Science of Computer Programming 115–116 (2016) 100–126

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Parameterized, concurrent session types for asynchronous multi-actor interactions

Minas Charalambides [*], Peter Dinges, Gul Agha

ARTICLE INFO

Article history:
Received 24 January 2013
Received in revised form 2 October 2015
Accepted 5 October 2015
Available online 26 October 2015

Keywords:
Session types
Parameterized
Static
Actors
Concurrency

ABSTRACT

Session types have been proposed as a means of statically verifying implementations of communication protocols. Although prior work has been successful for some classes of protocols, it does not cope well with parameterized, multi-actor scenarios with inherent asynchrony. For example, the sliding window protocol is not expressible in previously proposed session type notations. This article defines System-A: a novel session type system, as well the associated programming language that together overcome many of the limitations of prior work. With explicit support for asynchrony and concurrency, as well as multiple forms of parameterization, we demonstrate that System-A can be used for the static verification of a large class of asynchronous communication protocols.

1. Introduction

Session types [35] are a means of expressing the order and type of messages exchanged by concurrently executing processes. In particular, session types can be used to statically check if a group of processes communicates according to a given specification. In these systems, a *global type* specifies the permissible sequences of messages that participants may exchange in a given *session*, as well as the types of these messages. The typing requires the programmer to provide the *global type*. A *projection* algorithm then generates the restrictions implied by the global type for each participant. Such restrictions are called *end-point types* or *local types* and describe the expected behavior of the individual participants in the protocol. The actual program code implementing the behavior of a participant is checked for conformance against this localized behavior specification. We are interested in generalizing prior work on session types to typing coordination constraints in *parameterized* actor [1] programs, which can then be enforced, e.g., with *Synchronizers* [27,28,24] or other ways [42,2].

Typing coordination constraints in actors requires addressing two problems. First, asynchronous communication leads to delays that require considering arbitrary shuffles. Second, we need to consider *parameterized* protocols. For example, assume two actors communicating through a sliding window protocol [51]: the actors agree on the length of the window (i.e., the number of messages that may be buffered) and then proceed to a concurrent exchange of messages. Prior work on session types is not suitable for typing interactions such as the sliding window protocol. The reason for this limitation is that their respective type languages depend on other formalisms for type checking (such as typed λ-calculus [3] or System T [32]), and these formalisms do not support a concurrency construct.

Contributions We present a programming language along with a session type system that overcomes many of the aforementioned limitations through the use of parameters and novel constructs. The session types were originally introduced in

[*] Corresponding author.
E-mail addresses: charala1@illinois.edu (M. Charalambides), pdinges@acm.org (P. Dinges), agha@illinois.edu (G. Agha).

http://dx.doi.org/10.1016/j.scico.2015.10.006
0167-6423/© 2015 Elsevier B.V. All rights reserved.

our 2012 FOCLASA paper [17], which serves as the foundation of the present article. The three primary extensions are (a) the introduction and formal definition of Lang-A, a programming language for expressing protocols typeable in our session type system, System-A; (b) an inference algorithm that derives local System-A types from Lang-A programs, and (c) the formal treatment of the type system.

Overall, our work on System-A makes the following contributions: (i) the introduction of *parameterized* constructs for expressing asynchrony, concurrency, sequence, choice and atomicity in protocols; (ii) a projection mechanism that extracts local type constraints on individual actors from the global type; (iii) a formalization of the conditions under which conformance of these constraints to the global type is assured; (iv) a normalization algorithm for local types, which allows equivalence testing; and (v) trace semantics of System-A and Lang-A. We furthermore supply the formal proofs of various useful properties, such as those concerning items (iii) and (iv), as well as standard sanity checks of the type system.

Limitations We do not address dynamic actor creation in this article, and briefly discuss the related difficulties in Section 11. We furthermore omit support for session delegation, and do not deal with issues of progress. Finally, we do not consider overlapping indexed names when nested in multiple operators. This disallows some cases, such as all-to-all communication. A more accurate description of how we restrict the use of indices is given in Section 7.

2. Related work

Session types [35,54,50,34] originate from the context of π-calculi as statically derivable descriptions of process interaction behaviors. In two-party sessions, they allow us to statically verify that the participants have compatible behavior by requiring *dual* session types, that is, behaviors where each participant expects precisely the message sequence that the other participant sends and vice versa. Extensions to session types support asynchronous message passing [36] and introduce subtyping [29] for a looser notion of type compatibility. Session types have been integrated into functional [52,47] and object-oriented [23,38,31] languages among others. Other extensions deal with evolving system specifications using transformations [25]. Exception handling, which allows the participants of a protocol to escape the normal flow of control and coordinate on another, has also been considered [14,12]. The present article introduces a novel combination of two enhancements to session types: we parameterize both the number of participants, and the type constructs, including those introducing asynchrony. This greatly extends the applicability of types.

Asynchronous multi-party sessions Many real-world protocols involve more than two participants, which makes their description in terms of multiple two-party sessions unnatural. To overcome this limitation, Honda et al. [36] extend session types to support multiple participants: A *global type* specifies the interactions between all participants from a global perspective. A projection algorithm then mechanically derives the behavior specification of each individual participant, that is, its *local type*.

The notion of a global specification and the associated correctness requirements for projection were first studied by Carbone et al. [11], although in the context of concrete implementations rather than session types; Bonelli's work on multipoint session types [9] treats multi-party protocols from the local perspective only. Bettini et al. [7] allow multi-party sessions to interleave and derive a type system guaranteeing global progress. Gay and Vasconcelos [30] consider subtyping in presence of asynchrony. Recent work by Carbone and Montesi [13] takes a novel approach to asynchrony by treating programs from the global perspective and regaining concurrent composition through a suitable swap relation. Although it obviates the need for an explicit concurrent operator, their approach does not support parameterized programs. In contrast, we include a parameterized concurrent construct in our types. Concurrent composition is also treated in the work of Kouzapas et al. [39], who apply session type discipline to π-calculus. An event-driven approach to asynchrony is explored by Hu et al. [37]. However, neither the system of Kouzapas, nor that of Hu handles parameterized asynchrony.

The type systems introduced by Puntigam [49,48] deal with asynchrony in the context of actors. His approach has the benefit of not utilizing global types; instead, given an actor's specification, the system ensures its proper use. To the best of our knowledge, this is also the only relevant body of work where the language supports dynamic actor creation, which is something that System-A does not handle. However, it is important to note that Puntigam's systems ensure safety properties on a per-actor basis, which is (naturally, due to the absence of global types) somehow limited in scope. Parameters are also not considered.

The present article builds on the foundation of a global protocol specification and its projection onto local behaviors, as in the work of Honda et al. [36]. Unlike Honda's approach (but following Castagna et al. [15,16]), we simplify the notation for global types by replacing recursion with the Kleene star and limiting each pair of participants to use a single bidirectional channel. We introduce an explicit shuffle operator to preserve the commutativity of message arrivals that can be achieved using multiple channels. Explicit shuffles also reduce the need for a special subtyping relationship that allows the permutation of (Lamport-style) concurrent asynchronous events for optimization [43].

Following Castagna's global type syntax further, we support join operations. Joins cannot be expressed in Honda et al.'s global types because of the linearity requirement. However, as Deniélou and Yoshida remark [22], join operations can only describe series–parallel graphs. Protocols such as the alternating bit protocol [41] that require interleaved synchronization between two processes consequently cannot be expressed in our global type language. Our choice to not support generic