarrow v}{\mathscr{E} \parallel \mathsf{inl}(t) \Downarrow \mathsf{inl}(v)} \qquad \frac{\mathscr{E} \parallel u \Downarrow w}{\mathscr{E} \parallel \mathsf{inr}(u) \Downarrow \mathsf{inr}(w)}$$

$$\frac{\mathscr{E} \parallel t \Downarrow \mathsf{inl}(v) \parallel u[v/x] \Downarrow w}{\mathscr{E} \parallel \mathsf{match}\, t\, \mathsf{of}\, \mathsf{inl}(x).u/\mathsf{inr}(y).u' \Downarrow w} \qquad \frac{\mathscr{E} \parallel t \Downarrow \mathsf{inr}(v) \parallel u'[v/y] \Downarrow w}{\mathscr{E} \parallel \mathsf{match}\, t\, \mathsf{of}\, \mathsf{inl}(x).u/\mathsf{inr}(y).u' \Downarrow w}$$

Figure 2: The definition of evaluation relation of the Amida calculus. $\mathscr{E}$ is possibly the empty evaluation sequence.

Theorem 3.1 (Type Preservation of the Amida calculus)**.** *If terms $t_0, \dots, t_n$ have a hypersequent $\vdash t_0 : \varphi_0 \parallel \cdots \parallel \vdash t_n : \varphi_n$ and an evaluation sequence $t_0 \Downarrow v_0 \parallel \cdots \parallel t_n \Downarrow v_n$ derivable, then $\vdash v_0 : \varphi_0 \parallel \cdots \parallel \vdash v_n : \varphi_n$ is also derivable.*

Proof. By induction on evaluation using the propositions below. We analyze the cases by the last rule.

(Merge) By Proposition 3.2, we can use the induction hypothesis.

($\mathsf{let}\, t\, \mathsf{be}\, \langle x, \_ \rangle\, \mathsf{in}\, u$) By Proposition 3.3, we can use the induction hypothesis.

(Other cases) Similar to above. □

Two hypersequents $\mathscr{O}$ and $\mathscr{O}'$ are *channel-disjoint* if and only if it is not the case that $\mathscr{O}$ contains c and $\mathscr{O}'$ contains $\bar{c}$ for any channel c.

Proposition 3.2 (Split)**.** *If a type derivation leading to $\mathscr{O} \parallel \mathscr{O}'$ exists for two channel-disjoint hypersequents, both $\mathscr{O}$ and $\mathscr{O}'$ are derivable separately.*

Proof. By induction on the type derivation. □

Proposition 3.3 (Inversion on &L)**.** *If $\mathscr{O} \parallel \Gamma \vdash \mathsf{let}\, t\, \mathsf{be}\, \langle x, \_ \rangle\, \mathsf{in}\, u : \theta$ is derivable, then there is a partition of Γ into Γ_0 and Γ_1 (up to exchange) such that $\mathscr{O} \parallel \Gamma_0 \vdash t : \varphi \,\&\, \psi \parallel \Gamma_1, x : \varphi \vdash u : \theta$ is derivable.*

Proof. By induction on the original derivation. □

Determinacy states that if $t \Downarrow v$ and $t \Downarrow w$ hold, then v and w are identical. Since our evaluation is given to possibly multiple terms at the same time, it is easier to prove a more general version.

Theorem 3.4 (General Determinacy of the Amida calculus)**.** *If* $t_0 \Downarrow v_0 \;\|\; t_1 \Downarrow v_1 \;\|\; \cdots \;\|\; t_n \Downarrow v_n$ *and* $t_0 \Downarrow w_0 \;\|\; t_1 \Downarrow w_1 \;\|\; \cdots \;\|\; t_n \Downarrow w_n$ *hold, then each* v_i *is identical to* w_i*.*

Proof. By induction on the height of evaluation derivation. Each component in the conclusion has only one applicable rule. Also, the order of decomposing different components is irrelevant (the crucial condition is freshness of c and $\overline{c}$ in Figure 2). □

Convergence would state that whenever a closed term t is typed $\vdash t : \varphi$, then an evaluation $t \Downarrow v$ is also derivable for some canonical form v. It is a desirable property so that Abramsky [1] proves it for LF, but there are counter examples against convergence of the Amida calculus. Consider a typed term $\vdash c(\overline{c}(\mathsf{inl}(*))) : \mathbf{1} \oplus \mathbf{1}$ with no evaluation. One explanation for the lack of evaluation is deadlock. This illustrates that the current form of Amida calculus lacks deadlock-freedom. In order to avoid the deadlock and to evaluate this closed term, we can add the following eval-subst rule:

$$\text{eval-subst}\ \frac{\mathscr{E} \;\|\; t \Downarrow v \;\|\; u[v/x] \Downarrow w}{\mathscr{E} \;\|\; u[t/x] \Downarrow w},$$

which enables an evaluation $c(\overline{c}(\mathsf{inl}(*))) \Downarrow \mathsf{inl}(*)$. Moreover, the eval-subst rule enables an evaluation $\overline{c}[C[cv]] \Downarrow v$, which reminds us of the call-with-current-continuation primitive [31] and shift/reset primitives [12, 2]. However, adding the eval-subst rule breaks the current proof of Theorem 3.1 (safety), but with some modifications, the safety property can possibly be proved. The main difficulty in proving the safety property can be seen in the form of eval-subst rule. When we only know the conclusion of an eval-subst occurrence, there are many possible assumptions involving free variables, all of which we must consider if we are to prove the type safety.

4 Session Types and Processes as Abbreviations

In order to see the usefulness of the communication primitives, we try implementing a process calculus and a session type system using the Amida calculus.

Session Types as Abbreviations As an abbreviation, we introduce *session types*. Session types [33, 17] can specify a communication protocol over a channel. The following definitions and the descriptions are modification from Wadler's translations and descriptions of session types [35]. The notation here is different from the original notation by Takeuchi, Honda and Kubo [33].

$!\varphi\ \psi \equiv \varphi \multimap \psi$	output a value of φ then behave as ψ
$?\varphi\ \psi \equiv \varphi \otimes \psi$	input a value of φ then behave as ψ
$\oplus\{l_i\colon \varphi_i\}_{i\in I} \equiv \varphi_0 \,\&\, \cdots \,\&\, \varphi_n, \quad I = \{0,\dots,n\}$	select from φ_i with label l_i
$\&\{l_i\colon \varphi_i\}_{i\in I} \equiv \varphi_0 \oplus \cdots \oplus \varphi_n, \quad I = \{0,\dots,n\}$	offer choice of φ_i with label l_i
$\mathsf{end} \equiv \mathbf{1}$	terminator

where I is a finite downward-closed set of natural numbers like $\{0,1,2,3\}$. As Wadler [35] notes, the encoding looks opposite of what some would expect, but as Wadler [35] explains, we are typing channels instead of processes.

The grammar $\varphi,\psi ::= \mathsf{end} \mid X \mid {!\varphi}\ \psi \mid {?\varphi}\ \psi \mid \oplus\{l_i\colon \varphi_i\}_{i\in I} \mid \&\{l_i\colon \varphi_i\}_{i\in I}$ covers all types. A linear type ($\varphi^{\sim}$ possibly with subscript) is generated by this grammar:

$$\varphi^{\sim} ::= \mathsf{end} \mid {!\psi}\ \varphi^{\sim} \mid {?\psi}\ \varphi^{\sim} \mid \oplus\{l_i\colon \varphi_i^{\sim}\}_{i\in I} \mid \&\{l_i\colon \varphi_i^{\sim}\}_{i\in I}$$

We define duals of linear types. Again the definition is almost the same as Wadler's [35] except that end is self-dual.

$$\overline{{!\psi}\ \varphi^{\sim}} = {?\psi}\ \overline{\varphi^{\sim}} \qquad \overline{{?\psi}\ \varphi^{\sim}} = {!\psi}\ \overline{\varphi^{\sim}}$$
$$\overline{\oplus\{l_i\colon \varphi_i^{\sim}\}_{i\in I}} = \&\{l_i\colon \overline{\varphi_i^{\sim}}\}_{i\in I} \qquad \overline{\&\{l_i\colon \varphi_i^{\sim}\}_{i\in I}} = \oplus\{l_i\colon \overline{\varphi_i^{\sim}}\}_{i\in I}$$
$$\overline{\mathsf{end}} = \mathsf{end}\ .$$

Processes as Abbreviations We define the sending and receiving constructs of process calculi as abbreviations:

$x\langle u\rangle.t \equiv t[(xu)/x]$	send u through channel x and then use x in t
$x(y).t \equiv \mathsf{let}\,x\,\mathsf{be}\,y\otimes x\,\mathsf{in}\,t$	receive y through channel x and use x and y in t
$0 \equiv *$	do nothing

We have to be careful about substitution combined with process abbreviations. For example, $(x\langle u\rangle.t)[s/x]$ is not $s\langle u\rangle.t$ because the latter is not defined. Following the definition, $(x\langle u\rangle.t)[s/x]$ is actually $(t[xu/x])[s/x] = t[su/x]$. We are going to introduce the name restriction $\nu x.t$ after implementing channels.

