FuSe: A Simple Library Implementation of Binary Sessions
(tool demonstration)

Hernán Melgratti (Buenos Aires, Argentina) and Luca Padovani (Torino, Italy)

Gay and Vasconcelos [2010] have defined a session type system that integrates smoothly with an ML-like functional language. From a practical viewpoint, however, the type system appears challenging to adopt in a mainstream programming language, for it requires peculiar features for (F.1) describing structured protocols as sequences of I/O operations and branching points, (F.2) checking that the peer endpoints of a session are used according to dual protocols, and (F.3) ensuring the linear usage of session endpoints. Successful attempts to encode these features in Haskell have been reported by Neubauer and Thiemann [2004], Sakman and Eisenbach [2008], Pucella and Tov [2008], and Imai et al. [2010]. These works accomplish (F.1) and (F.2) using advanced features of Haskell’s type system and (F.3) by encapsulating endpoints in a suitable monad. As discussed by Imai et al. [2010], however, all of these solutions have a price in terms of expressiveness, usability, or portability.

FuSe [Padovani, 2015] is an OCaml implementation of binary sessions that diverges from previous approaches in two ways. Concerning (F.1), FuSe represents session types according to the continuation-passing encoding of binary sessions documented by Dardha et al. [2012] while maintaining the natural communication semantics given by Gay and Vasconcelos [2010]. The main advantage of the chosen representation of session types is that duality boils down to type equality and this gives a straightforward and portable solution to (F.2). Concerning (F.3), FuSe adopts a lightweight mechanism that detects potentially unsafe linearity violations at runtime. Similar mechanisms have been described by Tov and Pucella [2010] and Hu and Yoshida [2016]. With this setup and piggybacking on OCaml, FuSe provides a faithful implementation of the communication API given by Gay and Vasconcelos [2010]. In addition, it supports equi-recursive and polymorphic session types [Bono et al., 2013], session subtyping [Gay and Hole, 2005], and enables complete session type inference through OCaml’s inference engine.

In this demonstration we glance at the internals of FuSe and then see it at work on a few well-known examples taken from the existing literature on session types. We devote part of the demonstration to illustrate the effectiveness of FuSe in detecting errors and providing comprehensible inferred types and error messages. In particular, we argue that the very nature of session types makes it possible to detect a fair number of linearity violations even if OCaml’s type system is not sub-structural. To compensate for the fact that encoded session types are often difficult to comprehend, FuSe provides an accompanying utility called rosetta that pretty prints OCaml types using the familiar session type notation.

Two more features of FuSe are worth mentioning, but their demonstration is subjected to time availability. Recently, Thiemann and Vasconcelos [2016] have introduced context-free session types to enable precise descriptions of protocols that elude the expressiveness of conventional session types. Examples include serialization protocols for tree-like data structures and XML documents and interaction protocols with non-uniform objects [Nierstrasz, 1993; Ravara and Vasconcelos, 2000] such as buffers and reentrant locks. FuSe provides a practical solution to the problems of context-free session type checking (left open by Thiemann and Vasconcelos) and context-free session type inference as well. Finally, we have developed a theory of contract monitoring [Findler and Felleisen, 2002] adapting to sessions the concept of chaperone contract [Strickland et al., 2012] and proving a blame theorem result [Wadler and Findler, 2009] in presence of possibly dependent session contracts. Runtime contract monitoring has been implemented and will be made publicly available in a forthcoming relase of FuSe.
References


