Type Congruence, Duality, and Iso-Recursive Session Types

Marco Giunti
University of Oxford, UK

Nobuko Yoshida
University of Oxford, UK

Session types [17, 31, 18] are an effective method to control the behaviour of software components that run in message-passing distributed systems. In most works on session types, recursive types follow an *equi-recursive* view [27] and represent infinite trees that are manipulated co-inductively. This representation does not have a direct counterpart in non-lazy programming languages, which typically resort to *iso-recursive* types [1, 27] that are manipulated inductively. Moreover, lazy evaluation of predicates on equi-recursive trees might not terminate, and is thus not effective for static program analysis. In practice, session types are embedded in non-lazy languages by encoding equi-recursive types; for instance, [20] defines infinite sequence of types as polymorphic lenses [9] by using OCaml *GADTs*.

Our proposal to overcome this problem consists in introducing a theory of *iso-recursive session types* relying on a type system that uses a novel notion of *type congruence* to relate the types of dual sessions. This contribution complements recent results [14] presenting a theory of iso-recursive multiparty session types. The paper [14] follows the bottom-up approach known as *generalised multiparty session types*, e.g. [21, 30, 29, 2, 12, 26, 16, 3], and decides deadlock-freedom without using global types. Differently from previous work, it considers iso-recursive types and computes the properties of session environments in the type system, instead of assessing these properties with model checkers (cf. [29]).

In this talk, we type check the parallel composition of sessions typed with folded and unfolded dual iso-recursive session types by means of a *type congruence* on types. We *mechanise* type congruence in Coq [6] without resorting to coinductive types, and use the proof assistant to establish two key properties of type congruence: *closure* under (i) session type *duality* and (ii) *labelled transitions of types*.

Iso-recursive sessions. The syntax of types and processes is below. We consider contractive [27] *iso-recursive* types of the form $\mu X.T$ where $\mu X.T$ and its unfolding are not equal, but isomorphic. We stress that types have a *finite representation* rather than abstract infinite trees (cf. equi-recursive types).

$\mathscr{S} \ni S$	$:= \mathsf{nat} \mid \mathsf{int} \mid \mathsf{str} \mid \mathsf{bool} \mid \mathsf{unit}$	*Sorts*
$\mathscr{T} \ni T$	$:= \mathtt{r}!l(S).T \mid \mathtt{r}?l(S).T \mid T+T \mid \mathsf{end} \mid \mu X.T \mid X \mid \bot$	*Types*
$\mathscr{P} \ni P$	$:= \mathtt{r}!l\langle e\rangle.P \mid \mathtt{r}?l(x).P \mid P+P \mid \mu\chi.P \mid \chi \mid \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \mid \mathbf{0}$	*Processes*
$\mathbb{M} \ni \mathscr{M}$	$:= \mathtt{r} \triangleleft P \mid (\mathtt{p} \triangleleft P \parallel \overline{\mathtt{p}} \triangleleft Q)$	*Sessions*

We use $\mathtt{p},\mathtt{q},\mathtt{r}$ to range over *participants*, l to range over *labels*, X to range over *type variables*, e to range over expressions, v to range over *values*, x to range over *variables*, and χ to range over *process variables*. We assume an involution $\overline{\mathtt{r}}$. Type $\bot$ is reserved for closing the composition of two threads [18]. The constructor μ is a *binder* in types and processes; the remaining binder for processes is input. *Free* variables are those that are not bound; *closed* terms are those without free variables. We assume the *substitution* of free occurrences of a type variable X in a type T_1 with a closed type T_2, written $T_1\{T_2/X\}$; similarly, we assume the process substitution $P_1\{P_2/\chi\}$ whenever P_2 is closed. The labelled transition semantics of sessions $\mathscr{M}$ rely on structural equivalence to swap the order of threads; most rules are standard [15]. We note that recursive sessions $\mathtt{r} \triangleleft \mu\chi.P$ are silently unfolded to $\mathtt{r} \triangleleft P\{\mu\chi.P/\chi\}$.

Type congruence. Following [23, Chapter 4], we let type congruence be the union of all symmetric equivalences. We devise a notion of type equivalence that is tailored at equating folded and unfolded *well-formed* types (WF(T) [15, App. A]). A relation $\mathscr{R} \subseteq \mathscr{T} \times \mathscr{T}$ is a *type equivalence* whenever $T_1 \mathscr{R} T_2$ and

To appear in EPTCS.

