Foundations of Session Types: 10 Years Later

This is the author's manuscript

Original Citation:

Availability:
This version is available http://hdl.handle.net/2318/1712465 since 2019-09-28T17:58:04Z

Publisher:
ACM

Published version:
DOI:10.1145/3354166.3356340

Terms of use:
Open Access
Anyone can freely access the full text of works made available as "Open Access". Works made available under a Creative Commons license can be used according to the terms and conditions of said license. Use of all other works requires consent of the right holder (author or publisher) if not exempted from copyright protection by the applicable law.

(Article begins on next page)
Foundations of Session Types: 10 Years Later

Giuseppe Castagna
CNRS - IRIF
Université de Paris
France

Elena Giachino
Università di Bologna
Italy

Mariangiola Dezani-Ciancaglini
Dipartimento di Informatica
Università di Torino
Italy

Luca Padovani
Dipartimento di Informatica
Università di Torino
Italy

CCS CONCEPTS
- Theory of computation → Process calculi. Type structures; Operational semantics.

KEYWORDS
Concurrency, Communication-Centered Programming, Sessions Types, Testing Equivalencies, Semantic Subtyping

ACM Reference Format:

We were thrilled to know that our PPDP’09 paper “Foundations of Session Types” [10] was selected for the PPDP Most Influential Paper 10-Year Award. Just moments after being notified of this, we couldn’t help looking at the works that cited—and in some cases were inspired by—our own. The result is the following short note, in which we recollect the main ideas behind our own work and the related ones that followed. The tight gap between the award notification and the deadline for the production of the PPDP’19 proceedings prevents us from providing an exhaustive survey of the related literature and we apologize in advance for the conciseness of our report and any relevant omission. Fortunately, there exist recent surveys [2, 4, 14, 21] that may help the interested readers orient themselves into the vast literature of session types as a whole.

Sessions and session types have proved to be extremely successful concepts for the structuring and the analysis of communications in distributed systems. A session is a private communication channel through which participating processes, using the so-called session endpoints, can communicate without interference from other processes. This privacy property of sessions enables the modular reasoning on complex systems, whereby each session is treated—and typed—in isolation. Thus, it is relatively easy to conceive type systems that ensure communication safety (only messages of the expected type are exchanged), protocol fidelity (communication progresses according to a given sequence of input/output operations) and under some additional assumptions also progress (the system does not get stuck in a configuration with unperformed pending operations). These type systems associate each session endpoint with a session type, which essentially describes a communication protocol in terms of the sequence of operations a process is supposed to perform on that endpoint. Most often, this protocol description also contains branching points, whereby processes may choose (internally) or offer (externally) different continuations, depending on the value of a specific message that identifies a branch.

The milestone papers by Honda [17] and Honda et al. [18] introduce binary sessions and binary session types along with the key notion of duality. Duality captures the idea that peer endpoints of the same (binary) session should be used, according to the respective session types, in complementary ways: where one session type specifies the output of a message, its dual specifies the input of a message with the same type; where one session type specifies an internal choice, the dual one specifies an external choice. Ultimately, duality ensures that the interaction between two channels typed by dual session types will never get stuck.

Another notion that stems from type theory and that applies to session types is that of subtyping: it is safe to use a value with type \( T \) whereby a value with type \( S \) is expected, provided that \( S \) is a subtype of \( T \). Gay and Hole [15] were the first to study a notion of subtyping for session types and their work has become a classic reference in the session type literature. They proceed by first defining a subtyping relation (in the particular case, using coinduction, since session types may be recursive), and then proving that this relation is sound (i.e., it does not compromise any of the safety properties ensured by sessions) if used in the type system for a particular process calculus.

