A Dependently-typed Intermediate Language with General Recursion

Foo Bar Baz

The University of Foo

{foo,bar,baz}@foo.edu

Abstract

This is gonna to be written later.

Categories and Subject Descriptors D.3.1 [*Programming Languages*]: Formal Definitions and Theory

General Terms Languages, Design

Keywords Dependent types, Intermediate langauge

1. Introduction

These are definitely drafts and only some main points are listed in each section.

a) Motivations:

- Because of the reluctance to introduce dependent types[1], the current intermediate language of Haskell, namely System F_C [9], separates expressions as terms, types and kinds, which brings complexity to the implementation as well as further extensions [11, 12].
- Popular full-spectrum dependently typed languages, like Agda, Coq, Idris, have to ensure the termination of functions for the decidability of proofs. No general recursion and the limitation of enforcing termination checking make such languages impractical for general-purpose programming.
- We would like to introduce a simple and compiler-friendly dependently typed core language with only one hierarchy, which supports general recursion at the same time.

b) Contribution:

- A core language based on Calculus of Constructions (CoC) that collapses terms, types and kinds into the same hierarchy.
- General recursion by introducing recursive types for both terms and types by the same μ primitive.
- Decidable type checking and managed type-level computation by replacing implicit conversion rule of CoC with generalized fold/unfold semantics.
- First-class equality by coercion, which is used for encoding GADTs or newtypes without runtime overhead.
- Surface language that supports datatypes, pattern matching and other language extensions for Haskell, and can be encoded into the core language.

c) Related work:

- Henk [3] and one of its implementation [5] show the simplicity of the Pure Type System (PTS). [6] also tries to combine recursion with PTS.
- Zombie [2, 7] is a language with two fragments supporting logics with non-termination. It limits the β-reduction for congruence closure [8].
- $\Pi\Sigma$ [1] is a simple, dependently-typed core language for expressing high-level constructions[2]. UHC compiler [4] tries to use a simplified core language with coercion to encode GADTs.
- System F_C [9] has been extended with type promotion [12] and kind equality [11]. The latter one introduces a limited form of dependent types into the system[3], which mixes up types and kinds.

2. Overview with examples

- Give a brief of overview of the surface language and its capability.
- Explain why this method has decidable type checking.
- Use examples to show the necessity of generalized fold and unfold.

3. The core langauge

- Give an overview of the core language and its syntax.
- Show the typing rules and operational semantics.
- The original formalization is suggested to rewrite using ott[4] which is a standard in academia. For example, the formalization

[1] This might be changed in the near future. See `https://ghc.haskell.org/trac/ghc/wiki/DependentHaskell/Phase1`.

[2] But the paper didn't give any meta-theories about the langauge.

[3] Richard A. Eisenberg is going to implement kind equality [11] into GHC. The implementation is proposed at `https://phabricator.haskell.org/D808` and related paper is at `http://www.cis.upenn.edu/~eir/papers/2015/equalities/equalities-extended.pdf`.

[4] `http://www.cl.cam.ac.uk/~pes20/ott/`

of GHC `https://github.com/ghc/ghc/tree/master/docs/core-spec`.

4. Metatheory

- Give formal proof of the soundness of the core language.
- Subject reduction and progress theorems will be proved.

5. Surface language

- Expand the core language with datatypes and pattern matching by encoding.
- Give translation rules.
- Encode GADTs and maybe other Haskell extensions.

6. Related work

7. Future work

Acknowledgments

Thanks to Blah. This work is supported by Blah.

References

[1] T. Altenkirch, N. A. Danielsson, A. Löh, and N. Oury. ΠΣ: Dependent types without the sugar. In *Functional and Logic Programming*, pages 40–55. Springer, 2010.

[2] C. Casinghino, V. Sjöberg, and S. Weirich. Combining proofs and programs in a dependently typed language. *ACM SIGPLAN Notices*, 49(1):33–45, 2014.

[3] S. P. Jones and E. Meijer. Henk: a typed intermediate language. 1997.

[4] A. Middelkoop, A. Dijkstra, and S. D. Swierstra. A lean specification for gadts: system f with first-class equality proofs. *Higher-Order and Symbolic Computation*, 23(2):145–166, 2010.

[5] J.-W. Roorda and J. Jeuring. Pure type systems for functional programming. 2007.

[6] P. G. Severi and F.-J. J. de Vries. Pure type systems with corecursion on streams: from finite to infinitary normalisation. In *ACM SIGPLAN Notices*, volume 47, pages 141–152. ACM, 2012.

[7] V. Sjöberg. *A Dependently Typed Language with Nontermination*. PhD thesis, University of Pennsylvania, 2015.

[8] V. Sjöberg and S. Weirich. Programming up to congruence. In *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, pages 369–382, New York, NY, USA, 2015. ACM. .

[9] M. Sulzmann, M. M. Chakravarty, S. P. Jones, and K. Donnelly. System f with type equality coercions. In *Proceedings of the 2007 ACM SIGPLAN international workshop on Types in languages design and implementation*, pages 53–66. ACM, 2007.

[10] J. C. Vanderwaart, D. Dreyer, L. Petersen, K. Crary, R. Harper, and P. Cheng. *Typed compilation of recursive datatypes*, volume 38. ACM, 2003.

[11] S. Weirich, J. Hsu, and R. A. Eisenberg. Towards dependently typed haskell: System fc with kind equality. In *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming, ICFP*, volume 13. Citeseer, 2013.

[12] B. A. Yorgey, S. Weirich, J. Cretin, S. Peyton Jones, D. Vytiniotis, and J. P. Magalhães. Giving haskell a promotion. In *Proceedings of the 8th ACM SIGPLAN workshop on Types in language design and implementation*, pages 53–66. ACM, 2012.

A. Appendix Title

Additional proof goes here.