(i) $T_1 = \mu X.U_1$ and $T_2 = \mu X.U_2$ imply $U_1 \mathscr{R} U_2$ and $U_1\{\mu X.U_1/X\} \mathscr{R} T_2$ and $T_1 \mathscr{R} U_2\{\mu X.U_2/X\}$

(ii) $T_1 = \mu X.U_1$ and $T_2 \neq \mu X.U_2$ imply $U_1\{\mu X.U_1/X\} \mathscr{R} T_2$

(iii) $T_1 \neq \mu X.U_1$ and $T_2 = \mu X.U_2$ imply $T_1 \mathscr{R} U_2\{\mu X.U_2/X\}$

The remaining cases are homomorphic. A relation $\mathscr{R} \subseteq \mathscr{T} \times \mathscr{T}$ is a *structural equivalence* whenever it is (i) a type equivalence and (ii) symmetric. *Type congruence*, noted $\equiv \subseteq \mathscr{T} \times \mathscr{T}$, is the union of all structural equivalences. The *mechanisation* of types in Coq takes advantage of iso-recursion and does not rely on the command `CoInductive` (cf. [7]). The mechanisation of type congruence follows.

```
Notation X £ T := (substT T X (typ_mu X T)) (at level 40).
Definition equiv R := ∀ t1 t2, R t1 t2 → match t1, t2 with
  | typ_mu X1 U1, typ_mu X2 U2 ⇒ R U1 U2 ∧ R (X1 £ U1) t2 ∧ R t1 (X2 £ U2)
  | typ_mu X U, _ ⇒ R (X £ U) t2 | _, typ_mu X U ⇒ R t1 (X £ U) | ··· end.
Definition struct_equiv R := equiv R ∧ symmetric typ R. Definition typ_scongr := union_st typ typ struct_equiv.
Notation "T1 == T2" := (typ_scongr T1 T2) (at level 40). Check equiv_scongr. equiv_scongr : equiv typ_scongr
Check exist_se. exist_se : ∀ (R : typ → typ → Prop) (t1 t2 : typ), struct_equiv R → R t1 t2 → t1 == t2
Ltac prove_scongr R := match goal with | ⊢ ?W1 == ?W2 ⇒ eapply (exist_se R); eauto end.
```

Type system and subject reduction. Let Γ map variables to sorts and process variables to types. We consider a type system for processes, noted $\Gamma \vdash P \colon T$, and a type system for sessions, noted $\Gamma \Vdash \mathscr{M} : T$. The system $\Vdash$ only invokes the system $\vdash$ with *well-formed types*. The process rules depicting the essence of iso-recursive session types are the following:

$$\text{T-REC}\;\frac{\Gamma, \chi \colon \mu X.T \vdash P \colon T\{\mu X.T/X\}}{\Gamma \vdash \mu\chi.P \colon \mu X.T} \qquad \text{T-VAR}\;\frac{\Gamma(\chi) = \mu X.T}{\Gamma \vdash \chi \colon \mu X.T}$$

Note the difference with equi-recursive systems [11], where T-REC requires that the type of $\mu\chi.P$ and of P are equal, because $\mu X.T$ and $T\{\mu X.T/X\}$ are equal. The rule for typing a session composition is defined as follows, where *type duality*, noted $\overline{T}$, is syntactic rather than coinductive [10, 13].

$$\Gamma \vdash P \colon T_1 \text{ and } \Gamma \vdash Q \colon T_2 \text{ and } \mathrm{WF}(T_1) \text{ and } \mathrm{WF}(T_2) \text{ and } \overline{T_1} \equiv T_2 \text{ imply } \Gamma \Vdash \mathtt{p} \lhd P \parallel \overline{\mathtt{p}} \lhd Q \colon \bot$$

The proof of subject reduction relies on a transition system of types, noted $T \xrightarrow{\alpha} T'$. Recursive types $\mu X.T$ silently reach $T\{\mu X.T/X\}$. The proof stands on two mechanised results on type congruence.

