A Decade of Dependent Session Types

Bernardo Toninho
NOVA School of Science and Technology and NOVA-LINCS
Portugal

Luís Caires
NOVA School of Science and Technology and NOVA-LINCS
Portugal

Frank Pfenning
Carnegie Mellon University
USA

CCS Concepts: • **Theory of computation → Linear logic**; **Type theory**; • **Software and its engineering → Concurrent programming languages**.

ACM Reference Format:
Bernardo Toninho, Luís Caires, and Frank Pfenning. 2021. A Decade of Dependent Session Types. In *23rd International Symposium on Principles and Practice of Declarative Programming (PPDP 2021), September 6–8, 2021, Tallinn, Estonia.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3479394.3479398

We begin this brief retrospective of our work by thanking the PPDP Steering Committee for awarding the "PPDP 10 Year Most Influential Paper Award" to our paper on dependent session types [22]. This abstract gives an account of the context that led to our 2011 paper, summarizes its key contributions and gives an (incomplete) account of the works that followed it and the remaining challenges in this space.

1 Context

Session types date back to the seminal work of Honda et al. [14] and are, in their essence, a mechanism to statically ensure correctness properties of communication in message-passing programs. Session types achieve this by taking a protocols-as-types view of protocol compliance. By viewing communication protocols as the types of communication channels, it becomes possible to statically ensure that communication is protocol compliant (*session fidelity* in the literature), even in the presence of so-called higher-order communication (i.e. sending channels along channels). A key ingredient of session types is type *duality*, which captures the fact that communication between peers proceeds in tandem. When one party sends, the other receives, when one offers a choice, the other chooses. Session type duality embodies this principle, ensuring that processes that communicate along the same channel must follow dual types, thus achieving communication safety. Session types (in the sense of [14]) also ensure deadlock-freedom (or progress) for processes communicating along a single session channel.

In the sense above, session types have two key limitations: progress guarantees only apply in a limited setting (i.e. two processes communicating exclusively on a single channel without higher-order communication) and, types only describe simple properties of communication, rather than properties of exchanged data or how data affects communication. To address the former, a notion of *multiparty* session types [15] was introduced, which can progress in the presence of multiple communicating peers, at the cost of a more complex theory; the latter limitation was mostly unsolved until the publication of our work. We highlight the work [4], which integrates decidable assertions in the multiparty session types framework, in the style of type refinements, by relying on an ad-hoc semantic notion of well-asserted protocol to ensure satisfiability of assertions.

Concurrently to the developments above, Caires and Pfenning [5] discovered a deep connection between session types and linear logic, providing a purely logical account of the key features of session types and the first true propositions-as-types account for concurrent communication, where session types are interpreted as the propositions of linear logic, their proofs as well-typed processes and proof reduction as communication. Beyond providing stronger static correctness guarantees, namely progress in the presence of interleaved, higher-order linear and replicated session communication, the logical foundation of session types provided the basis on which to build our notion of dependent session type.

2 Dependent Session Types

Our work [22] expanded the propositions-as-types correspondence of [5] by introducing a logically motivated concept of *dependent* session types, by formulating a computational interpretation of first-order linear logic as a typed session calculus. In such a setting, processes can not only communicate channels along channels (as in [5]), but also send values drawn from a (dependent) type theory, effectively communicating not just simple terms but also *checkable* proof objects which attest the properties that can be asserted in session types. In this sense, dependent session types address the expressiveness limitation of the previous section in a general way, given that explicit proof communication allows types to express *arbitrary* properties of data (unlike [4]) that are witnessed by the proof objects that are exchanged during communication. Moreover, the logical

PPDP 2021, September 6–8, 2021, Tallinn, Estonia

ACM ISBN 978-1-4503-8689-0/21/09.
https://doi.org/10.1145/3479394.3479398

foundation allows us to naturally incorporate a notion of *proof irrelevance* as a principled way of eliminating some of the communication overhead generated by the exchange of explicit proof objects, requiring the proof objects to exist during typechecking but allowing them to be erased at runtime. Our work also introduced a primitive *forwarder*, equating two session channels, which has been adopted in subsequent works based on the session interpretation of linear logic [25].

