A Semantic Logical Relation for Termination of Intuitionistic Linear Logic Session Types

Tarakaram Gollamudi

Jules Jacobs
Cornell University

Yue Yao
Carnegie Mellon University

Stephanie Balzer
Carnegie Mellon University

ABSTRACT

Session types are behavioral types that specify protocols of message-passing concurrent systems. Typing not only delimits the range of possible messages that can be exchanged but also the order in which the exchange must happen. Session types enjoy a strong theoretical foundation, established by a Curry-Howard correspondence between the session-typed π-calculus and linear logic. Desirable language properties, such as deadlock freedom and termination, immediately hold true for languages building on this foundation, by virtue of cut elimination of the underlying logic. However, these results only generalize to such languages, if cut reductions are assumed to inform transitions. Yet existing session type languages typically define transitions explicitly by an operational semantics. Then termination must be proved by other means, for example by a logical relations argument.

This paper develops a *logical relation for termination* for *intuitionistic linear logic session types (ILLST)* and proves that well-typed ILLST programs are terminating (fundamental theorem). The logical relation is *semantic* and does not require terms to be well-typed. All results have been mechanized in the Coq proof assistant, amounting to the first mechanization of a logical relation for termination of session types. The paper introduces ILLST and the logical relation, along with a novel *rooted tree dynamics*.

Keywords: Intuitionistic linear logic session types, semantic logical relation

1 INTRODUCTION

Message passing is a prevalent concurrency paradigm, studied in research and employed by practical programming languages. Programs in this setting amount to a number of concurrently running *processes* which exchange messages along *channels*. To specify not only the types of messages but also their sequencing (*a.k.a.,* protocols), *session types* have been invented [4, 16, 17, 33]. Mechanization of session type metatheory has been receiving increasingly attention [5–7, 12–15, 18–20, 26, 27, 30]. This paper contributes the first *mechanization for termination* for *intuitionistic linear logic session types (ILLST)* [4] using the *logical relations* method [24, 25, 28, 29].

A distinguishing feature of our logical relation is its adoption of a constructive standpoint, typically referred to as *semantic* [8, 21, 31], permitting untyped terms to be inhabitants as long as they behave as prescribed by the logical relation. The fact that our logical relation is semantic sets it apart from existing logical relations for linear logic session types [1, 3, 9, 10, 22, 23]. Moreover, none of these prior results are mechanized.

Another distinguishing feature of our logical relation is its use of a novel *rooted tree dynamics*, exploiting its grounding in intuitionistic linear logic [4]. The rooted tree dynamics simplifies the handling of higher-order connectives $\otimes$ and $\multimap$ for the sending and receiving of channels along channels *resp.* These connectives induce tree transformations, where a child node becomes a sibling ($\otimes$) or a sibling node becomes a child ($\multimap$). Since the dynamics maintains the tree structure, these operations fall out (almost) for free. The rooted tree dynamics also facilitates local scoping of channel names and renders unnecessary the explicit naming of the root channel, referred to as the "providing" channel in the ILLST literature. Both aspects simplify the handling of names and reduce the number of substitutions and renamings.

2 ILLST

Next, we introduce intuitionistic linear logic session types (ILLST).

2.1 Type System

ILLST process terms are typed using the sequent

$$\Delta \vdash P : C$$

which reads as *"process term P provides a session of type C, given the typing of sessions provided along channel variables in Δ"*. Δ is a *linear* context and consists of a finite set of assumptions of the form $x_i : A_i$. Linearity enforces a treatment of channel variables as resources, forbidding duplication (contraction) and deletion (weakening). A crucial characteristic of session-typed processes is that processes change their type in accordance with the message exchange. As a result, a process' type always reflects the current protocol state. An overview of ILLST types and terms is given in Table 1.

As usual, the typing rules are given in a *sequent calculus*, where the conclusion denotes the protocol state before the message exchange and the premise the state after the exchange. The rules are standard [4, 32], so we just give the right and left rules for the higher-order connective $\otimes$, allowing us to send a channel:

$$(\otimes_R)\ \frac{\Delta \vdash P : A_2}{\Delta, y : A_1 \vdash \mathbf{pair}(y; P) : A_1 \otimes A_2} \qquad (\otimes_L)\ \frac{\Delta, x_1 : A_1, x : A_2 \vdash Q : C}{\Delta, x : A_1 \otimes A_2 \vdash \mathbf{split}(x; x_1.\, Q) : C}$$