Our goal in writing the PPDP’09 paper was to give a theory of session types that was as much language independent as possible and where all the relevant definitions, duality and subtyping in particular, were semantically justified. Our approach was founded on two key insights. The first one was the realization that duality and subtyping are closely connected. Laneve and Padovani [22] had already pinpointed remarkable connections between session subtyping and testing equivalences for processes [12], where the notion of duality between two session types can be generalized to the relation between a process and a test. The second key insight...
was inspired by the work of Frisch et al. [13], who had shown that the notion of subtyping could emerge semantically, given a suitable interpretation of types. If a type is interpreted as a set of values, namely, as the set of all the possible results that the expressions of that type may produce, then subtyping can be defined as result containment: an expression of a given type \( T \) can be safely used in a context where an expression of type \( S \) is expected if and only if the results that may be produced by that expression are less than—i.e., included in—those that the context can safely handle. This explains why their approach is called semantic subtyping.

Transposing the notion of semantic subtyping from a traditional setting (where types describe values) to sessions (where types describe protocols) requires defining a set-theoretic interpretation of session types. Session types cannot be characterized by the set of their results, insofar as typical sessions do not produce any result but, rather, a set of possibly infinite interactions—more precisely, a set of possibly infinite sequences of interaction offers. In our PPDP’09 paper, the interpretation of a session type is taken to be the set of its duals and two session types are related by subtyping if and only if all dual types of the smaller type were dual of the larger type as well. This construction is reminiscent of the realizability semantics of constructive logic. In hindsight, this interpretation is rather natural recalling that subtyping is a preorder relation defined to capture “safe replacement”: every expression of a given type can be safely (in the sense of Wright and Felleisen [29]) used wherever an expression of a supertype is expected. Compared to the work on semantic subtyping by Frisch et al. [13], in PPDP’09 we skip one step, so to speak: instead of characterizing a type by a set of results and, indirectly, a context by the set of results it can safely handle, we directly characterized a type by a set of contexts that can safely handle the interactions (i.e., the session type “results”) offered by endpoints of that type. In light of that, it is then quite natural to interpret a session type as the set of its dual session types, that is, as the set of contexts that can safely handle the interactions offered by that session type: this interpretation characterizes the interactions that an endpoint of that session type may offer (trivially, the more the dual types, the more the interactions the endpoint may produce). We thus recover the “safe replacement” interpretation of subtyping: an endpoint implementing a session type \( T \) can be safely used in a context where an endpoint of a session type \( S \) is expected if and only if the interactions it may offer are less than—i.e., included in—those that the context can safely handle. Our transposition of semantic subtyping to session types has subsequently inspired the subtyping relations given by Bartoletti et al. [5] and Barbanera and de’Liguoro [3] and Bejleri et al. [6] also model internal and external choices according to the same intuition.

One of the most intriguing aspects of the PPDP’09 semantic approach to the definition of session subtyping is the relationship between the parallel operator. Notoriously, it is not trivial to guarantee the absence of deadlock when processes are allowed to interact through more than one session. Our solution makes use of a stack to constrain the order in which sessions can be used: only the session on top of the stack can communicate. Similar approaches to prevent deadlocks are described by Padovani [24] and Spaccasassi and Koutavas [27, 28]. Subsequently, other approaches to prevent deadlocks and not requiring an explicit ordering of sessions have emerged following the logical interpretation of session types [9]. Related to the notion of subtyping is that of preciseness [23], which requires to find a context expecting a term of type \( T \) and a term of type \( S \) such that filling the context with the term “goes wrong” whenever \( S \) is not a subtype of \( T \). Preciseness for session types is studied by Chen et al. [11] and Ghilezan et al. [16] referring to our PPDP’09 paper, since semantic subtyping is precise by construction.

We conclude this note observing that two problems left open by Castagna et al. [10] have now been solved. The first one is to allow a channel to transmit a message containing itself, and in particular to transmit just itself. Bernardi and Hennessy [7] give a set theoretic model of session types without the need of stratification, so that it becomes possible to type channels carrying themselves. The second problem is the extension of the approach to multiparty session types [19], which describe interactions between a fixed but arbitrary number of processes. This extension has been developed by Padovani [26], where session types can be composed also using the parallel operator.

REFERENCES