Lemma *Let $\alpha_1 \mathscr{D} \alpha_2$ be action duality.*

(i) *$T_1 \equiv T_2$ implies $\overline{T_1} \equiv \overline{T}_2$*

(ii) *$WF(T_1)$ and $WF(T_2)$ and $T_1 \equiv \overline{T_2}$ and $\alpha_1 \mathscr{D} \alpha_2$ and $T_1 \xrightarrow{\alpha_1} U_1$ and $T_2 \xrightarrow{\alpha_2} U_2$ imply $U_1 \equiv \overline{U_2}$*

The substitution of recursive processes at execution time preserves typing; the result is mechanised.

Lemma (Process Substitution) *If $\Gamma, \chi : T \vdash P \colon U$ and $\Gamma \vdash Q \colon T$ then $\Gamma \vdash P\{Q/\chi\} \colon U$.*

Subject reduction also relies on value substitution, and on type preservation of structural congruence; the last result is mechanised. Case (i) occurs when (2) is inferred from the reduction of an if-then-else process. The mechanisation of subject reduction is ongoing. The proof of the theorem is closed; it is left to prove intermediate results used in the proof on type well-formedness and on value substitution.

Theorem (Subject Reduction) *Let $\mathscr{M}$ be a closed session and assume (1) $\Gamma \Vdash \mathscr{M} : T$ and (2) $\mathscr{M} \xrightarrow{\alpha} \mathscr{M}'$. We have (i) $\Gamma \Vdash \mathscr{M}' : T$ or (ii) $T \xrightarrow{\alpha} T'$ and $\Gamma \Vdash \mathscr{M}' : T'$.*

Related Work. Only few works follow an iso-recursive approach to session types. Recently [14, 15] we introduced iso-recursive multiparty session types and automatically verified [25, 5, 8] the properties of a function deciding the soundness of compositions. [19] studies iso-recursive and equi-recursive subtyping for binary session propositions with least and greatest fixed points [4, 32]. Many recent papers [33, 34, 35, 36, 28, 24, 22] present iso-recursive variants of the λ-calculus, following the seminal work on Amber rules [1]. Pierce [27] discusses the differences between iso-recursive and equi-recursive types.

References

[1] Martín Abadi & Luca Cardelli (1996): *A Theory of Objects*. Monographs in Computer Science, Springer, doi:10.1007/978-1-4419-8598-9.

[2] Adam D. Barwell, Alceste Scalas, Nobuko Yoshida & Fangyi Zhou (2022): *Generalised Multiparty Session Types with Crash-Stop Failures*. In Bartek Klin, Slawomir Lasota & Anca Muscholl, editors: *33rd International Conference on Concurrency Theory, CONCUR 2022, September 12-16, 2022, Warsaw, Poland*, *LIPIcs* 243, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 35:1–35:25, doi:10.4230/LIPICS.CONCUR.2022.35.

[3] Matthew Alan Le Brun & Ornela Dardha (2023): *MAGπ: Types for Failure-Prone Communication*. In Thomas Wies, editor: *Programming Languages and Systems - 32nd European Symposium on Programming, ESOP 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings*, *Lecture Notes in Computer Science* 13990, Springer, pp. 363–391, doi:10.1007/978-3-031-30044-8_14.

[4] Luís Caires & Frank Pfenning (2010): *Session Types as Intuitionistic Linear Propositions*. In Paul Gastin & François Laroussinie, editors: *CONCUR 2010 - Concurrency Theory, 21th International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010. Proceedings*, *Lecture Notes in Computer Science* 6269, Springer, pp. 222–236, doi:10.1007/978-3-642-15375-4_16.

[5] Arthur Charguéraud, Jean-Christophe Filliâtre, Cláudio Lourenço & Mário Pereira (2019): *GOSPEL - Providing OCaml with a Formal Specification Language*. In Maurice H. ter Beek, Annabelle McIver & José N. Oliveira, editors: *Formal Methods - The Next 30 Years - Third World Congress, FM 2019, Porto, Portugal, October 7-11, 2019, Proceedings*, *Lecture Notes in Computer Science* 11800, Springer, pp. 484–501, doi:10.1007/978-3-030-30942-8_29.

[6] Coq development team: *Reference manual*. Available at `https://coq.inria.fr/doc/V8.20.0/refman/`.

[7] Burak Ekici & Nobuko Yoshida (2024): *Completeness of Asynchronous Session Tree Subtyping in Coq*. In Yves Bertot, Temur Kutsia & Michael Norrish, editors: *15th International Conference on Interactive Theorem Proving, ITP 2024, September 9-14, 2024, Tbilisi, Georgia*, *LIPIcs* 309, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 13:1–13:20, doi:10.4230/LIPICS.ITP.2024.13.