Reading bottom-up, the right rule $\otimes_R$ types a provider of a session $A_1 \otimes A_2$, executing the term $\mathbf{pair}(y; P)$ (conclusion), indicating that the provider sends the channel y. After the send, the provider transitions to executing P, having lost channel y, and now providing session A_1 (premise). Conversely, the left rule $\otimes_L$ types a client of a session $A_1 \otimes A_2$ executing the term $\mathbf{split}(x; x_1.\, Q)$ (conclusion), indicating that the client receives a channel along x and binds it

Table 1: ILLST types and corresponding process terms

Type	Term	Description
$x{:}A_1 \oplus A_2$	$\mathbf{injL}(\mathbf{R});P$	send L(R), continue as P
	$\mathbf{case}(x;\ (Q_1,\ Q_2))$	receive L(R), continue as Q_i
$x{:}A_1 \otimes A_2$	$\mathbf{pair}(y;P)$	send channel $y{:}A_1$, continue as P
	$\mathbf{split}(x;\ x_1.\ Q)$	receive channel x_1, continue as Q
$x{:}A_1 \multimap A_2$	$\mathbf{rcv}(x.\ P)$	receive channel to x, continue as P
	$\mathbf{send}(x;\ x_1;\ Q)$	send channel $x_1{:}A_1$, continue as Q
$x{:}A_1 \,\&\, A_2$	$\mathbf{offer}(P_1\ P_2)$	receive L(R), continue as P_i
	$\mathbf{selectL}(\mathbf{R})(x;\ Q)$	send L(R), continue as Q
$x{:}1$	$\mathbf{close}$	terminate
	$\mathbf{wait}\ x;\ Q$	close channel x, continue as Q

Judgmental rules	Description
$\mathbf{spawn}\ [\overline{y}]\ (P;\ x.\ Q)$	spawn provider P with children $\overline{y}$, continue as Q
$\mathbf{fwd}\ (x)$	forward the channel x

to x_1. After the receive, the client transitions to executing Q, with channel x at type A_2 and the received channel $x_1{:}A_1$ (premise).

2.2 Rooted Tree Dynamics

Our development uses a novel dynamics, which makes explicit that ILLST-typed processes form *rooted* trees at runtime, unlike classical linear logic session types, which produce unrooted trees. The notion of rooted trees is conceptually present in prior work [2, 9, 11], but manifests in our dynamics, simplifying the handling of higher-order channels and permitting local scoping of channels.

We denote a rooted tree by the term $P \lhd \mu$, where P is the root process and μ its child forest, defined by the following grammar:

$$\Omega \ ::=\ P \lhd \mu \qquad \mu \ ::=\ \cdot \mid \mu \otimes a \hookrightarrow \Omega$$

The root process P refers to each of its child trees by a locally-bound channel name, *i.e.,* $\mu = a_1 \hookrightarrow \Omega_1 \otimes \ldots \otimes a_n \hookrightarrow \Omega_n$, where $\otimes$ denotes concurrent composition.

To provide a flavor of our dynamics, we give the rule for $\otimes$, using the judgment $P \lhd \mu \longmapsto P' \lhd \mu'$ for stepping $P \lhd \mu$ to $'P \lhd \mu'$:

$$\frac{b' \notin \mathrm{dom}\ (\mu)}{\begin{array}{c}\mathbf{split}(a;\ x_1.\ Q) \lhd \mu \otimes a \hookrightarrow (\mathbf{pair}(b;P) \lhd \mu_1 \otimes b \hookrightarrow \Omega') \\ \longmapsto [b'/x_1]Q \lhd \mu \otimes a \hookrightarrow (P \lhd \mu_1) \otimes b' \hookrightarrow \Omega'\end{array}}\ (\textsc{step-}\otimes)$$

The rule captures the scenario where the child process $\mathbf{pair}(b;P)$ sends the channel b, along with the sub-tree Ω' rooted at b, to its parent process $\mathbf{split}(a;\ x_1.\ Q)$. This sub-tree becomes the child of the parent $[b'/x_1]Q$ after the exchange, where the parent chooses a locally fresh channel b' to refer to it.

3 LOGICAL RELATION

To prove that well-typed ILLST process terms P yield terminating rooted process trees $P \lhd \mu$ at runtime, we define a unary logical relation $\Omega \in \mathcal{E}[\![A]\!]$ by structural induction on A. The index A is the session type provided by the root process P.[1] As usual, we find it convenient to define the relation by mutually recursive term and value interpretations. The term interpretation, $\Omega \in \mathcal{E}[\![A]\!]$, steps the configuration Ω internally until it cannot take any further internal steps, but its root process needs to communicate with its client, as prescribed by A. The value interpretation, $\Omega \in \mathcal{V}[\![A]\!]$, details this exchange, for each ILLST connective.