To illustrate these ideas, we revisit an example of [22] of a simple session type describing a minimalistic bank service:

$$\$\mathsf{id} \multimap \&\{\mathsf{dep} : \$\mathsf{int} \multimap (\$\mathsf{int} \otimes \mathbf{1}), \mathsf{bal} : \$\mathsf{int} \otimes \mathbf{1}\}$$

The type above describes an agent that expects to receive ($\multimap$) a value of type id, abstracting the client's identification data. The bank will then offer its clients a choice ($\&$) between a deposit and a balance operation, triggered by the client sending the label dep or bal, respectively. In the former case, the client must send the deposit amount, after which the bank sends ($\otimes$) back a receipt of the deposited amount and the session ends (**1**). In the latter case, the bank simply sends the account balance. Using the framework of dependent session types, we can enrich the type for the bank beyond indicating the simple types of exchanged values:

$$\begin{aligned}\forall s{:}\mathsf{string}.\$\mathsf{uid}(s) \multimap \&\{&\mathsf{bal} : \exists m{:}\mathsf{int}.\$\mathsf{balance}(s,m) \otimes \mathbf{1},\\ &\mathsf{dep} : \forall n{:}\mathsf{int}.\$\mathsf{dpst}(s,n) \multimap \$\mathsf{rcpt}(s,n) \otimes \mathbf{1}\}\end{aligned}$$

By using dependent types we can refine the specification by having the client and the bank exchange objects whose typing certifies that the initial value s sent by the client is a valid user id ($\mathsf{uid}(s)$), and, for the deposit operation, that the deposit of amount n is intended for the client with id s (witnessed by the object of type $\mathsf{dpst}(s,n)$) and, that the value mentioned in the receipt is exactly n ($\mathsf{rcpt}(s,n)$) ensuring the bank does not charge for the deposit. A variant of the deposit branch that charges no more than 2 units for the deposit is:

$$\forall n{:}\mathsf{int}.\$\mathsf{dpst}(s,n) \multimap \exists m{:}\mathsf{int}.\exists p{:}(n-2 \leq m \leq n).\$\mathsf{rcpt}(s,m) \otimes \mathbf{1}$$

In this case, the bank will send an integer m, a proof p that m is between $n-2$ and n, and the object $\mathsf{rcpt}(s,m)$.

Thus, the concept of dependent session types enabled a new paradigm of expressiveness for session types, ensuring not just session fidelity and global progress of communications but also statically certifying that the properties asserted in types must hold, with the added benefit that certificates of such properties are exchanged by processes – a key requirement in a potentially distributed setting – and can also be selectively ommitted when such is deemed appropriate.

3 Impact and Further Developments

Logical Foundations. Following the publication of our work, the framework of dependent session types as an exploration of first-order linear logic was further developed in [6, 18], which incorporated an indexed affirmation modality to allow for specifications to refer to digital signatures. As the logical foundation of session types matured, a session interpretation for classical linear logic was proposed [7, 25].

Intrinsic to the pure logical view of session types is strong normalization of well-typed programs, disallowing general recursive definitions. To this end, various works [16, 23] have studied how to add recursive process and type definitions to the logical interpretation (in both classical and intuitionistic settings) while preserving termination and confluence and so maintaining logical soundness. Going beyond the forms of recursion allowed in the works above, the work [9] proposes nested session types, a system with parametric polymorphism, recursively defined types with free nesting polymorphic type constructors. The work shows type equality is decidable in such a setting and presents a practical algorithm for type equality with nested types.

In terms of session type dependency and refinement arising from the logical interpretation, the works [10] and [11] crucially leverage a form of value dependency (via type-indexing) to develop *temporal* and *ergometric* session types to statically analyze work and span of parallel and concurrent programs. However, the interaction of recursive types and various forms of session type dependency has only begun to be understood more recently. The works [12, 13] show that (indexed) recursive session types with arithmetic refinements from Presburger arithmetic result in undecidable type equality and subtyping, proposing a coinductive, deterministic algorithm that soundly approximates type equality.

Beyond Linearity. Another way to increase the expressiveness of logical sessions is by considering a sharing semantics for the exponential modality [2] via sharing modalities in the type structure, allowing cyclic proof structures (and so deadlocks). A similar effect is achieved by conflating dual session types in classical linear logic [1]. Deadlock-freedom can be recovered through extra-logical means [3, 8] or, in recent work, by exploring differential linear logic [19].

Verified Session-Typed Programming. While the session interpretation for first-order linear logic used a session π-calculus, several works have studied the ideas of dependent sessions in less austere settings. We highlight the work [13], which introduces the Rast language and studies the problem of refinement reconstruction in the presence of arithmetic refinements. Outside the logical interpretation, statically verified refinements have been studied for multiparty session embeddings in F* [26] and F# [17].