Below, we are going to justify these abbreviations statically and dynamically.

Process Typing Rules as Abbreviations The session type abbreviation and the processes abbreviation allow us to use the typing rules in the next proposition.

Proposition 4.1 (Process Typing Rules: senders and receivers)**.** *These rules are admissible.*

$$\text{recv}\ \frac{\mathscr{O} \,\|\, y\colon\psi, x\colon\chi \vdash t\colon\varphi}{\mathscr{O} \,\|\, x\colon{?\psi}\ \chi \vdash x(y).t\colon\varphi} \quad \text{send}\ \frac{\mathscr{O} \,\|\, \Gamma, x\colon\chi \vdash t\colon\varphi \,\|\, \Delta \vdash u\colon\psi}{\mathscr{O} \,\|\, \Gamma,\Delta, x\colon{!\psi}\ \chi \vdash x\langle u\rangle.t\colon\varphi} \quad \text{end}\ \frac{\mathscr{O} \,\|\, \Gamma \vdash t\colon\varphi}{\mathscr{O} \,\|\, \Gamma, x\colon\mathsf{end} \vdash \mathsf{ign}\,x\,\mathsf{in}\,t\colon\varphi} \quad \frac{}{\vdash 0\colon\mathbf{1}}$$

Proof. Immediate. ☐

We note that the types of variable x change in the rules. This reflects the intuition of session types: the session type of a channel changes after some communication occurs through the channel.

Example 4.2 (Typed communicating terms)**.** *Using Theorem 4.1, we can type processes. Figure 3 contains one process, which sends a channel y through x and then waits for input in a channel y'. Here is another process that takes an input w' from channel x', where the input w' itself is expected to be a channel. After receiving w', the process puts* $\mathsf{inl}(*)$ *in w'.*

$$\textit{recv}\ \cfrac{\textit{end}\ \cfrac{\textit{send}\ \cfrac{\textit{end}\ \cfrac{\mathbf{1}R\ \cfrac{}{\vdash *\colon\mathbf{1}}}{w'\colon\mathsf{end} \vdash \mathsf{ign}\,w'\,\mathsf{in}\,*\colon\mathbf{1}} \qquad \oplus R\ \cfrac{\mathbf{1}R\ \cfrac{}{\vdash *\colon\mathbf{1}}}{\vdash \mathsf{inl}(*)\colon\mathbf{1}\oplus\mathbf{1}}}{w'\colon{!(\mathbf{1}\oplus\mathbf{1})}\ \mathsf{end} \vdash w'\langle\mathsf{inl}(*)\rangle.\,\mathsf{ign}\,w'\,\mathsf{in}\,*\colon\mathbf{1}}}{w'\colon{!(\mathbf{1}\oplus\mathbf{1})}\ \mathsf{end}, x'\colon\mathsf{end} \vdash \mathsf{ign}\,x'\,\mathsf{in}\,w'\langle\mathsf{inl}(*)\rangle.\,\mathsf{ign}\,w'\,\mathsf{in}\,*\colon\mathbf{1}}}{x'\colon{?({!(\mathbf{1}\oplus\mathbf{1})}\ \mathsf{end})}\ \mathsf{end} \vdash x'(w').\,\mathsf{ign}\,x'\,\mathsf{in}\,w'\langle\mathsf{inl}(*)\rangle.\,\mathsf{ign}\,w'\,\mathsf{in}\,*\colon\mathbf{1}}$$

$$
\text{send}\ \dfrac{\text{recv}\ \dfrac{\text{end}\ \dfrac{\text{end}\ \dfrac{\text{Ax}\ \dfrac{}{z : \mathbf{1} \oplus \mathbf{1} \vdash z : \mathbf{1} \oplus \mathbf{1}}}{z : \mathbf{1} \oplus \mathbf{1}, y : \mathsf{end} \vdash \mathsf{ign}\, y\, \mathsf{in}\, z : \mathbf{1} \oplus \mathbf{1}}}{z : \mathbf{1} \oplus \mathbf{1}, x : \mathsf{end}, y : \mathsf{end} \vdash \mathsf{ign}\, x, y\, \mathsf{in}\, z : \mathbf{1} \oplus \mathbf{1}}}{x : \mathsf{end}, y : ?(\mathbf{1} \oplus \mathbf{1})\, \mathsf{end} \vdash y(z).\, \mathsf{ign}\, x, y\, \mathsf{in}\, z : \mathbf{1} \oplus \mathbf{1}} \qquad \text{Ax}\ \dfrac{}{y' : !(\mathbf{1} \oplus \mathbf{1})\, \mathsf{end} \vdash y' : !(\mathbf{1} \oplus \mathbf{1})\, \mathsf{end}}}{y : ?(\mathbf{1} \oplus \mathbf{1})\, \mathsf{end}, x : !(!(\mathbf{1} \oplus \mathbf{1})\, \mathsf{end})\, \mathsf{end}, y' : !(\mathbf{1} \oplus \mathbf{1})\, \mathsf{end} \vdash x\langle y \rangle . y'(z).\, \mathsf{ign}\, x, y\, \mathsf{in}\, z : \mathbf{1} \oplus \mathbf{1}}
$$

Figure 3: A typed process.

Implementing Channels. We introduced primitives c and $\overline{c}$ implementing the behavior specified by $(\mathbf{1} \multimap \varphi) \otimes (\varphi \multimap \mathbf{1})$. These primitives can be seen as channels of session types $?\varphi\, \mathsf{end}$ and $!\varphi\, \mathsf{end}$. Indeed, $?\varphi\, \mathsf{end}$ is $\varphi \otimes \mathbf{1}$ (which is inter-derivable with $\mathbf{1} \multimap \varphi$) and $!\varphi\, \mathsf{end}$ is $\varphi \multimap \mathbf{1}$. We can generalize this phenomenon to the more complicated session types[1].

Proposition 4.3 (Session realizers)**.** *For any linear type $\varphi^{\sim}$, the hypersequent $\vdash t : \varphi^{\sim} \mid \vdash u : \overline{\varphi^{\sim}}$ is derivable for some terms t and u.*

Proof. Induction on $\varphi^{\sim}$.

(end) $\text{Merge}\ \dfrac{\text{Ax}\ \dfrac{}{\vdash * : \mathbf{1}} \qquad \text{Ax}\ \dfrac{}{\vdash * : \mathbf{1}}}{\vdash * : \mathbf{1} \mid \vdash * : \mathbf{1}}$ is what we seek.

$(!\psi\, \varphi^{\sim})$ By the induction hypothesis, $\vdash t' : \varphi^{\sim} \mid \vdash u' : \overline{\varphi^{\sim}}$ is derivable. Using this, we can make the following derivation:

$$
\dfrac{\otimes\text{R}\ \dfrac{\text{Sync}\ \dfrac{\text{Merge}\ \dfrac{\text{Ax}\ \dfrac{}{x : \psi \vdash x : \psi} \qquad \dfrac{\text{IH}}{\vdash t' : \varphi^{\sim} \mid \vdash u' : \overline{\varphi^{\sim}}}}{x : \psi \vdash x : \psi \mid \vdash t' : \varphi^{\sim} \mid \vdash u' : \overline{\varphi^{\sim}}}}{cx : \varphi^{\sim} \vdash x : \psi \mid \vdash \overline{c}t' : \psi \mid \vdash u' : \overline{\varphi^{\sim}}}}{x : \psi \vdash cx : \varphi^{\sim} \mid \vdash (\overline{c}t') \otimes u' : \psi \otimes \overline{\varphi^{\sim}}}}{\vdash \lambda x.cx : \psi \multimap \varphi^{\sim} \mid \vdash (\overline{c}t') \otimes u' : \psi \otimes \overline{\varphi^{\sim}}}\ .
$$

$(?\psi\, \varphi)$ Symmetric to the above.