[8] Jean-Christophe Filliâtre & Andrei Paskevich (2013): *Why3 - Where Programs Meet Provers*. In Matthias Felleisen & Philippa Gardner, editors: *Programming Languages and Systems - 22nd European Symposium on Programming, ESOP 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, *Lecture Notes in Computer Science* 7792, Springer, pp. 125–128, doi:10.1007/978-3-642-37036-6_8.

[9] J. Nathan Foster, Michael B. Greenwald, Jonathan T. Moore, Benjamin C. Pierce & Alan Schmitt (2007): *Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem*. *ACM Trans. Program. Lang. Syst.* 29(3), p. 17, doi:10.1145/1232420.1232424.

[10] Simon J. Gay & Malcolm Hole (2005): *Subtyping for session types in the pi calculus*. *Acta Informatica* 42(2-3), pp. 191–225, doi:10.1007/S00236-005-0177-Z.

[11] Silvia Ghilezan, Svetlana Jaksic, Jovanka Pantovic, Alceste Scalas & Nobuko Yoshida (2019): *Precise subtyping for synchronous multiparty sessions*. *J. Log. Algebraic Methods Program.* 104, pp. 127–173, doi:10.1016/J.JLAMP.2018.12.002.

[12] Silvia Ghilezan, Jovanka Pantović, Ivan Prokić, Alceste Scalas & Nobuko Yoshida (2023): *Precise Subtyping for Asynchronous Multiparty Sessions*. *ACM Trans. Comput. Logic* 24(2), doi:10.1145/3568422.

[13] Marco Giunti & Vasco Thudichum Vasconcelos (2016): *Linearity, session types and the Pi calculus*. *Math. Struct. Comput. Sci.* 26(2), pp. 206–237, doi:10.1017/S0960129514000176.

[14] Marco Giunti & Nobuko Yoshida (2025): *Iso-Recursive Multiparty Sessions and their Automated Verification*. In: *34th European Symposium on Programming (ESOP 2025)*, Lecture Notes in Computer Science, Springer. To appear: full version [15].

[15] Marco Giunti & Nobuko Yoshida (2025): *Iso-Recursive Multiparty Sessions and their Automated Verification – Technical Report*, doi:10.48550/ARXIV.2501.17778. arXiv:2501.17778.

[16] Paul Harvey, Simon Fowler, Ornela Dardha & Simon J. Gay (2021): *Multiparty Session Types for Safe Runtime Adaptation in an Actor Language*. In Anders Møller & Manu Sridharan, editors: *35th European Conference on Object-Oriented Programming, ECOOP 2021, July 11-17, 2021, Aarhus, Denmark (Virtual Conference)*, *LIPIcs* 194, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 10:1–10:30, doi:10.4230/LIPIcs.ECOOP.2021.10.

[17] Kohei Honda (1993): *Types for Dyadic Interaction*. In Eike Best, editor: *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings*, *Lecture Notes in Computer Science* 715, Springer, pp. 509–523, doi:10.1007/3-540-57208-2_35.

[18] Kohei Honda, Vasco Thudichum Vasconcelos & Makoto Kubo (1998): *Language Primitives and Type Discipline for Structured Communication-Based Programming*. In Chris Hankin, editor: *Programming Languages and Systems - ESOP'98, 7th European Symposium on Programming, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'98, Lisbon, Portugal, March 28 - April 4, 1998, Proceedings*, *Lecture Notes in Computer Science* 1381, Springer, pp. 122–138, doi:10.1007/BFB0053567.

[19] Ross Horne & Luca Padovani (2024): *A logical account of subtyping for session types*. *J. Log. Algebraic Methods Program.* 141, p. 100986, doi:10.1016/J.JLAMP.2024.100986.

[20] Keigo Imai, Rumyana Neykova, Nobuko Yoshida & Shoji Yuen (2020): *Multiparty Session Programming With Global Protocol Combinators*. In Robert Hirschfeld & Tobias Pape, editors: *34th European Conference on Object-Oriented Programming, ECOOP 2020, November 15-17, 2020, Berlin, Germany (Virtual Conference)*, *LIPIcs* 166, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 9:1–9:30, doi:10.4230/LIPICS.ECOOP.2020.9.

