Embedding Session Types in Haskell

(Tool presentation)

Sam Lindley J. Garrett Morris

The University of Edinburgh, UK

{Sam.Lindley,Garrett.Morris}@ed.ac.uk

Abstract

We present a novel embedding of session-typed concurrency in Haskell. We extend an existing HOAS embedding of linear λ-calculus with a set of core session-typed primitives, using indexed type families to express the constraints of the session typing discipline. We give two interpretations of our embedding, one in terms of GHC's built-in concurrency and another in terms of purely functional continuations. Our safety guarantees, including deadlock freedom, are assured statically and introduce no additional runtime overhead.

1. Introduction and Overview

Many communication protocols specify not just the types or formats of data or commands in the protocol, but also place restrictions on the order in which data is to be communicated. For example, the simple mail transfer protocol (SMTP) not only includes commands to specify the sender, recipients, and contents of an email message, but also requires that the sender command precede the recipient commands, which must in turn precede the commands giving the message body. Session types [3, 4, 8] capture such protocols in the types of communication channels. Session types have two distinguishing features. First, the endpoints of a channel must be given dual types: if one process expects to send a value along some channel, the process on the other end of the channel must expect to receive it. Second, session types must evolve over the course of a computation to prevent processes from repeating or skipping steps of the protocol.

Much of the existing work presents session types in the context of core concurrency-focused calculi (frequently based on either π-calculus or linear λ-calculus). Such calculi provide a holistic view of session types, integrating aspects of their syntax, the distinguishing aspects of the types themselves (such as duality), and their concurrent interpretations. However, typically they do not address how session types can be integrated into existing languages or the relationship between the concurrency expressed using session typing and that provided by existing concurrent primitives. We have developed a core session-typed functional calculus called GV [5, 6]. GV has strong connections to classical linear logic; consequently, its type system guarantees deadlock freedom in addition to typical safety properties. Our development of GV is also intended to be modular. We build on a standard linear λ-calculus, and attempt to minimize the number of concurrent features, preferring to express concurrent features in terms of λ-calculus constructs when possible. GV's metatheory is developed modularly as well; for example, this allows us to show that the addition of several non-logical features does not compromize GV's deadlock freedom, even though the extended calculus no longer enjoys a tight correspondence with classical linear logic.

In this work, we developed a parameterized tagless embedding [1, 2] of GV in Haskell and two implementations of that embedding. (We will use the term *parameterized tagless* or just *tagless* in preference to *finally tagless* or *tagless final*.) We began with an embedding of GV in Haskell, building on Polakow's [7] embedding of linear λ-calculus in Haskell. In doing so, we demonstrate the generality of Polakow's embedding: first, we are able to extend his core calculus with GV's concurrent primitives, and second, we are able to build a monadic interpretation of his embedding to support computations with side effects. Then, we built two implementations of our embedding, one based on the concurrent primitives in Haskell's IO monad and another that expresses concurrency using continuations. The former shows that this approach has practical applicability. We are able to wrap existing concurrent primitives with new type information, providing additional static safety guarantees without introducing runtime cost. The latter validates that our primitives also have a purely functional interpretation, following the formal semantics of GV. It also provides general insight into parameterized tagless embeddings and translations between them; in particular, while we are able to implement GV in terms of a more explicit language, Polarized GV, such an implementation requires limitations on the modularity of our source language.

The contributions of the work are as follows. First, we gave a monadic interpretation of Polakow's embedding of linear λ-calculus in Haskell. We gave an embedding of the syntax and typing of the core GV calculus in Haskell. We built two implementations of GV. The first uses the IO monad, and demonstrates that GV's static guarantees need introduce no runtime overhead. We also showed extensions of this embedding that increase its expressivity (at the cost of some of its static guarantees), demonstrating GV's modular nature. The second realizes the CPS semantics of GV in the continuation monad. The CPS semantics is non-parametric in that the translation of some term forms depends on the type at which they are used. To restore parametricity, we introduce a polarized version of the calculus. We then showed that we can implement the original language in terms of its polarized variant. These implementations show that GV concurrency can be used in a purely functional setting (or other setting in which using IO would be undesirable, such as STM), and show that our embeddings are are suitable for metaprogramming.

The code underlying this work is available at the following URL:

`http://github.com/jgbm/gvinhs/`

References

[1] R. Atkey, S. Lindley, and J. Yallop. Unembedding domain-specific languages. In S. Weirich, editor, *Proceedings of the 2nd ACM SIGPLAN Symposium on Haskell, Haskell 2009, Edinburgh, Scotland, UK, 3 September 2009*, pages 37–48. ACM, 2009.

[2] J. Carette, O. Kiselyov, and C. Shan. Finally tagless, partially evaluated: Tagless staged interpreters for simpler typed languages. *J. Funct. Program.*, 19(5):509–543, 2009.

[3] K. Honda. Types for dyadic interaction. In E. Best, editor, *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings*, volume 715 of *Lecture Notes in Computer Science*, pages 509–523. Springer, 1993.

[4] K. Honda, V. T. Vasconcelos, and M. Kubo. Language primitives and type discipline for structured communication-based programming. In C. Hankin, editor, *Programming Languages and Systems - ESOP'98, 7th European Symposium on Programming, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'98, Lisbon, Portugal, March 28 - April 4, 1998, Proceedings*, volume 1381 of *Lecture Notes in Computer Science*, pages 122–138. Springer, 1998.

[5] S. Lindley and J. G. Morris. A semantics for propositions as sessions. In J. Vitek, editor, *Programming Languages and Systems - 24th European Symposium on Programming, ESOP 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, volume 9032 of *Lecture Notes in Computer Science*, pages 560–584. Springer, 2015.

[6] S. Lindley and J. G. Morris. Talking bananas: Structural recursion for session types. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, ICFP 2016, Nara, Japan, September 19-21, 2016*. ACM, 2016.

[7] J. Polakow. Embedding a full linear lambda calculus in Haskell. In B. Lippmeier, editor, *Proceedings of the 8th ACM SIGPLAN Symposium on Haskell, Haskell 2015, Vancouver, BC, Canada, September 3-4, 2015*, pages 177–188. ACM, 2015.

[8] K. Takeuchi, K. Honda, and M. Kubo. An interaction-based language and its typing system. In C. Halatsis, D. G. Maritsas, G. Philokyprou, and S. Theodoridis, editors, *PARLE '94: Parallel Architectures and Languages Europe, 6th International PARLE Conference, Athens, Greece, July 4-8, 1994, Proceedings*, volume 817 of *Lecture Notes in Computer Science*, pages 398–413. Springer, 1994.