$(\oplus\{l_i : \varphi_i\})$ By the induction hypothesis, for each $i \in I$, we have $\vdash t_i : \varphi_i \mid \vdash u_i : \overline{\varphi_i}$ derived. Hence derivable is $\vdash t_i : \varphi_i \mid \vdash i(u_i) : \oplus_{j \in I} \overline{\varphi_j}$ where $i(u_i)$ is an appropriate nesting of $\mathsf{inl}(\cdot)$, $\mathsf{inr}(\cdot)$ and u_i. Combining $|I|$ such derivations, we can derive $\vdash \langle t_i \rangle_{i \in I} : \&_{i \in I} \varphi_i \mid \langle \vdash i(u_i) : \oplus_{j \in I} \overline{\varphi_j} \rangle_{i \in I}$ for a fresh natural number n.

$(\&\{l_i : \varphi_i\})$ Symmetric to above. □

We call the above pair t, u in the statement the *session realizers* of $\varphi^{\sim}$ and denote them by $\triangleright(\varphi^{\sim}), \triangleleft(\varphi^{\sim})$. Moreover, we use $\triangleright\triangleleft(\varphi^{\sim})$ to denote the pair $\triangleright(\varphi^{\sim}) \otimes \triangleleft(\varphi^{\sim})$. So far, a free variable with a linear type represented a channel serving the corresponding session type. Now, we can substitute the free variables with the session realizers so that the typed processes can actually communicate. If we have two terms that use free variables of type $\varphi^{\sim}$ and $\overline{\varphi^{\sim}}$, we can replace those free variables by session realizers.

Corollary 4.4 (Binding both ends of a channel)**.** *If $\mathscr{O} \mid \Gamma, x : \varphi^{\sim} \vdash t : \psi \mid \Delta, y : \overline{\varphi^{\sim}} \vdash u : \theta$ is derivable, then $\mathscr{O} \mid \Gamma \vdash t[\triangleright(\varphi^{\sim})/x] : \psi \mid \Delta \vdash u[\triangleleft(\varphi^{\sim})/y] : \theta$ is also derivable.*

[1] This is impossible using the ordinary linear types.

$$\dfrac{\mathscr{E} \;\rule{2pt}{10pt}\; \rhd(\varphi^{\sim}) \Downarrow v' \;\rule{2pt}{10pt}\; \lhd(\varphi^{\sim}) \Downarrow w' \;\rule{2pt}{10pt}\; t_0[v'/x] \Downarrow v \;\rule{2pt}{10pt}\; u \Downarrow u' \;\rule{2pt}{10pt}\; t_1[u'/z][w'/y] \Downarrow w}{\mathscr{E} \;\rule{2pt}{10pt}\; \rhd(\varphi^{\sim}) \Downarrow v' \;\rule{2pt}{10pt}\; \lhd(\varphi^{\sim}) \Downarrow w' \;\rule{2pt}{10pt}\; t_0[v'/x] \Downarrow v \;\rule{2pt}{10pt}\; u \Downarrow u' \;\rule{2pt}{10pt}\; \mathsf{let}\, u' \otimes w' \,\mathsf{be}\, z \otimes y \,\mathsf{in}\, t_1 \Downarrow w}$$

$$\dfrac{\dfrac{}{\lambda x.cx \Downarrow \lambda x.cx} \qquad \mathscr{E} \;\rule{2pt}{10pt}\; \rhd(\varphi^{\sim}) \Downarrow v' \;\rule{2pt}{10pt}\; \lhd(\varphi^{\sim}) \Downarrow w' \;\rule{2pt}{10pt}\; t_0[v'/x] \Downarrow v \;\rule{2pt}{10pt}\; u \Downarrow u' \;\rule{2pt}{10pt}\; \mathsf{let}\, u' \otimes w' \,\mathsf{be}\, z \otimes y \,\mathsf{in}\, t_1 \Downarrow w}{\mathscr{E} \;\rule{2pt}{10pt}\; \rhd(\varphi^{\sim}) \Downarrow v' \;\rule{2pt}{10pt}\; \lhd(\varphi^{\sim}) \Downarrow w' \;\rule{2pt}{10pt}\; t_0[v'/x] \Downarrow v \;\rule{2pt}{10pt}\; \lambda x.cx \Downarrow \lambda x.cx \;\rule{2pt}{10pt}\; u \Downarrow u' \;\rule{2pt}{10pt}\; \mathsf{let}\, u' \otimes w' \,\mathsf{be}\, z \otimes y \,\mathsf{in}\, t_1 \Downarrow w}$$

$$\dfrac{}{\mathscr{E} \;\rule{2pt}{10pt}\; \bar{c}(\rhd(\varphi^{\sim})) \Downarrow u' \;\rule{2pt}{10pt}\; \lhd(\varphi^{\sim}) \Downarrow w' \;\rule{2pt}{10pt}\; t_0[v'/x] \Downarrow v \;\rule{2pt}{10pt}\; \lambda x.cx \Downarrow \lambda x.cx \;\rule{2pt}{10pt}\; cu \Downarrow v' \;\rule{2pt}{10pt}\; \mathsf{let}\, u' \otimes w' \,\mathsf{be}\, z \otimes y \,\mathsf{in}\, t_1 \Downarrow w}$$

$$\dfrac{}{\mathscr{E} \;\rule{2pt}{10pt}\; \bar{c}(\rhd(\varphi^{\sim})) \Downarrow u' \;\rule{2pt}{10pt}\; \lhd(\varphi^{\sim}) \Downarrow w' \;\rule{2pt}{10pt}\; t_0[v'/x] \Downarrow v \;\rule{2pt}{10pt}\; (\lambda x.cx)u \Downarrow v' \;\rule{2pt}{10pt}\; \mathsf{let}\, u' \otimes w' \,\mathsf{be}\, z \otimes y \,\mathsf{in}\, t_1 \Downarrow w}$$

$$\text{eval-subst}\ \dfrac{}{\mathscr{E} \;\rule{2pt}{10pt}\; \bar{c}(\rhd(\varphi^{\sim})) \Downarrow u' \;\rule{2pt}{10pt}\; \lhd(\varphi^{\sim}) \Downarrow w' \;\rule{2pt}{10pt}\; t_0[(\lambda x.cx)u/x] \Downarrow v \;\rule{2pt}{10pt}\; \mathsf{let}\, u' \otimes w' \,\mathsf{be}\, z \otimes y \,\mathsf{in}\, t_1 \Downarrow w}$$

$$\dfrac{}{\mathscr{E} \;\rule{2pt}{10pt}\; \bar{c}(\rhd(\varphi^{\sim})) \otimes \lhd(\varphi^{\sim}) \Downarrow u' \otimes w' \;\rule{2pt}{10pt}\; t_0[(\lambda x.cx)u/x] \Downarrow v \;\rule{2pt}{10pt}\; \mathsf{let}\, u' \otimes w' \,\mathsf{be}\, z \otimes y \,\mathsf{in}\, t_1 \Downarrow w}$$

$$\dfrac{\dfrac{}{\lambda x.cx \Downarrow \lambda x.cx} \qquad \mathscr{E} \;\rule{2pt}{10pt}\; \bar{c}(\rhd(\varphi^{\sim})) \otimes \lhd(\varphi^{\sim}) \Downarrow u' \otimes w' \;\rule{2pt}{10pt}\; t_0[(\lambda x.cx)u/x] \Downarrow v \;\rule{2pt}{10pt}\; \mathsf{let}\, u' \otimes w' \,\mathsf{be}\, z \otimes y \,\mathsf{in}\, t_1 \Downarrow w}{\mathscr{E} \;\rule{2pt}{10pt}\; \lambda x.cx \Downarrow \lambda x.cx \;\rule{2pt}{10pt}\; \bar{c}(\rhd(\varphi^{\sim})) \otimes \lhd(\varphi^{\sim}) \Downarrow u' \otimes w' \;\rule{2pt}{10pt}\; t_0[(\lambda x.cx)u/x] \Downarrow v \;\rule{2pt}{10pt}\; \mathsf{let}\, u' \otimes w' \,\mathsf{be}\, z \otimes y \,\mathsf{in}\, t_1 \Downarrow w}$$

Figure 4: Proof of Lemma 4.7. The conclusion is identical to our goal up to abbreviations.

Now we can define the name restriction operator as an abbreviation:

$$\nu x : \varphi^{\sim}.t \equiv \mathsf{let} \rhd\lhd(\varphi^{\sim}) \,\mathsf{be}\, x_L \otimes x_R \,\mathsf{in}\, t$$

where we assume injections $x \mapsto x_L$ and $x \mapsto x_R$ whose images are disjoint.

Then, in addition to Theorem 4.1, more typing rules are available.