[21] Julien Lange, Nicholas Ng, Bernardo Toninho & Nobuko Yoshida (2018): *A static verification framework for message passing in Go using behavioural types*. In Michel Chaudron, Ivica Crnkovic, Marsha Chechik & Mark Harman, editors: *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, ACM, pp. 1137–1148, doi:10.1145/3180155.3180157.

[22] Jay Ligatti, Jeremy Blackburn & Michael Nachtigal (2017): *On Subtyping-Relation Completeness, with an Application to Iso-Recursive Types*. *ACM Trans. Program. Lang. Syst.* 39(1), pp. 4:1–4:36, doi:10.1145/2994596.

[23] Robin Milner (1989): *Communication and concurrency*. PHI Series in computer science, Prentice Hall.

[24] Marco Patrignani, Eric Mark Martin & Dominique Devriese (2021): *On the semantic expressiveness of recursive types*. *Proc. ACM Program. Lang.* 5(POPL), pp. 1–29, doi:10.1145/3434302.

[25] Mário Pereira & António Ravara (2021): *Cameleer: A Deductive Verification Tool for OCaml*. In Alexandra Silva & K. Rustan M. Leino, editors: *Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part II*, *Lecture Notes in Computer Science* 12760, Springer, pp. 677–689, doi:10.1007/978-3-030-81688-9_31.

[26] Kirstin Peters & Nobuko Yoshida (2024): *Separation and Encodability in Mixed Choice Multiparty Sessions*. In Pawel Sobocinski, Ugo Dal Lago & Javier Esparza, editors: *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2024, Tallinn, Estonia, July 8-11, 2024*, ACM, pp. 62:1–62:15, doi:10.1145/3661814.3662085.

[27] Benjamin C. Pierce (2002): *Types and programming languages*. MIT Press.

[28] Andreas Rossberg (2023): *Mutually Iso-Recursive Subtyping*. *Proc. ACM Program. Lang.* 7(OOPSLA2), pp. 347–373, doi:10.1145/3622809.

[29] Alceste Scalas & Nobuko Yoshida (2019): *Less is more: multiparty session types revisited*. *Proc. ACM Program. Lang.* 3(POPL), pp. 30:1–30:29, doi:10.1145/3290343.

[30] Alceste Scalas, Nobuko Yoshida & Elias Benussi (2019): *Verifying message-passing programs with dependent behavioural types*. In Kathryn S. McKinley & Kathleen Fisher, editors: *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, Phoenix, AZ, USA, June 22-26, 2019*, ACM, pp. 502–516, doi:10.1145/3314221.3322484.

[31] Kaku Takeuchi, Kohei Honda & Makoto Kubo (1994): *An Interaction-based Language and its Typing System*. In Constantine Halatsis, Dimitris G. Maritsas, George Philokyprou & Sergios Theodoridis, editors: *PARLE '94: Parallel Architectures and Languages Europe, 6th International PARLE Conference, Athens, Greece, July 4-8, 1994, Proceedings*, *Lecture Notes in Computer Science* 817, Springer, pp. 398–413, doi:10.1007/3-540-58184-7_118.

[32] Philip Wadler (2014): *Propositions as sessions*. *J. Funct. Program.* 24(2-3), pp. 384–418, doi:10.1017/S095679681400001X.

[33] Litao Zhou & Bruno C. d. S. Oliveira (2025): *QuickSub: Efficient Iso-Recursive Subtyping*. *Proc. ACM Program. Lang.* 9(POPL), doi:10.1145/3704869.

[34] Litao Zhou, Qianyong Wan & Bruno C. d. S. Oliveira (2024): *Full Iso-Recursive Types*. *Proc. ACM Program. Lang.* 8(OOPSLA2), pp. 192–221, doi:10.1145/3689718.

[35] Litao Zhou, Yaoda Zhou & Bruno C. d. S. Oliveira (2023): *Recursive Subtyping for All*. *Proc. ACM Program. Lang.* 7(POPL), pp. 1396–1425, doi:10.1145/3571241.

[36] Yaoda Zhou, Jinxu Zhao & Bruno C. d. S. Oliveira (2022): *Revisiting Iso-Recursive Subtyping*. *ACM Trans. Program. Lang. Syst.* 44(4), pp. 24:1–24:54, doi:10.1145/3549537.