Definition 3.1 (Logical relation). Termination of a rooted tree Ω is defined as a unary logical relation $\Omega \in \mathcal{E}[\![A]\!]$, defined by structural induction on the providing session type A of the root process.

$$\begin{array}{lll}
\Omega \in \mathcal{E}[\![A]\!] & \text{iff} & \exists \Omega'.\Omega \mapsto^* \Omega' \wedge \Omega' \text{ done} \wedge \Omega' \in \mathcal{V}[\![A]\!] \\
\Omega \in \mathcal{V}[\![1]\!] & \text{iff} & \Omega = \mathbf{close} \lhd \emptyset \\
\Omega \in \mathcal{V}[\![A_1 \oplus A_2]\!] & \text{iff} & (\Omega = \mathbf{injL}; P \lhd \mu \wedge P \lhd \mu \in \mathcal{E}[\![A_1]\!]) \vee \\
 & & (\Omega = \mathbf{injR}; P \lhd \mu \wedge P \lhd \mu \in \mathcal{E}[\![A_2]\!]) \\
\Omega \in \mathcal{V}[\![A_1 \& A_2]\!] & \text{iff} & \Omega = \mathbf{offer}(P_1\ P_2) \lhd \mu \ \wedge \\
 & & \forall i \in \{1,2\}.P_i \lhd \mu \in \mathcal{E}[\![A_i]\!] \\
\Omega \in \mathcal{V}[\![A_1 \otimes A_2]\!] & \text{iff} & \Omega = \mathbf{pair}(b;P) \lhd \mu \otimes (b \hookrightarrow \Omega') \wedge \\
 & & \Omega' \in \mathcal{E}[\![A_1]\!] \ \wedge \ P \lhd \mu \in \mathcal{E}[\![A_2]\!] \\
\Omega \in \mathcal{V}[\![A_1 \multimap A_2]\!] & \text{iff} & \Omega = \mathbf{rcv}(x.\ P) \lhd \mu \wedge \\
 & & \forall\, \Omega' \in \mathcal{E}[\![A_1]\!], a \notin \mathrm{dom}\ (\mu). \\
 & & [a/x]P \lhd \mu \otimes (a \hookrightarrow \Omega') \in \mathcal{E}[\![A_2]\!]
\end{array}$$

To prove our main result, that well-typed ILLST process terms terminate (Thm. 3.4), we first need to define a closing substitution function, γ, mapping variables to either variables or channel symbols, and its lifting, $\hat{\gamma}$, to process terms.

Definition 3.2 (Substitution function). Given an *injective* substitution function γ, a variable context Δ, and a symbol context Σ, we write $\vDash_\Sigma \gamma : \Delta$ if for $x \notin \Delta$ we have $\gamma(x) = x$ and for $x \in \mathrm{dom}\ (\Delta)$ the following conditions are satisfied:

- γ bijectively maps dom (Δ) to dom (Σ)
- if $\Delta(x) = A$ then $\Sigma(\gamma(x)) = A$.

Definition 3.3. Given a symbol context Σ we write $\mu \in \mathcal{E}[\![\Sigma]\!]$ if dom (μ) = dom (Σ) and $\forall\, a_i \in \mathrm{dom}\ (\Sigma).\ \mu(a_i) \in \mathcal{E}[\![\Sigma(a_i)]\!]$.

We now can state our main result, the fundamental theorem of the logical relation:

Theorem 3.4 (FTLR). *If* $\Delta \vdash P : C$, *then* $\forall\, \gamma, \Sigma, \mu$ *such that* $\vDash_\Sigma \gamma : \Delta$ *and* $\mu \in \mathcal{E}[\![\Sigma]\!]$, *we have* $\hat{\gamma}(P) \lhd \mu \in \mathcal{E}[\![C]\!]$.

Condition $\vDash_\Sigma \gamma : \Delta$ guarantees that each variable in Δ is associated with a unique channel symbol of the same type, and condition $\mu \in \mathcal{E}[\![\Sigma]\!]$ guarantees that the trees rooted as these channel symbols are inhabitants of the logical relation.

Lastly, we prove the following *adequacy* results:

Corollary 3.5. *If* $\emptyset \vdash P : C$, *then there exists* Ω *such that* $P \lhd \emptyset \mapsto^* \Omega$ *and* Ω done.

Corollary 3.6. *If* $\emptyset \vdash P : 1$, *then* $P \lhd \emptyset \mapsto^* \mathbf{close} \lhd \emptyset$.

The first corollary states that a well-typed process term of an arbitrary type yields a "done" configuration, the second corollary states that a well-typed process term of type 1 yields a configuration consisting of a single root process without any children, attempting to close. All these results are mechanized in the Coq proof assistant.