Proposition 4.5 (Process typing rule: name restriction)**.** *The following typing rule is admissible.*

$$\dfrac{\mathscr{O} \;\rule{2pt}{10pt}\; \Gamma, x : \varphi^{\sim}, y : \overline{\varphi^{\sim}} \vdash t : \psi}{\mathscr{O} \;\rule{2pt}{10pt}\; \Gamma \vdash \nu x : \varphi^{\sim}.t[x_L/x][x_R/y] : \psi}$$

Example 4.6 (Connecting processes using session realizers)**.** *Using the session realizers, we can connect the processes typed in Example 4.2. Indeed,*

$$\begin{aligned}&\vdash \nu(x : ?(!(\mathbf{1}\oplus\mathbf{1})\,\mathsf{end})\,\mathsf{end}).\nu(y : !(\mathbf{1}\oplus\mathbf{1})\,\mathsf{end}).\\ &\quad (x_R\langle y_L\rangle.\,y_R(z).\,\mathsf{ign}\,x_R, y_R\,\mathsf{in}\,z) \otimes \big(x_L(w').\,\mathsf{ign}\,x_L\,\mathsf{in}\,w'\langle \mathsf{inl}(*)\rangle.\,\mathsf{ign}\,w'\,\mathsf{in}\,*\big) : (\mathbf{1}\oplus\mathbf{1})\otimes\mathbf{1}\end{aligned}$$

is derivable.

Now we have to check the evaluation of the term in this example. For that we prepare a lemma.

Process Evaluation as Abbreviation. The intention of defining $x\langle u\rangle.\,t_0$ and $y(z).\,t_1$ is mimicking communication in process calculi. When we substitute x and y with session type realizers, these terms can actually communicate.

The next lemma can help us evaluate session realizers.

Lemma 4.7. *Let t_0 be a term containing a free variable x and t_1 be a term containing free variables y and z. The rule*

$$\dfrac{\mathscr{E} \;\rule{2pt}{10pt}\; \rhd(\varphi^{\sim}) \Downarrow v' \;\rule{2pt}{10pt}\; \lhd(\varphi^{\sim}) \Downarrow w' \;\rule{2pt}{10pt}\; t_0[v'/x] \Downarrow v \;\rule{2pt}{10pt}\; u \Downarrow u' \;\rule{2pt}{10pt}\; t_1[u'/z][w'/y] \Downarrow w}{\mathscr{E} \;\rule{2pt}{10pt}\; \rhd(!\psi\,\varphi^{\sim}) \Downarrow \lambda x.cx \;\rule{2pt}{10pt}\; \lhd(!\psi\,\varphi^{\sim}) \Downarrow u' \otimes w' \;\rule{2pt}{10pt}\; (x\langle u\rangle.\,t_0)[\lambda x.cx/x] \Downarrow v \;\rule{2pt}{10pt}\; (y(z).\,t_1)[u' \otimes w'/y] \Downarrow w}$$

is admissible under presence of the eval-subst rule.

Proof. By the derivation in Figure 4. ☐

Example 4.8 (Evaluation of communicating processes)**.** *Here is an example of evaluation using the eval-subst rule.*

$$
\dfrac{
  \dfrac{
    \dfrac{
      \dfrac{}{\triangleright(\mathsf{end}) \Downarrow * \parallel \triangleleft(\mathsf{end}) \Downarrow *}
      \quad
      \dfrac{
        \dfrac{\dfrac{}{* \Downarrow *}}{\mathsf{inl}(*) \Downarrow \mathsf{inl}(*)}
        \quad
        \dfrac{}{* \Downarrow *}
      }{\mathsf{inl}(*) \Downarrow \mathsf{inl}(*) \parallel * \Downarrow *}
      \quad
      \dfrac{\dfrac{}{* \Downarrow *}}{\mathsf{inl}(*) \Downarrow \mathsf{inl}(*)}
    }{
      \triangleright(\mathsf{end}) \Downarrow * \parallel \triangleleft(\mathsf{end}) \Downarrow * \parallel \mathsf{inl}(*) \Downarrow \mathsf{inl}(*) \parallel * \Downarrow * \parallel \mathsf{inl}(*) \Downarrow \mathsf{inl}(*)
    }
    \ \diamondsuit
  }{
    \begin{array}{c}
    \triangleright(!(\mathbf{1}\oplus\mathbf{1})\ \mathsf{end}) \Downarrow \lambda x.cx \parallel \triangleleft(!(\mathbf{1}\oplus\mathbf{1})\ \mathsf{end}) \Downarrow \mathsf{inl}(*) \otimes * \parallel \\
    (x_L\langle \mathsf{inl}(*)\rangle.\,\mathsf{ign}\, x_L \,\mathsf{in}\, *)[\lambda x.cx/x_L] \Downarrow * \parallel (x_R(z).\,\mathsf{ign}\, x_R \,\mathsf{in}\, z)[\mathsf{inl}(*) \otimes */x_R] \Downarrow \mathsf{inl}(*)
    \end{array}
  }
}{
  \dfrac{
    \begin{array}{c}
    \triangleright\triangleleft(!(\mathbf{1}\oplus\mathbf{1})\ \mathsf{end}) \Downarrow \lambda x.cx \otimes (\mathsf{inl}(*) \otimes *) \parallel \\
    (x_L\langle \mathsf{inl}(*)\rangle.\,\mathsf{ign}\, x_L \,\mathsf{in}\, *)[\lambda x.cx/x_L] \otimes (x_R(z).\,\mathsf{ign}\, x_R \,\mathsf{in}\, z)[\mathsf{inl}(*) \otimes */x_R] \Downarrow * \otimes \mathsf{inl}(*)
    \end{array}
  }{
    \nu(x : !(\mathbf{1}\oplus\mathbf{1})\ \mathsf{end}).\,(x_L\langle \mathsf{inl}(*)\rangle.\,\mathsf{ign}\, x_L \,\mathsf{in}\, *) \otimes (x_R(z).\,\mathsf{ign}\, x_R \,\mathsf{in}\, z) \Downarrow * \otimes \mathsf{inl}(*)
  }
}
$$

The step $\diamondsuit$ *uses Lemma 4.7.*

Proposition 4.9 (Copycatting)**.** *For any linear type* $\varphi^{\sim}$*, we can derive* $x : \varphi^{\sim}, y : \overline{\varphi^{\sim}} \vdash t : \mathbf{1}$ *for some term* t*.*

Proof. By induction on $\varphi^{\sim}$. ☐

4.1 Correctness with Respect to Abelian Logic

We compare the Amida calculus and Abelian logic and discover the fact that they are identical.

Theorem 4.10 (Completeness of the Amida Calculus for Abelian Logic)**.** *A formula is a theorem of Abelian logic if and only if the formula is inhabited in the Amida calculus.*

Theorem 4.11 (Soundness of the Amida Calculus for Abelian Logic)**.** *An inhabited type in the Amida calculus is a theorem of Abelian logic.*

Proof. Proofs appear in the author's thesis [16]. ☐

Now we can use some previous literature (Meyer and Slaney [26] and Casari [10]) to find some facts.

Corollary 4.12 (Division by two)**.** *If* $\varphi \otimes \varphi$ *is inhabited, so is* φ*.*

Corollary 4.13. *The law of excluded middle* $\varphi \otimes (\varphi \multimap \mathbf{1})$ *and prelinearity* $(\varphi \multimap \psi) \oplus (\psi \multimap \varphi)$ *are inhabited in the Amida calculus.*

5 Related Work

Metcalfe, Olivetti and Gabbay [24] gave a hypersequent calculus for Abelian logic and proved cut-elimination theorem for the hypersequent calculus. His formulation is different from ours because Metcalfe's system does not use conjunctive hypersequents. Shirahata [32] studied the multiplicative fragment of Abelian logic, which he called CMLL (compact multiplicative linear logic). He gave a categorical semantics for the proofs of a sequent calculus presentation of CMLL and then proved that the cut-elimination procedure of the sequent calculus preserves the semantics[2].

Kobayashi, Pierce and Turner [20] developed a type system for the π-calculus processes. Similarly to the type system presented here, their type system can specify types of communication contents through a name and how many times a name can be used. In some sense, that type system is more flexible than the one shown in this paper; their type system allows multiple uses of a channel, replicated processes

[2] Ciabattoni, Straßburger and Terui [11] already pointed out the fact that Shirahata [32] and Metcalfe, Olivetti and Gabbay [25] studied the same logic.

and weakening [20, Lemma 3.2]. In other respects, the type system in [20] is less expressible. That type system does not have lambda abstractions. Also, in contrast to our type system, it is impossible to substitute a free variable with a process in that type system.

