Subtyping Context-Free Session Types

Gil Silva, Andreia Mordido, and Vasco T. Vasconcelos

LASIGE, Faculdade de Ciências da Universidade de Lisboa, Lisbon, Portugal

Keywords: Session types · Subtyping · Simulation · Simple grammars

Session types allow the specification of structured communication protocols on bidirectional, heterogeneously typed channels. Typically, these specifications include the type, direction and order of messages, as well as branching points where participants can decide how communication should proceed.

Support for session types in a programming language provides for safer concurrent programming by allowing its typechecker to validate the behavior of programs against the protocols specified by these types, ensuring session fidelity (communication proceeds as specified), privacy (channels are only known to participating processes), communication safety (no mismatch in direction or type of messages) and, in some variants, deadlock freedom.

Until recently, session types were bound by tail recursion and therefore restricted to the specification of protocols described by (the union of ω-regular and) regular languages. This class excludes many protocols of practical interest, with the quintessential example being the serialization of tree-structured data on a single channel. Context-free session types, proposed by Thiemann and Vasconcelos [5], liberate types from tail recursion by introducing a sequential composition operator with a monoidal structure and a left and right identity in type Skip, which represents no action. As their name hints, context-free session types can specify protocols corresponding to (simple deterministic) context-free languages and are thus considerably more expressive than their regular counterparts.

In their quest for safe concurrency, session type systems can become too rigid for practical use. Subtyping is meant to alleviate this tension between safety and flexibility. It is often justified by appealing to Liskov's *principle of safe substitution*: a type T can be considered a subtype of U if a value of type T can be used in place of a value of type U in whatever context, without violating the desirable properties of the type system in question.

What does it mean for a session type to be a subtype of another? One possible answer, based on the safe substitution of channels, can be found in Gay and Hole's seminal work on subtyping for regular session types [3]. Succinctly, their notion of subtyping allows increased flexibility in the interactions between session participants, namely on the type of messages they exchange and on the choices each participant has available at a branching points. A practical benefit of this flexibility is that it promotes *modular development*: the behaviour of one participant (e.g., a server) may be refined, while that of the other (e.g., a client) is kept intact.

While the algorithmic properties of subtyping for regular session types are by now well known, the same cannot be said for their context-free counterparts. The main challenges in this respect are the monoidal and distributive properties they exhibit, along with their equirecursive interpretation. A previous investigation [4] into this topic has shown this problem to be undecidable – the usual, unfortunate price to pay for expressive power.

Despite these challenges, we present a semantic approach to subtyping for context-free session types, supported by a novel kind of observational preorder we call $\mathcal{XYZW}$-simulation, which generalizes the notion of $\mathcal{XY}$-simulation proposed by Aarts and Vaandrager [1]. This relation allows us to selectively combine the requirements of simulation, reverse simulation and (a strong form of) contra-simulation in order to handle the covariant and contravariant properties of session type constructors to their full extent (previous work on this topic only accounted for subtyping with syntactically equal message types).

$\mathcal{XYZW}$-similarity generalizes bisimilarity, on which the notion of type equivalence for context-free session types is usually based. Taking advantage of this fact, we are able to derive a subtyping algorithm from an existing type equivalence algorithm, that of Almeida et al. [2]. The resulting algorithm is sound but, due to the undecidability of the subtyping problem, necessarily incomplete. Since the original algorithm reduces the type equivalence problem to the bisimilarity of simple grammars, our adaptation also results in an algorithm to check the $\mathcal{XYZW}$-similarity of simple grammars.

We implemented our algorithm in the FreeST programming language compiler, which natively supports context-free session types. In order to evaluate its performance, we ran it on a test suite comprised of 4000 subtyping pairs randomly generated with the aid of the Quickcheck library for Haskell. We obtained satisfactory performance, but observed 200 timeouts. To compare it with the original algorithm, we ran both side-by-side on a suite of 288 FreeST programs without subtyping. We observed no significant difference in performance. Together, these results suggest that our algorithm is viable for practical use.

References

1. Aarts, F., Vaandrager, F.: Learning I/O automata. In: CONCUR 2010 - Concurrency Theory. pp. 71–85. Springer Berlin Heidelberg, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15375-4_6
2. Almeida, B., Mordido, A., Vasconcelos, V.T.: Deciding the bisimilarity of context-free session types. In: Tools and Algorithms for the Construction and Analysis of Systems. pp. 39–56. Springer (2020). https://doi.org/10.1007/978-3-030-45237-7_3
3. Gay, S.J., Hole, M.: Subtyping for session types in the pi calculus. Acta Informatica **42**(2-3), 191–225 (2005). https://doi.org/10.1007/s00236-005-0177-z
4. Padovani, L.: Context-free session type inference. ACM Trans. Program. Lang. Syst. **41**(2), 9:1–9:37 (2019). https://doi.org/10.1145/3229062
5. Thiemann, P., Vasconcelos, V.T.: Context-free session types. In: Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, ICFP 2016, Nara, Japan, September 18-22, 2016. pp. 462–475. ACM (2016). https://doi.org/10.1145/2951913.2951926