A Session Logic for Communication Protocols

Florin Craciun[1], Wei-Ngan Chin[2] and Andreea Costea[2]

[1] Faculty of Mathematics and Computer Science, Babes-Bolyai University
craciunf@cs.ubbcluj.ro
[2] School of Computing, National University of Singapore
chinwn@comp.nus.edu.sg, andreeac@comp.nus.edu.sg

Abstract. Communication-centered programming is one of the most challenging programming paradigms. Development of modern software applications requires expressive mechanisms to specify and verify the communications between different parties. In the last decade, many works have used session types to characterize the various aspects of structured communications. Different from session types, we propose a novel session logic with disjunctions to specify and verify the implementation of communication protocols. Our current logic is based on only two-party channel sessions, but it is capable of handling delegation naturally through the use of higher-order channels. Due to our use of disjunctions to model both internal and external choices, we rely solely on conditional statements to support such choices, as opposed to specialized switch constructs in prior proposals. Furthermore, since our proposal is based on an extension of separation logic, it also supports heap-manipulating programs and copyless message passing. We demonstrate the expressivity and applicability of our logic on a number of examples.

With the success of tools like [1], [14] and [4] for the development and verification of non-concurrent systems, there are now deeper desires for similar tools for distributed systems. Additionally, the communication requirement, which is ubiquitous in software systems from the information exchange between software entities to the communication of these systems with their environments, must be properly verified to affirm the system's correctness. Due to its importance, a number of researchers have focused on the problems of ensuring safe communication in the last few decades.

CSP (Communicating Sequential Processes) [9] and CCS (Calculus of Communicating Systems) [11] are among the earliest theories to address the communication problems. The most recent extensions for these works are based on session types, and their derivatives, such as contracts [15]. In the last decade, session types have been integrated into a number of programming languages and process calculi, including functional languages [13, 5], object-oriented languages [8, 6], calculi of mobile processes [7], and higher-order processes [12]. Recently, session types have also been extended with logic [3] to act as a contract between the communication entities. This extension allows a more precise verification of the involved parties by enabling a concise specification of the transmitted messages on what one party must ensure, and from which the other party can rely on it.

There was also a proposal for multi-party session logic [2], but this logic tries to also summarize the effects of processes involved in the protocol. In contrast, we propose a session logic which focuses entirely on the communication patterns, while the effects of the associated processes are summarized directly in each thread's pre- and postcondition.

Contributions: Different from previous approaches, we propose a session logic with a novel (and natural) use of disjunction to specify and verify the implementation of communication protocols. Even though the currently proposed logic is based on two-party channel sessions, it can also handle delegation through the use of higher-order channels. Unlike past solution on delegation [6], our proposal uses the same send/receive channel methods for sending values, data structures, and channels. For example, [6] requires a separate set of send/receive methods to support higher-order channels. Furthermore, due to our use of disjunctions to model both internal and external choices, we need only use conventional conditional statements to support both kinds of choices. In contrast, past proposals typically require the host languages to be extended with a set of specialized switch constructs to model both internal and external choices. Additionally, our proposal is based on an extension of separation logic, and thus it supports heap-manipulating programs and copyless message passing. Lately, Villard et al. [10] have designed a logic for copyless message passing communication. Their logic relies on state-based global contracts while our more general logic of session is built as an extension of separation logic with disjunction to support communication choices. The logical formulae on protocols can also be localised to each channel and may be freely passed through procedural boundaries. Villard et al. [10] currently use double-ended channels to solely support communication safety, but do not guarantee deadlock freedom. In contrast, a channel in our proposal is multi-ended with its complementary properties captured in local specifications, which are supplied via each of the channel's aliases. As channels can support a variety of messages, we can treat the read content as dynamically typed where conditionals are dispatched based on the received types. Alternatively, we may also guarantee type-safe casting via verifying communication safety. We can also go beyond such cast safety by ensuring that heap memory and properties of values passed into the channels are suitably captured. Lastly by using a subsumption relation on our communication proposal, we allow specifications on channels to differ between threads but would ensure that they remain compatible at each join point, in order to prevent intra-channel deadlocks. More realistically, we also assume the presence of asynchronous communication protocols, where send commands are non-blocking.

In this paper, we argue strongly on the simplicity, expressivity and applicability of our logic by demonstrating it through a number of examples.

References

1. Abdulla, P.A., Deneux, J., Stålmarck, G., Ågren, H., Åkerlund, O.: Designing safe, reliable systems using scade. In: Leveraging Applications of Formal Methods, pp. 115–129. Springer (2006)

2. Bocchi, L., Demangeon, R., Yoshida, N.: A multiparty multi-session logic. In: Trustworthy Global Computing, pp. 97–111. Springer (2013)
3. Bocchi, L., Honda, K., Tuosto, E., Yoshida, N.: A theory of design-by-contract for distributed multiparty interactions. In: CONCUR 2010-Concurrency Theory, pp. 162–176. Springer (2010)
4. Chin, W.N., David, C., Gherghina, C.: A HIP and SLEEK verification system. In: Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion. pp. 9–10. ACM (2011)
5. DeLine, R., Fähndrich, M.: Enforcing high-level protocols in low-level software. ACM SIGPLAN Notices 36(5), 59–69 (2001)
6. Dezani-Ciancaglini, M., Mostrous, D., Yoshida, N., Drossopoulou, S.: Session types for object-oriented languages. In: ECOOP 2006–Object-Oriented Programming, pp. 328–352. Springer (2006)
7. Gay, S., Hole, M.: Subtyping for session types in the π calculus. Acta Informatica 42(2-3), 191–225 (2005)
8. Gay, S.J., Vasconcelos, V.T., Ravara, A., Gesbert, N., Caldeira, A.Z.: Modular session types for distributed object-oriented programming. In: ACM Sigplan Notices. vol. 45, pp. 299–312. ACM (2010)
9. Hoare, C.A.R.: Communicating sequential processes. Communications of the ACM 21(8), 666–677 (1978)
10. Lozes, É., Villard, J.: Shared contract-obedient channels. Science of Computer Programming 100, 28–60 (2015)
11. Milner, R.: A Calculus of Communication Systems. LNCS, vol. 92 (1980)
12. Mostrous, D., Yoshida, N.: Session typing and asynchronous subtyping for the higher-order π-calculus. Information and Computation 241, 227–263 (2015)
13. Pucella, R., Tov, J.A.: Haskell Session Types with (Almost) No Class. SIGPLAN Not. 44(2) (Sep 2008)
14. Schneider, S.: The B-method: An introduction. Palgrave Oxford (2001)
15. Villard, J., Lozes, É., Calcagno, C.: Proving copyless message passing. In: Programming Languages and Systems, pp. 194–209. Springer (2009)