Caires and Pfenning [9] provide a type system for a fragment of π-calculus. Their type system is, on some processes, more restrictive than the Amida calculus. For example, this escrowing process P below is not typable in their type system: $P = x\langle y\rangle.x(a).y\langle a\rangle.0$. The process first emits a channel y through channel x and then takes an input from x and outputs it to y. Following the informal description of types by Caires and Pfenning [9], the process P should be typable as $\vdash P :: x : (A \multimap \mathbf{1}) \otimes A$. However, such typing is not possible because $(A \multimap \mathbf{1}) \otimes A$ is not a theorem of dual intuitionistic linear logic (DILL), which their type system is based on. In our type system, the following sequent is derivable

$$x : (?A\,\mathsf{end})!A\,\mathsf{end} \vdash \nu(y : ?A\,\mathsf{end}).x\langle y_L\rangle.x(a).y_R\langle a\rangle.\,\mathsf{ign}\,x, y_L, y_R\,\mathsf{in}\,0 : \mathbf{1}$$

The resulting sequent indicates that the process is typable with one open channel x that first emits a channel that one can receive A from, and second sends a value of A. This concludes an example of a term which our type system can type but the type system in Caires and Pfenning [9] cannot. However, we cannot judge their type system to be too restrictive because we have not yet obtained both type safety and deadlock-freedom of Amida calculus at the same time.

On the other hand, the most complicated example in Caires and Pfenning [9], which involves a drink server, directs us towards a useful extension of the Amida calculus.

Example 5.1 (Drink server from Caires and Pfenning [9] in the Amida cal.)**.**

$$\begin{aligned} \mathit{ServerProto} &= (N \multimap I \multimap (N \otimes \mathbf{1}))\,\&\,(N \multimap (I \otimes \mathbf{1})) \\ &= (!N\;!I\;?N\;\mathsf{end})\,\&\,(!N\;?I\;\mathbf{1}) \end{aligned}$$

N stands for the type of strings and I stands for the type of integers, but following Caires and Pfenning [9], we identify both N and I with $\mathbf{1}$. *Below, SP abbreviates ServerProto. Here is the process of the server, which serves one client and terminates.*

$$\mathit{Serv} = \langle s(pn).s(cn).s\langle rc\rangle.\,\mathsf{ign}\,pn, cn, s\,\mathsf{in}\,0,\quad s(pn).s\langle pr\rangle.\,\mathsf{ign}\,s, pn\,\mathsf{in}\,0\rangle$$

We can derive a sequent $s : \overline{SP} \vdash \mathit{Serv} : \mathbf{1}$. *Here is one client:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash 0 : \mathbf{1}}}{s : \mathsf{end} \vdash \mathsf{ign}\,s\,\mathsf{in}\,0 : \mathbf{1}}}{s : \mathsf{end}, pr : I \vdash \mathsf{ign}\,pr, s\,\mathsf{in}\,0 : \mathbf{1}}}{s : ?I\,\mathsf{end} \vdash s(pr).\,\mathsf{ign}\,pr, s\,\mathsf{in}\,0 : \mathbf{1}} \quad \dfrac{}{\vdash tea : N}}{s : !N\,?I\,\mathsf{end} \vdash s\langle tea\rangle.s(pr).\,\mathsf{ign}\,pr, s\,\mathsf{in}\,0 : \mathbf{1}}}{s : \mathit{ServerProto} \vdash \mathsf{let}\,s\,\mathsf{be}\,\langle \_, s\rangle\,\mathsf{in}\,s\langle tea\rangle.s(pr).\,\mathsf{ign}\,pr, s\,\mathsf{in}\,0 : \mathbf{1}}$$

In words, the client first chooses the server's second protocol, which is price quoting, and asks the price of the tea, receives the price and terminates. We can combine the server with this client. However, since the Amida calculus lacks the exponential modality, Amida calculus cannot type any term with $!\mathit{ServerProto}$, *which the type system of Caires and Pfenning can [9]. In order to do that, we might want to tolerate inconsistency and add* μ *and* ν *operators from the modal* μ*-calculus, like Baelde [6] did, and express* $!\mathit{ServerProto}$ *as* $\nu X.(\mathit{SeverProto} \otimes X)$.

Wadler [35] gave a type system for a process calculus based on classical linear logic. Although the setting is classical, the idea is more or less the same as Caires and Pfenning [9]. Wadler's type system cannot type the escrowing process above.

Giunti and Vasconcelos [14] give a type system for π-calculus with the type preservation theorem. Their type system is extremely similar to our type system although the motivations are different; their motivation is process calculi while our motivation is computational interpretation of a logic. It will be worthwhile to compare their system with our type system closely.

6 Some Future Work and Conclusion

Implementation. Since Abelian logic is incompatible with contraction or weakening, straightforward implementation the Amida calculus on top of Haskell or OCaml would not be a good way to exploit the safety of the Amida calculus. One promising framework on which to implement the Amida calculus is linear ML[3], whose type system is based on linear logic. Another way is using the type level programming technique of Haskell. Imai, Yuen and Agusa [19] implemented session types on top of Haskell using the fact that Haskell types can contain arbitrary trees of symbols; thus we should be able to use the same technique to encode the types of Amida calculus in the Haskell types.

Adding Modalities. A tempting extension is to add modalities representing agents and then study the relationship with the multiparty session types [7, 18].

Cut-Elimination. It is easy to see that the prelinearity $(\varphi \multimap \psi) \oplus (\psi \multimap \varphi)$ does not have a cut-free proof. However, since there is a cut-free deduction system for Abelian logic [24], we consider it natural to expect the same property for a suitable extension of the Amida calculus.

Continuations. The eval-subst rule enables an evaluation $\overline{c}[C[cv]] \Downarrow v$, which reminds us of the call-with-current-continuation primitive [31] and shift/reset primitives [12, 2]. The appearance of these classical type system primitives is not surprising because Abelian logic validates $((p \multimap q) \multimap q) \multimap p$, which is a stronger form of the double negation elimination. Possibly we could use the technique of Asai and Kameyama [2] to analyze the Amida calculus with eval-subst rule.

Logic Programming. There are at least two ways to interpret logics computationally. One is proof reduction, which is represented by λ-calculi. The other is proof searching. We have investigated the Amida calculus, which embodies the proof reduction approach to the Amida axiom. Then what implication does The Amida axiom have in the proof searching approach? Let us cite an example from Kobayashi and Yonezawa [21, A.2]:

> Consumption of a message m by a process $m \multimap B$ is represented by the following deduction:
>
> $$(m \otimes (m \multimap B) \otimes C) \multimap (B \otimes C)$$
>
> where C can be considered as other processes and messages, or an environment.

[3]There are no publications but an implementation is available at `https://github.com/pikatchu/LinearML` .

Using the Amida axiom, the inverse

$$(B \otimes C) \multimap (m \otimes (m \multimap B)) \otimes C$$

is derivable. This suggests that the Amida axiom states that some computation is reversible in the context of proof searching. We suspect that this can be useful within the realm of reversible computation [34].

Conclusion. We found a new axiomatization of Abelian logic: the Amida axiom $(\varphi \multimap \psi) \otimes (\psi \multimap \varphi)$ on top of IMALL^{-}. The axiomatization has an application for encoding process calculi and session type systems. The encoding, which we name the Amida calculus, shows extra flexibility given by the new axiom. In the current form, the flexibility comes with the cost of convergence. Though there is a possible way to obtain convergence by adding a new evaluation rule, then, it is still under investigation whether type safety is preserved.

Acknowledgements. The author thanks Tadeusz Litak for encouragements and information on relevant research. The author also thanks Takeuti Izumi, who asked about changing a disjunction $\oplus$ into a conjunction $\otimes$ after the author talked about $(\varphi \multimap \psi) \oplus (\psi \multimap \varphi)$, a variant of which is used to model asynchronous communication in Hirai [15]. Anonymous refrees' careful comments and the workshop participants' direct questions improved the presentation of this paper.

References

[1] S. Abramsky (1993): *Computational Interpretations of Linear Logic*. *Theo. Comp. Sci.* 111(1-2), pp. 3–57, doi:10.1016/0304-3975(93)90181-R.

[2] K. Asai & Y. Kameyama (2007): *Polymorphic Delimited Continuations*. In: *APLAS '07*, *LNCS* 4807, Springer, pp. 239–254, doi:10.1007/978-3-540-76637-7_16.

[3] A. Avron (1991): *Hypersequents, logical consequence and intermediate logics for concurrency*. *Ann. Math. Artif. Intell.* 4, pp. 225–248, doi:10.1007/BF01531058.

[4] A. Avron (2000): *A Tableau System for Gödel–Dummett Logic Based on a Hypersequent Calculus*. In: *TABLEAUX '00*, *LNCS* 1847, Springer, pp. 98–111, doi:10.1007/10722086_11.