ACKNOWLEDGMENTS

This work is supported in part by the Air Force Office of Scientific Research under award number FA9550-21-1-0385 (Tristan Nguyen, program manager). Any opinions, findings and conclusions or recommendations expressed here are those of the author(s) and do not necessarily reflect the views of the U.S. Department of Defense.

[1] Restricting observations to the providing channel is sufficient for termination, but generally insufficient for program equivalence [1, 9].

REFERENCES

[1] Stephanie Balzer, Farzaneh Derakhshan, Robert Harper, and Yue Yao. Logical relations for session-typed concurrency. *CoRR*, abs/2309.00192, 2023. URL: https://doi.org/10.48550/arXiv.2309.00192, arXiv:2309.00192, doi:10.48550/ARXIV.2309.00192.

[2] Stephanie Balzer, Bernardo Toninho, and Frank Pfenning. Manifest deadlock-freedom for shared session types. In *28th European Symposium on Programming (ESOP)*, volume 11423 of *Lecture Notes in Computer Science*, pages 611–639. Springer, 2019. doi:10.1007/978-3-030-17184-1_22.

[3] Luís Caires, Jorge A. Pérez, Frank Pfenning, and Bernardo Toninho. Behavioral polymorphism and parametricity in session-based communication. In *22nd European Symposium on Programming (ESOP)*, pages 330–349, 2013. doi:10.1007/978-3-642-37036-6_19.

[4] Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In *21th International Conference onf Concurrency Theory (CONCUR)*, volume 6269 of *Lecture Notes in Computer Science*, pages 222–236. Springer, 2010. doi:10.1007/978-3-642-15375-4_16.

[5] David Castro-Perez, Francisco Ferreira, Lorenzo Gheri, and Nobuko Yoshida. Zooid: A DSL for certified multiparty computation: From mechanised metatheory to certified multiparty processes. In *PLDI*, pages 237–251, 2021. doi:10.1145/3453483.3454041.

[6] David Castro-Perez, Francisco Ferreira, and Nobuko Yoshida. EMTST: engineering the meta-theory of session types. In *TACAS (2)*, volume 12079 of *LNCS*, pages 278–285, 2020. doi:10.1007/978-3-030-45237-7_17.

[7] Luca Ciccone and Luca Padovani. A dependently typed linear π-calculus in agda. In *PPDP*, pages 8:1–8:14, 2020. doi:10.1145/3414080.3414109.

[8] Robert L. Constable, Stuart F. Allen, Mark Bromley, Rance Cleaveland, J. F. Cremer, Robert Harper, Douglas J. Howe, Todd B. Knoblock, Nax Paul Mendler, Prakash Panangaden, James T. Sasaki, and Scott F. Smith. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice Hall, 1986. URL: http://dl.acm.org/citation.cfm?id=10510.

[9] Farzaneh Derakhshan, Stephanie Balzer, and Limin Jia. Session logical relations for noninterference. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–14. IEEE Computer Society, 2021. doi:10.1109/LICS52264.2021.9470654.

[10] Henry DeYoung, Frank Pfenning, and Klaas Pruiksma. Semi-axiomatic sequent calculus. In *5th International Conference on Formal Structures for Computation and Deduction (FSCD)*, volume 167 of *LIPIcs*, pages 29:1–29:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.FSCD.2020.29.

[11] Simon Fowler, Wen Kokke, Ornela Dardha, Sam Lindley, and J. Garrett Morris. Separating sessions smoothly. In *32nd International Conference on Concurrency Theory (CONCUR)*, volume 203 of *LIPIcs*, pages 36:1–36:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. URL: https://doi.org/10.4230/LIPIcs.CONCUR.2021.36, doi:10.4230/LIPICS.CONCUR.2021.36.

[12] Simon J. Gay, Peter Thiemann, and Vasco T. Vasconcelos. Duality of session types: The final cut. In *PLACES*, volume 314 of *EPTCS*, pages 23–33, 2020. doi:10.4204/EPTCS.314.3.

[13] Matthew A. Goto, Radha Jagadeesan, Alan Jeffrey, Corin Pitcher, and James Riely. An extensible approach to session polymorphism. *MSCS*, 26(3):465–509, 2016. doi:10.1017/S0960129514000231.

[14] Jonas Kastberg Hinrichsen, Jesper Bengtson, and Robbert Krebbers. Actris: Session-type based reasoning in separation logic. *PACMPL*, 4(POPL), December 2020. doi:10.1145/3371074.