Beyond Value Dependencies. All works on dependent sessions listed above deal with some form of limited dependency or refinement. The work [24] enables both session and functional dependencies on (session-typed) quoted processes, but does not study inductive or recursive types. A notion of label dependency is studied in [21], which also includes type-level natural numbers and iteration.

4 The Next Decade

We now engage in some futurology on the topic of dependent session types. While much has been explored in this space, we have yet to see a full dependent type theory of sessions that can allow for generalized forms of dependency in the presence of inductive and coinductive session types. The recent works on type-based termination for sessions [20] and nested session types [9] provide promising insights.

Another crucial avenue of exploration is the reconciliation of type dependency in settings that go beyond the tree-like topologies of linear session typing. The recent work of [19] that achieves sharing through the logical means of differential linear logic has great potential in this matter.

Acknowledgments

This work is supported by NOVA LINCS (UIDB/04516/2020).

References

[1] Robert Atkey, Sam Lindley, and J. Garrett Morris. 2016. Conflation Confers Concurrency. In *A List of Successes That Can Change the World - Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday*, Sam Lindley, Conor McBride, Philip W. Trinder, and Donald Sannella (Eds.). Springer, 32–55. https://doi.org/10.1007/978-3-319-30936-1_2

[2] Stephanie Balzer and Frank Pfenning. 2017. Manifest sharing with session types. *Proc. ACM Program. Lang.* 1, ICFP (2017), 37:1–37:29. https://doi.org/10.1145/3110281

[3] Stephanie Balzer, Bernardo Toninho, and Frank Pfenning. 2019. Manifest Deadlock-Freedom for Shared Session Types. In *28th European Symposium on Programming, ESOP 2019*, Luís Caires (Ed.). Springer, 611–639. https://doi.org/10.1007/978-3-030-17184-1_22

[4] Laura Bocchi, Kohei Honda, Emilio Tuosto, and Nobuko Yoshida. 2010. A Theory of Design-by-Contract for Distributed Multiparty Interactions. In *CONCUR 2010 - Concurrency Theory, 21th International Conference*, Paul Gastin and François Laroussinie (Eds.). Springer, 162–176. https://doi.org/10.1007/978-3-642-15375-4_12

[5] Luís Caires and Frank Pfenning. 2010. Session Types as Intuitionistic Linear Propositions. In *CONCUR 2010 - Concurrency Theory, 21th International Conference*, Paul Gastin and François Laroussinie (Eds.). Springer, 222–236. https://doi.org/10.1007/978-3-642-15375-4_16

[6] Luís Caires, Frank Pfenning, and Bernardo Toninho. 2012. Towards concurrent type theory. In *TLDI 2012: 7th ACM SIGPLAN Workshop on Types in Languages Design and Implementation*, Benjamin C. Pierce (Ed.). ACM, 1–12. https://doi.org/10.1145/2103786.2103788

[7] Luís Caires, Frank Pfenning, and Bernardo Toninho. 2016. Linear logic propositions as session types. *Math. Struct. Comput. Sci.* 26, 3 (2016), 367–423. https://doi.org/10.1017/S0960129514000218

[8] Ornela Dardha and Simon J. Gay. 2018. A New Linear Logic for Deadlock-Free Session-Typed Processes. In *Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018*, Christel Baier and Ugo Dal Lago (Eds.). Springer, 91–109. https://doi.org/10.1007/978-3-319-89366-2_5

[9] Ankush Das, Henry DeYoung, Andreia Mordido, and Frank Pfenning. 2021. Nested Session Types. In *30th European Symposium on Programming, ESOP 2021*, Nobuko Yoshida (Ed.). Springer, 178–206. https://doi.org/10.1007/978-3-030-72019-3_7

[10] Ankush Das, Jan Hoffmann, and Frank Pfenning. 2018. Parallel complexity analysis with temporal session types. *Proc. ACM Program. Lang.* 2, ICFP (2018), 91:1–91:30. https://doi.org/10.1145/3236786

[11] Ankush Das, Jan Hoffmann, and Frank Pfenning. 2018. Work Analysis with Resource-Aware Session Types. In *33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018*, Anuj Dawar and Erich Grädel (Eds.). ACM, 305–314. https://doi.org/10.1145/3209108.3209146