[5] M. Baaz, A. Ciabattoni & C. G. Fermüller (2003): *Hypersequent Calculi for Gödel Logics–a Survey*. *Journal of Logic and Computation* 13(6), pp. 835–861, doi:10.1093/logcom/13.6.835.

[6] D. Baelde (2012): *Least and Greatest Fixed Points in Linear Logic*. *ACM Trans. Comput. Logic* 13(1), pp. 2:1–2:44, doi:10.1145/2071368.2071370.

[7] A. Bejleri & N. Yoshida (2009): *Synchronous Multiparty Session Types*. *Electronic Notes in Theoretical Computer Science* 241(0), pp. 3–33, doi:10.1016/j.entcs.2009.06.002.

[8] R. Blute & P. Scott (2004): *Category Theory for Linear Logicians*. *Linear Logic in Computer Science* 316, pp. 3–65, doi:10.1017/CBO9780511550850.002.

[9] L. Caires & F. Pfenning (2010): *Session Types as Intuitionistic Linear Propositions*. In P. Gastin & F. Laroussinie, editors: *CONCUR 2010*, *LNCS* 6269, Springer, pp. 222–236, doi:10.1007/978-3-642-15375-4_16.

[10] E. Casari (1989): *Comparative Logics and Abelian l-groups*. In R. Ferro, C. Bonotto, S. Valentini & A. Zanardo, editors: *Logic Colloquium '88*, *Studies in logic and the foundations of mathematics* 127, North-Holland, pp. 161–190, doi:10.1016/S0049-237X(08)70269-6.

[11] A. Ciabattoni, L. Straßburger & K. Terui (2009): *Expanding the Realm of Systematic Proof Theory*. In: *CSL*, pp. 163–178, doi:10.1007/978-3-642-04027-6_14.

[12] O. Danvy & A. Filinski (1990): *Abstracting control*. In: *Proceedings of the 1990 ACM conference on LISP and functional programming*, LFP '90, ACM, pp. 151–160, doi:10.1145/91556.91622.

[13] N. Galatos, P. Jipsen, T. Kowalski & H. Ono (2007): *Residuated Lattices: An Algebraic Glimpse at Substructural Logics*, 1st edition. *Studies in Logic and The Foundations of Mathematics* 151, Elsevier.

[14] M. Giunti & V. T. Vasconcelos (2010): *A Linear Account of Session Types in the Pi Calculus*. In: *CONCUR 2010*, LNCS, Springer, pp. 432–446, doi:10.1007/978-3-642-15375-4_30.

[15] Y. Hirai (2012): *A Lambda Calculus for Gödel-Dummett Logic Capturing Waitfreedom*. In T. Schrijvers & P. Thiemann, editors: *Functional and Logic Programming*, *LNCS* 7294, Springer, pp. 151–165, doi:10.1007/978-3-642-29822-6_14.

[16] Y. Hirai (2013): *Hyper-Lambda Calculi*. Ph.D. thesis, Univ. of Tokyo.

[17] K. Honda (1993): *Types for Dyadic Interaction*. In: *CONCUR '93*, *LNCS* 715, Springer, pp. 509–523, doi:10.1007/3-540-57208-2_35.

[18] K. Honda, N. Yoshida & M. Carbone (2008): *Multiparty Asynchronous Session Types*. In: *POPL '08*, POPL '08, ACM, pp. 273–284, doi:10.1145/1328438.1328472.

[19] K. Imai, S. Yuen & K. Agusa (2010): *Session Type Inference in Haskell*. In: *PLACES*, pp. 74–91, doi:10.4204/EPTCS.69.6.

[20] N. Kobayashi, B. C. Pierce & D. N. Turner (1999): *Linearity and the pi-calculus*. *ACM Trans. Program. Lang. Syst.* 21(5), pp. 914–947, doi:10.1145/237721.237804.

[21] N. Kobayashi & A. Yonezawa (1995): *Higher-Order Concurrent Linear Logic Programming*. In: *Theory and Practice of Parallel Programming*, *LNCS* 907, Springer, pp. 137–166, doi:10.1007/BFb0026568.

[22] F. Lamarche (2008): *Proof Nets for Intuitionistic Linear Logic: essential nets*. Technical Report, Loria & INRIA-Lorraine.

[23] S. Marlow et al. (2010): *Haskell 2010 Language Report*. *Available online http://www.haskell.org/onlinereport/haskell2010*.

[24] G. Metcalfe (2006): *Proof Theory for Casari's Comparative Logics*. *Journal of Logic and Computation* 16(4), pp. 405–422, doi:10.1093/logcom/exl001.

[25] G. Metcalfe, N. Olivetti & D. Gabbay (2002): *Analytic Sequent Calculi for Abelian and Łukasiewicz Logics*. In U. Egly & C. Fermüller, editors: *TABLEAUX 2002*, *LNCS* 2381, Springer, pp. 191–205, doi:10.1007/3-540-45616-3_14.

[26] R. K. Meyer & J. K. Slaney (1989): *Abelian Logic (from A to Z)*. In G. Priest, R. Richard & J. Norman, editors: *Paraconsistent Logic: Essays on the Inconsistent*, chapter IX, Philosophia Verlag, pp. 245–288.

[27] R. Milner (1978): *A Theory of Type Polymorphism in Programming*. *Journal of Computer and System Sciences* 17(3), pp. 348–375, doi:10.1016/0022-0000(78)90014-4.

[28] R. Milner (1999): *Communicating and Mobile Systems: the pi-calculus*. Cambridge University Press.

[29] R. Milner, M. Tofte, R. Harper & D. MacQueen (1997): *The definition of Standard ML*. MIT press.

[30] A. S. Murawski & C. H. L. Ong (2003): *Exhausting Strategies, Joker Games and Full Completeness for IMLL with Unit*. *Theoretical Computer Science* 294(1-2), pp. 269–305, doi:10.1016/S0304-3975(01)00244-4.

[31] J. Rees & W. Clinger (1986): *Revised Report on the Algorithmic Language Scheme*. *SIGPLAN Not.* 21(12), pp. 37–79, doi:10.1145/15042.15043.

[32] M. Shirahata: *A Sequent Calculus for Compact Closed Categories*.

[33] K. Takeuchi, K. Honda & M. Kubo: *An Interaction-Based Language and its Typing System*. *PARLE'94 Parallel Architectures and Languages Europe*, pp. 398–413, doi:10.1007/3-540-58184-7_118.

[34] T. Toffoli (1980): *Reversible Computing*. In J. Bakker & J. Leeuwen, editors: *Automata, Languages and Programming*, *LNCS* 85, Springer, pp. 632–644, doi:10.1007/3-540-10003-2_104.

[35] P. Wadler (2012): *Propositions as Sessions*. In: *Proceedings of the 17th ACM SIGPLAN International Conference on Functional Programming*, ICFP '12, ACM, pp. 273–286, doi:10.1145/2398856.2364568.

A Categorical Considerations

One might want to ask whether we can model the logic with a symmetric monoidal closed category [8] with identified isomorphisms $\sigma_{ABCD}\colon (A \multimap B) \otimes (C \multimap D) \to (A \multimap D) \otimes (C \multimap B)$, with naturality conditions. Before considering equality among morphisms, we know there is a non-trivial example.

Example A.1 (Integer Model of [13, p. 107])**.** *The preorder formed by objects as integers and morphisms as the usual order among integers $\leq$ forms a symmetric monoidal closed category with swaps when we interpret $\otimes$ as addition and $m \multimap n$ as $n - m$.*

On the other hand, if we take another formulation requiring natural isomorphisms $\mathscr{C}(C,A) \times \mathscr{C}(D,B) \cong \mathscr{C}(D,A) \times \mathscr{C}(C,B)$, only singletons can be preorder models because $\langle id_A, id_B\rangle$ is mapped to $\langle f, g\rangle$ where $f\colon A \to B$ and $g\colon B \to A$ for any two objects A and B.

A straightforward reading of evaluation rules gives somewhat complicated equality conditions for morphisms. The condition says the following diagram commutes:

$$\begin{array}{ccc} (A \multimap B) \otimes (C \multimap D) & \xrightarrow{d_{ABCD}} & (A \otimes C) \multimap (B \otimes D) \\ \Big\downarrow{\scriptstyle \sigma_{ABCD}} & & \Big\downarrow{\scriptstyle id \multimap s_{B,D}} \\ (A \multimap D) \otimes (C \multimap B) & \xrightarrow{d_{ADCB}} & (A \otimes C) \multimap (D \otimes B) \end{array}$$

where d_{ABCD} is induced by adjunction between $\otimes$ and $\multimap$ from a morphism $((A \multimap B) \otimes (C \multimap D)) \otimes (A \otimes C) \to (B \otimes D)$, which is provided by symmetric monoidal closed properties.