[15] Jonas Kastberg Hinrichsen, Daniël Louwrink, Robbert Krebbers, and Jesper Bengtson. Machine-checked semantic session typing. In *CPP*, pages 178–198, 2021. doi:10.1145/3437992.3439914.

[16] Kohei Honda. Types for dyadic interaction. In *4th International Conference on Concurrency Theory (CONCUR)*, volume 715 of *Lecture Notes in Computer Science*, pages 509–523. Springer, 1993. doi:10.1007/3-540-57208-2_35.

[17] Kohei Honda, Vasco Thudichum Vasconcelos, and Makoto Kubo. Language primitives and type discipline for structured communication-based programming. In *7th European Symposium on Programming (ESOP)*, volume 1381 of *Lecture Notes in Computer Science*, pages 122–138. Springer, 1998. doi:10.1007/BFb0053567.

[18] Jules Jacobs. A self-dual distillation of session types. In *36th European Conference on Object-Oriented Programming (ECOOP)*, volume 222 of *LIPIcs*, pages 23:1–23:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. URL: https://doi.org/10.4230/LIPIcs.ECOOP.2022.23, doi:10.4230/LIPICS.ECOOP.2022.23.

[19] Jules Jacobs, Stephanie Balzer, and Robbert Krebbers. Connectivity graphs: a method for proving deadlock freedom based on separation logic. *Proc. ACM Program. Lang.*, 6(POPL):1–33, 2022. doi:10.1145/3498662.

[20] Jules Jacobs, Stephanie Balzer, and Robbert Krebbers. Multiparty GV: functional multiparty session types with certified deadlock freedom. *Proc. ACM Program. Lang.*, 6(ICFP):466–495, 2022. doi:10.1145/3547638.

[21] Per Martin-Löf. Constructive mathematics and computer programming. In *Logic, Methodology and Philosophy of Science VI*, volume 104 of *Studies in Logic and the Foundations of Mathematics*, pages 153–175. Elsevier, 1982. URL: https://www.sciencedirect.com/science/article/pii/S0049237X09701892, doi:https://doi.org/10.1016/S0049-237X(09)70189-2.

[22] Jorge A. Pérez, Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logical relations for session-based concurrency. In *21st European Symposium on Programming (ESOP)*, volume 7211 of *Lecture Notes in Computer Science*, pages 539–558. Springer, 2012. doi:10.1007/978-3-642-28869-2_27.

[23] Jorge A. Pérez, Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logical relations and observational equivalences for session-based concurrency. *Information and Computation*, 239:254–302, 2014. doi:10.1016/j.ic.2014.08.001.

[24] Andrew M. Pitts and Ian Stark. Operational reasoning for functions with local state. *Higher Order Operational Techniques in Semantics (HOOTS)*, pages 227–273, 1998.

[25] Gordon D. Plotkin. Lambda-definability and logical relations. Technical report, University of Edinburgh, 1973.

[26] Arjen Rouvoet, Casper Bach Poulsen, Robbert Krebbers, and Eelco Visser. Intrinsically-typed definitional interpreters for linear, session-typed languages. In *CPP*, pages 284–298, 2020. doi:10.1145/3372885.3373818.

[27] Alceste Scalas and Nobuko Yoshida. Less is more: multiparty session types revisited. *PACMPL*, 3(POPL):30:1–30:29, 2019. doi:10.1145/3290343.

[28] Richard Statman. Logical relations and the typed λ-calculus. *Information and Control*, 65(2/3):85–97, 1985. doi:10.1016/S0019-9958(85)80001-2.

[29] William W. Tait. Intensional interpretations of functionals of finite type I. *The Journal of Symbolic Logic*, 32(2):198–212, 1967. URL: http://www.jstor.org/stable/2271658.

[30] Peter Thiemann. Intrinsically-typed mechanized semantics for session types. In *PPDP*, pages 19:1–19:15, 2019. doi:10.1145/3354166.3354184.

[31] Amin Timany, Robbert Krebbers, Derek Dreyer, and Lars Birkedal. A logical approach to type soundness. *Journal of the ACM (JACM)*, 2024. To appear.

[32] Bernardo Toninho, Luís Caires, and Frank Pfenning. Higher-order processes, functions, and sessions: A monadic integration. In *22nd European Symposium on Programming (ESOP)*, volume 7792 of *Lecture Notes in Computer Science*, pages 350–369. Springer, 2013. doi:10.1007/978-3-642-37036-6_20.

[33] Philip Wadler. Propositions as sessions. In *ACM SIGPLAN International Conference on Functional Programming (ICFP)*, pages 273–286. ACM, 2012. doi:10.1145/2364527.2364568.