[12] Ankush Das and Frank Pfenning. 2020. Session Types with Arithmetic Refinements. In *CONCUR 2020 - Concurrency Theory, 31th International Conference*, Igor Konnov and Laura Kovács (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 13:1–13:18. https://doi.org/10.4230/LIPIcs.CONCUR.2020.13

[13] Ankush Das and Frank Pfenning. 2020. Verified Linear Session-Typed Concurrent Programming. In *PPDP '20: 22nd International Symposium on Principles and Practice of Declarative Programming*. ACM, 7:1–7:15. https://doi.org/10.1145/3414080.3414087

[14] Kohei. Honda, Vasco T. Vasconcelos, and Makoto Kubo. 1998. Language Primitives and Type Discipline for Structured Communication-Based Programming. In *7th European Symposium on Programming Languages and Systems (ESOP'98)*. Springer LNCS 1381, 122–138.

[15] Kohei Honda, Nobuko Yoshida, and Marco Carbone. 2016. Multiparty Asynchronous Session Types. *J. ACM* 63, 1 (2016), 9:1–9:67. https://doi.org/10.1145/2827695

[16] Sam Lindley and J. Garrett Morris. 2016. Talking bananas: structural recursion for session types. In *21st ACM SIGPLAN International Conference on Functional Programming, ICFP 2016*, Jacques Garrigue, Gabriele Keller, and Eijiro Sumii (Eds.). ACM, 434–447. https://doi.org/10.1145/2951913.2951921

[17] Rumyana Neykova, Raymond Hu, Nobuko Yoshida, and Fahd Abdeljallal. 2018. A session type provider: compile-time API generation of distributed protocols with refinements in F#. In *27th International Conf. on Compiler Construction, CC 2018*, Christophe Dubach and Jingling Xue (Eds.). ACM, 128–138. https://doi.org/10.1145/3178372.3179495

[18] Frank Pfenning, Luís Caires, and Bernardo Toninho. 2011. Proof-Carrying Code in a Session-Typed Process Calculus. In *Certified Programs and Proofs - First International Conference, CPP 2011*, Jean-Pierre Jouannaud and Zhong Shao (Eds.). Springer, 21–36. https://doi.org/10.1007/978-3-642-25379-9_4

[19] Pedro Rocha and Luís Caires. 2021. Propositions-as-Types and Shared State. *Proc. ACM Program. Lang.* 5, ICFP. to appear.

[20] Siva Somayyajula and Frank Pfenning. 2021. Circular Proofs as Processes: Type-Based Termination via Arithmetic Refinements. (2021). https://arxiv.org/abs/2105.06024

[21] Peter Thiemann and Vasco T. Vasconcelos. 2020. Label-dependent session types. *Proc. ACM Program. Lang.* 4, POPL (2020), 67:1–67:29. https://doi.org/10.1145/3371135

[22] Bernardo Toninho, Luís Caires, and Frank Pfenning. 2011. Dependent session types via intuitionistic linear type theory. In *13th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming*, Peter Schneider-Kamp and Michael Hanus (Eds.). ACM, 161–172. https://doi.org/10.1145/2003476.2003499

[23] Bernardo Toninho, Luís Caires, and Frank Pfenning. 2014. Corecursion and Non-divergence in Session-Typed Processes. In *Trustworthy Global Computing - 9th International Symposium, TGC 2014, Revised Selected Papers*, Matteo Maffei and Emilio Tuosto (Eds.). Springer, 159–175. https://doi.org/10.1007/978-3-662-45917-1_11

[24] Bernardo Toninho and Nobuko Yoshida. 2017. Certifying data in multiparty session types. *J. Log. Algebraic Methods Program.* 90 (2017), 61–83. https://doi.org/10.1016/j.jlamp.2016.11.005

[25] Philip Wadler. 2012. Propositions as sessions. In *ACM SIGPLAN International Conference on Functional Programming, ICFP'12*, Peter Thiemann and Robby Bruce Findler (Eds.). ACM, 273–286. https://doi.org/10.1145/2364527.2364568

[26] Fangyi Zhou, Francisco Ferreira, Raymond Hu, Rumyana Neykova, and Nobuko Yoshida. 2020. Statically verified refinements for multiparty protocols. *Proc. ACM Program. Lang.* 4, OOPSLA (2020), 148:1–148:30. https://doi.org/10.1145/3428216