Moreover, since $\varphi^* \equiv \varphi \multimap \mathbf{1}$ has derivable sequents $\mathbf{1} \vdash \varphi^* \otimes \varphi$ and $\varphi \otimes \varphi^* \vdash \mathbf{1}$, we can expect the semantics of φ^* to be the dual object of that of φ. Indeed, checking one of the coherence condition of compact closedness is evaluating the below typed term

$$\dfrac{\dfrac{\dfrac{t:\varphi \qquad \dfrac{}{\vdash *:\mathbf{1}}}{\vdash t:\varphi \,\boldsymbol{|}\, \vdash *:\mathbf{1}} \qquad \dfrac{\dfrac{\dfrac{}{x:\varphi \vdash x:\varphi} \qquad \dfrac{}{\vdash *:\mathbf{1}}}{x:\varphi \vdash cx:\mathbf{1} \,\boldsymbol{|}\, \vdash \bar{c}*:\varphi}}{x:\varphi \vdash cx:\mathbf{1} \,\boldsymbol{|}\, z:\mathbf{1} \vdash \mathsf{ign}\, z\, \mathsf{in}\, \bar{c}*:\varphi}}{\dfrac{\vdash t:\varphi \,\boldsymbol{|}\, x:\varphi \vdash cx:\mathbf{1} \,\boldsymbol{|}\, \vdash \mathsf{ign}\, *\, \mathsf{in}\, \bar{c}*:\varphi}{\dfrac{\vdash ct:\mathbf{1} \,\boldsymbol{|}\, \mathsf{ign}\, *\, \mathsf{in}\, \bar{c}*\varphi}{\vdash ct:\mathbf{1} \,\boldsymbol{|}\, y:\mathbf{1} \vdash \mathsf{ign}\, y\, \mathsf{in}\, \mathsf{ign}\, *\, \mathsf{in}\, \bar{c}*:\varphi}}}}{\vdash \mathsf{ign}\, ct\, \mathsf{in}\, \mathsf{ign}\, *\, \mathsf{in}\, \bar{c}*:\varphi}\ .$$

At least, if $t \Downarrow v$ is derivable, $\mathsf{ign}\, ct\, \mathsf{in}\, \mathsf{ign}\, *\, \mathsf{in}\, \bar{c}* \Downarrow v$ is also derivable.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{t \Downarrow v \qquad \dfrac{}{* \Downarrow *}}{t \Downarrow v \,\boldsymbol{|}\, * \Downarrow *}}{ct \Downarrow * \,\boldsymbol{|}\, \bar{c}* \Downarrow v} \qquad \dfrac{}{* \Downarrow *}}{ct \Downarrow * \,\boldsymbol{|}\, \mathsf{ign}\, *\, \mathsf{in}\, \bar{c}* \Downarrow v}}{\mathsf{ign}\, ct\, \mathsf{in}\, \mathsf{ign}\, *\, \mathsf{in}\, \bar{c}* \Downarrow v}}{}\ .$$

Showing the other direction is more involved because of eval-subst rule, but the author expects the case analysis on possible substitutions will succeed.

B Proof Nets

Toward better understanding the Amida calculus, a technique called proof nets seems promising. Generally, proof nets are straightforward for the multiplicative fragments but complicated when additive and exponential connectives are involved. Since the Amida axiom $(\varphi \multimap \psi) \otimes (\psi \multimap \varphi)$ does not contain additives $(\&, \oplus)$ or exponentials $(!, ?)$, we can focus on the multiplicative connectives ($\multimap$ and $\otimes$). The fragment is called *IMLL* (intuitionistic multiplicative fragment of linear logic). We also use the unit **1** for technical reasons. We first describe the IMLL proof nets and their properties. Then we add a new kind of edges called the Amida edges, which characterizes Abelian logic. The Amida links are named after the Amida lottery (also known as the Ghost Leg) for the syntactic similarity.

B.1 IMLL Essential Nets

The proof nets for intuitionistic linear logics are called *essential nets*. This subsection reviews some known results about the essential nets for intuitionistic multiplicative linear logic (IMLL). The exposition here is strongly influenced by Murawski and Ong [30].

We can translate a polarity $p \in \{+,-\}$ and an IMLL formula φ into a *polarized MLL formula* $\ulcorner\varphi\urcorner^p$ following Lamarche [22] and Murawski and Ong [30]. We omit the definition of polarized MLL formulae because the whole grammar is exposed in the translation below:

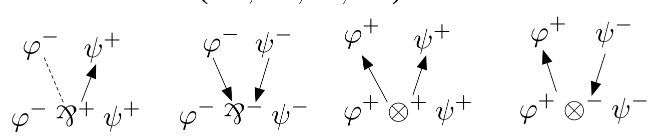
$$
\begin{aligned}
\ulcorner\mathbf{1}\urcorner^+ &= \mathbf{1}^+ & \ulcorner\mathbf{1}\urcorner^- &= \bot^- \\
\ulcorner X\urcorner^+ &= X^+ & \ulcorner X\urcorner^- &= X^- \\
\ulcorner\varphi \multimap \psi\urcorner^+ &= \ulcorner\varphi\urcorner^- \parr^+ \ulcorner\psi\urcorner^+ & \ulcorner\varphi \multimap \psi\urcorner^- &= \ulcorner\varphi\urcorner^+ \otimes^- \ulcorner\psi\urcorner^- \\
\ulcorner\varphi \otimes \psi\urcorner^+ &= \ulcorner\varphi\urcorner^+ \otimes^+ \ulcorner\psi\urcorner^+ & \ulcorner\varphi \otimes \psi\urcorner^- &= \ulcorner\varphi\urcorner^- \parr^- \ulcorner\psi\urcorner^- \quad .
\end{aligned}
$$

For example, the Amida axiom can be translated into a polarized MLL formula

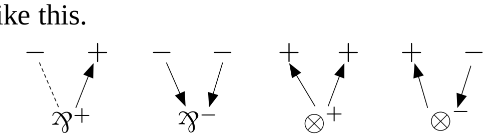
$$
\begin{aligned}
&\ulcorner(X \multimap Y) \otimes (Y \multimap X)\urcorner^+ \\
=&\ulcorner X \multimap Y\urcorner^+ \otimes^+ \ulcorner Y \multimap X\urcorner^+ \\
=&\left(\ulcorner X\urcorner^- \parr^+ \ulcorner Y\urcorner^+\right) \otimes^+ \left(\ulcorner Y\urcorner^- \parr^+ \ulcorner X\urcorner^+\right) \\
=&\left(X^- \parr^+ Y^+\right) \otimes^+ \left(Y^- \parr^+ X^+\right) \quad .
\end{aligned}
$$

The symbol $\parr$ is pronounced "parr."

Any polarized MLL formula can be translated further into a finite rooted tree containing these branches and polarized atomic formulae $(X^-, X^+, \mathbf{1}^+, \bot^-)$ at the leaves.

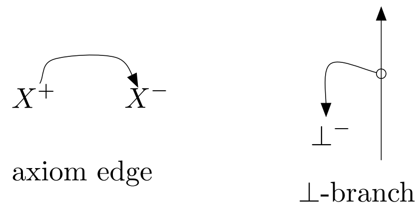


For brevity, we sometimes write only the top connectives of labeling formulae. In that case, these branching nodes above are denoted like this.

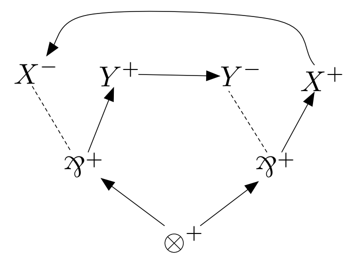


We call arrows with upward (resp. downward) signs *up-edges* (resp. *down-edges*). The *dashed child* of a $\parr^+$ node p is the node which the dashed line from p reaches. The branching nodes labeled by $\parr^+, \parr^-, \otimes^+$ and $\otimes^-$ are called *operator nodes*. A *path* follows solid edges according to the direction of the edges. Dashed edges are not directed and they are not contained in paths.

When we add axiom edges and $\bot$-branches (shown below) to the other operator nodes (shown above) we obtain an *essential net* of φ. Due to the arbitrariness of choosing axiom edges and $\bot$-branches, there are possibly multiple essential nets for a formula. Murawski and Ong [30] restricts the class of formulae to linearly balanced formulae so that the essential net is uniquely determined.



Example B.1 (An essential net of the Amida axiom)**.** *Here is one of the essential nets of the Amida axiom* $(X^- \parr^+ Y^+) \otimes^+ (Y^- \parr^+ X^+)$.



