Information and Computation 256 (2017) 253–286

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Session types revisited

Ornela Dardha [a,*], Elena Giachino [b], Davide Sangiorgi [b]

[a] School of Computing Science, University of Glasgow, United Kingdom
[b] INRIA Focus Team / DISI, University of Bologna, Italy

ARTICLE INFO

Article history:
Received 14 April 2015
Received in revised form 15 December 2016
Available online 7 June 2017

Keywords:
Session types
π-Calculus
Linear types
Variant types
Encoding

ABSTRACT

Session types are a formalism used to model structured communication-based programming. A binary session type describes communication by specifying the type and direction of data exchanged between two parties. When session types and session processes are added to the syntax of standard π-calculus they give rise to additional separate syntactic categories. As a consequence, when new type features are added, there is duplication of effort in the theory: the proofs of properties must be checked both on standard types and on session types. We show that session types are encodable into standard π-types, relying on linear and variant types. Besides being an expressivity result, the encoding (i) removes the above redundancies in the syntax, and (ii) the properties of session types are derived as straightforward corollaries, exploiting the corresponding properties of standard π-types. The robustness of the encoding is tested on a few extensions of session types, including subtyping, polymorphism and higher-order communications.

1. Introduction

In complex distributed systems, participants willing to communicate should previously agree on a protocol to follow. The specified protocol describes the types of messages that are exchanged as well as their direction. In this context *session types* [15,29,16] came into play: they describe a protocol as a type abstraction. Session types were originally designed for process calculi. However, they have been studied also for other paradigms, such as multi-threaded functional programming [32], component-based systems [30], object-oriented languages [10,11,3], Web Services and Contracts, W3C-CDL a language for choreography [5,22] and many more. Session types are a type formalism proposed as a theoretical foundation to describe and model structured communication-based programming, guaranteeing properties like session fidelity, privacy and communication safety.

Session types are defined as a sequence of input and output operations, explicitly indicating the types of messages being transmitted. This structured *sequentiality* of operations is what makes session types suitable to model protocols. However, they offer more flexibility than just performing inputs and outputs: they also permit internal and external choice. Branch and select are typical type (and term) constructs in the theory of session types, the former being the offering of a set of alternatives and the latter being the selection of one of the possible options at hand.

As mentioned above, session types guarantee session fidelity, privacy and communication safety. Session fidelity guarantees that the session channel has the expected structure. Privacy is guaranteed since session channels are known and used only by the participants involved in the communication. Such communication proceeds without any mismatch of direction

* Corresponding author.
E-mail addresses: Ornela.Dardha@glasgow.ac.uk (O. Dardha), egiachino@gmail.com (E. Giachino), davide.sangiorgi@gmail.com (D. Sangiorgi).

http://dx.doi.org/10.1016/j.ic.2017.06.002

and of message type. In order to achieve communication safety, a session channel is split by giving rise to two opposite endpoints, each of which is owned by one of the participants. These endpoints are used according to dual behaviours and thus have dual types, namely one participant sends what the other one is expecting to receive and vice versa. So, *duality* is a key concept in the theory of session types as it is the ingredient that guarantees communication safety.

To better understand session types and the notion of duality, let us consider a simple example: the equality test. A client and a server communicate over a session channel. The endpoints x and y of the session channel are owned by the client and the server respectively and exclusively and must have dual types. To guarantee duality of types, static checks are performed by the type system. If the type of x is

$$\texttt{?Int.?Int.!Bool.end}$$

— meaning that the process listening on channel endpoint x receives (?) an integer value followed by another integer value and then sends (!) back a boolean value corresponding to the equality test of the integers received — then the type of y should be

$$\texttt{!Int.!Int.?Bool.end}$$

— meaning that the process listening on channel endpoint y sends an integer value followed by another integer value and then waits to receive back a boolean value — which is exactly the dual type.

There is a precise moment at which a session between two participants is established. It is the *connection* phase, when a fresh (private) session channel is created and its endpoints are bound to each communicating process. The connection is also the moment when duality, hence compliance of two session types, is verified. In order to establish a connection, primitives like `accept/request` or $(\boldsymbol{\nu} xy)$, are added to the syntax of terms [29,16,31].

Session types and session terms are added to the syntax of standard π-calculus types and terms, respectively. In doing so, the syntax of types often needs to be split into two separate syntactic categories, one for session types and the other for standard π-calculus types [29,16,34,14] (this often introduces a duplication of type environments, as well). Common typing features, like subtyping, polymorphism, recursion are then added to both syntactic categories. Also the syntax of processes will contain both standard π-calculus process constructs and session process constructs (for example, the constructs mentioned above to create session channels). This redundancy in the syntax brings in redundancy also in the theory, and can make the proofs of properties of the language heavy. In particular, this duplication becomes more obvious in proofs by structural induction on types. Moreover, if a new type construct is added, the corresponding properties must be checked both on standard π-types and on session types. By "standard type systems" we mean type systems originally studied in depth in sequential languages such as the λ-calculus and then transplanted onto the π-calculus as types for channel names (rather than types for terms as in the λ-calculus). Such type systems may include constructs for products, records, variants, polymorphism, linearity, and so on.

In this paper we aim to understand to which extent this redundancy is necessary, in the light of the following similarities between session constructs and standard π-calculus constructs. Consider `?Int.?Int.!Bool.end`. This type is assigned to a session channel endpoint and it describes a structured sequence of inputs and outputs by specifying the type of messages that it can transmit. This way of proceeding reminds us of the *linearised* channels [21], which are channels used multiple times for communication but only in a sequential manner. This paper [21] discusses the possibility of encoding linearised channel types into linear types — i.e., channel types used *exactly once*.

The considerations above deal with input and output operations and the sequentiality of session types. Let us consider branch and select. These constructs give more flexibility by offering and selecting a range of possibilities. This brings in mind an already existing type construct in the π-calculus, namely the *variant* type [27,28]. Another analogy between the session types theory and the standard π-types theory, concerns duality. As mentioned above, duality is checked when connection takes place, in the typing rule for channel restriction. Duality describes the split of behaviour of session channel endpoints. This reminds us of the split of input and output *capabilities* of π-types: once a new channel is created via the ν construct, it can then be used by the two communicating processes owning the opposite capabilities.

In this paper, by following Kobayashi's approach [20], we define an encoding of binary session types into standard π-types and by exploiting this encoding, session types and their theory are shown to be derivable from the theory of the standard typed π-calculus. For instance, basic properties such as subject reduction and type safety become straightforward corollaries. Intuitively, a session type is interpreted as a linear channel type carrying a pair consisting of the original payload type and a new linear channel type, which is going to be used for the continuation of the communication. Furthermore, we present an optimisation of linear channels enabling the reuse of the same channel, instead of a new one, for the continuation of the communication. As stated above, the encoding we adopt follows Kobayashi [20] and the constructs we use are not new (linear types and variant types are well-known concepts in type theory and they are also well integrated in the π-calculus). Indeed the technical contribution of the paper may be considered minor (the main technical novelty being the optimisation in linear channel usage mentioned above). Rather than technical, the contribution of the paper is meant to be foundational: we show that Kobayashi's encoding

(i) does permit to derive session types and their basic properties; and
(ii) is a robust encoding.