However, the essential net in Example B.1 is rejected by the following correctness criterion.

Definition B.2 (Correct essential nets)**.** *A correct essential net is an essential net satisfying all these conditions:*

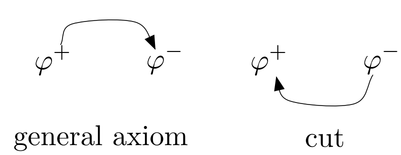
1. *Any node labeled with X^+ (resp. Y^-) is connected to a unique node labeled with X^- (resp. Y^+). Any leaf labeled with $\perp^-$ is connected to a $\perp$-branch. $\mathbf{1}^+$ is not connected to anything above itself;*
2. *the directed graph formed by up-edge, down-edge, axiom edges and $\perp$-branches is acyclic;*
3. *for every $\parr^+$-node p, every path from the root that reaches p's dashed child also passes through p.*

The essential net in Example B.1 is not correct for condition 3. Actually, the Amida axiom does not have any correct essential net. IMLL sequent calculus has the subformula property so that we can confirm that the Amida axiom is not provable in IMLL.

Theorem B.3 (Essential nets by [22, 30])**.** *An IMLL formula φ is provable in IMLL if and only if there exists a correct essential net of φ.*

Proof. The left to right is relatively easy. For the other way around, Lamarche [22] uses a common technique of decomposing an essential net from the bottom. Murawski and Ong [30] chose to reduce the problem to sequents of special forms called regular. □

Actually, Lamarche [22] also considers the cut rule (as well as additive operators and exponentials) in essential nets, thus we can include the following general axioms (as macros) and cuts (as primitives) and still use Theorem B.3:

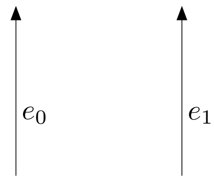


.

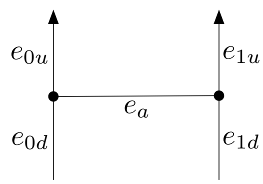
B.2 The Amida Nets

Definition B.4 (The Amida nets)**.** *For a hypersequent $\mathscr{H}$, Amida nets of $\mathscr{H}$ are inductively defined by the following three clauses:*

- *an essential net of $\mathscr{H}$ is an Amida net of $\mathscr{H}$;*
- *for an Amida net of $\mathscr{H}$ with two different*[4] *up-edges,*

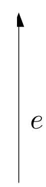


replacing these with

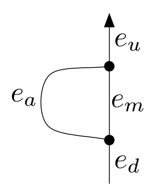


yields an Amida net of $\mathscr{H}$, where the above component has two paths $e_{0d}e_ae_{1u}$ and $e_{1d}e_ae_{0u}$;

- *for an Amida net of $\mathscr{H}$ with an up-edge,*



replacing this with

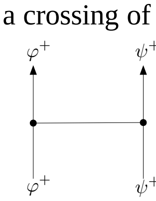


yields an Amida net of $\mathscr{H}$, where the above component has one finite path $e_de_ae_u$ and one infinite path $\cdots e_me_ae_me_a\cdots$.

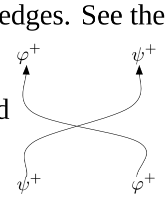
In these clauses, we call the edges labeled e_a the *Amida edges*.

Definition B.5 (Correct Amida nets)**.** *A correct Amida net is an Amida net satisfying the three conditions in Definition B.4.*

The Amida edge is not merely a crossing of up-edges. See the difference between



and

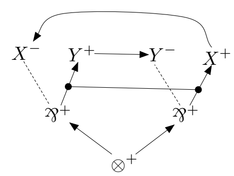


.

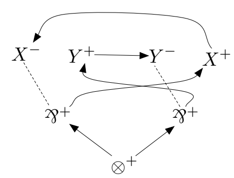
[4]The two edges can be connected by a new edge as long as they are different; their relative positions do not matter.

The difference is the labels at the bottom. Although Amida edges cross the paths, they do not transfer labels. This difference of labels makes Amida nets validate the Amida axiom.

Example B.6 (A correct Amida net for the Amida axiom)**.** *Here is a correct Amida net for the Amida axiom* $(X \multimap Y) \otimes (Y \multimap X)$.



In terms of the set of paths, the above Amida net is equivalent to the following correct essential net for $(X \multimap X) \otimes (Y \multimap Y)$.

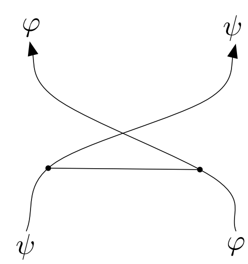


B.3 Soundness and Completeness of Amida nets

Theorem B.7 (Completeness of Amida nets)**.** *If a hypersequent* $\mathscr{H}$ *is derivable, there is a correct Amida net for* $\mathscr{H}$.

Proof. Inductively on hypersequent derivations. The Sync rule is translated into a crossing with an Amida edge:

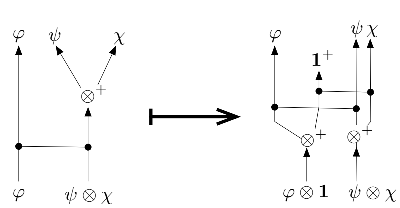
$$\frac{\Gamma \vdash \varphi \mid \Delta \vdash \psi}{\Gamma \vdash \psi \mid \Delta \vdash \varphi} \quad \mapsto$$



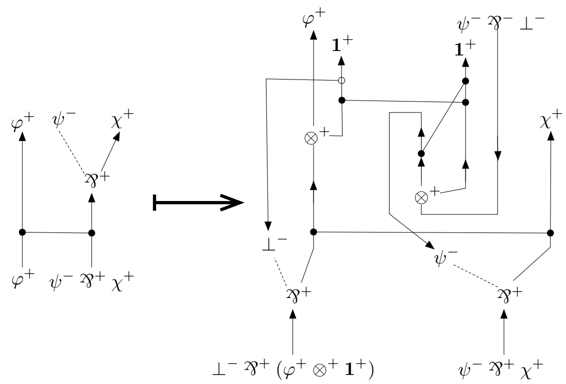
where the crossing exchanges the formulae and the Amida edge keeps the path connections vertically straight. □

Theorem B.8 (Soundness of Amida nets)**.** *If there is a correct Amida net for a hypersequent* $\mathscr{H}$*, then* $\mathscr{H}$ *is derivable.*

Proof. From a correct Amida net, first we move the Amida edges upwards until they are just below axiom edges[5]. The moves are as follows.



[5] The idea is similar to the most popular syntactic cut-elimination proofs.



These translations have two properties.

1. When the original (contained in a larger picture) is a correct Amida net, the translation (contained in the same larger picture) is also a correct Amida net.

2. The end nodes of the translation correspond to the end nodes of the original, and the corresponding end nodes have the same label (up to logical equivalence in IMLL). In case of $\otimes$ translation, φ and $\varphi \otimes \mathbf{1}$ are logically equivalent because $\mathbf{1}$ is the unit of $\otimes$. In case of $\parr$ translation, φ and $\bot \parr \varphi$ are logically equivalent because $\bot$ is the unit of $\parr$.

For checking the first condition, it is enough to follow the paths (crossing all Amida edges). For the second condition, it is enough to follow the vertical edges ignoring the Amida edges and $\bot$-branches.

The $\otimes$ move introduces Amida edges only above the branching rules. Although the $\parr$ move introduces an Amida edge below a branching rule, that branching rule is of $\otimes$ nature. Also, the $\parr$ move introduces an Amida edge below ψ-axiom link, which is actually a macro. So we have to continue applying the translation moves in the macro. However, since ψ is a strictly smaller subformula of $\psi \parr \chi$, this does not cause infinite recursion.

Then, by these translation moves, the whole Amida net is decomposed vertically into three layers. At the top, there is a layer with only axiom edges. In the middle, there is a layer with only vertical edges and Amida edges. At the bottom, there is a layer that contains only ordinary essential net nodes.

Since the middle layer is an Amida lottery, it defines a permutation. That permutation can be expressed as a product of transpositions, so that the original Amida lottery is equivalent to an encoding of a hypersequent derivation that consists of only Sync rules.

After we encode the top and the middle layer into a hypersequent derivation, encoding the bottom layer can be done in the same way as Lamarche's approach [22]. □

We wonder whether it is possible to add Amida edges to the IMALL$^-$ essential nets following Lamarche [22]. The additives are notoriously difficult for proof nets and we do not expect the combination of additive connectives and Amida edges can be treated in any straightforward way.