A Case Study in Programming Coinductive Proofs in Beluga: Howe's Method

David Thibodeau [*]
McGill University
Montreal, Canada
david.thibodeau@mail.mcgill.ca

Alberto Momigliano
University of Milan
Milan, Italy
momigliano@di.unimi.it

Brigitte Pientka
McGill University
Montreal, Canada
bpientka@cs.mcgill.ca

Abstract

Bisimulation proofs play a central role in programming languages in establishing rich properties such as contextual equivalence. They are also challenging to mechanize, since they require a combination of inductive and coinductive reasoning on open terms. Recently, we have described the mechanization of the property that similarity in call-by-name PCF is a pre-congruence using Howe's method in the Beluga proof environment (Thibodeau et al. 2016b). The end result is succinct and elegant, thanks to the high-level abstractions and primitives Beluga provides to model variable binding, substitutions, and contexts, especially compared to similar proofs in the literature, carried out in Abella, Isabelle/HOL and Coq. We hence believe that this mechanization is a text book example that illustrates Belugas strength at mechanizing challenging (co)inductive proofs using higher-order abstract syntax encodings and may also be a good benchmark for comparing proof environments.

Categories and Subject Descriptors D.1.1 [*Programming Techniques*]: Applicative (Functional) Programming; D.3.1 [*Programming Languages*]: Formal Definitions and Theory; F.3.1 [*Theory of Computation*]: Specifying and Verifying and Reasoning about Programs

Keywords Dependent Types; Coinduction; Functional Programming; Logical Frameworks

Logical frameworks, such as the logical framework LF (Harper et al. 1993), provide a meta-language for representing formal systems given via axioms and inference rules together with proofs about them. In particular, they factor out common and recurring issues such as modelling variable bindings together with renaming and substitution operations. As users do not need to build up the basic mathematical infrastructure, they make it easier to prototype proof environments and mechanize formal systems together with their meta-theory. This also can have substantial benefits for proof checking and proof search.

Over the past decades we have made substantial progress to understand how to reason inductively about LF specifications: the proof environment Beluga, (Pientka and Dunfield 2010; Pientka and Cave 2015) based contextual LF (Nanevski et al. 2008; Pientka 2008), can be seen as the the legitimate successor of Twelf. Many case-studies (for example how to encode normalization proofs using logical relations (Cave and Pientka 2015)) have demonstrated the value of HOAS encodings and pushed the limits of what was thought to be possible when reasoning over HOAS representations. However, how to establish strong program properties such as contextual equivalence remained an open challenge. Contextual equivalence requires that equivalent programs may be used interchangeably in any larger program context with no observable difference. This property is typically difficult to prove directly, even on paper, since one has to consider every possible program context. A more tractable, equivalent notion to contextual equivalence is bisimulation. As contextual equivalence is generally intended to reason about programs, it is usually a congruence. Hence a sound and complete bisimilarity also has to be a congruence. Proving congruence is thus a crucial step when working with contextual equivalence of programs.

In this talk, we discuss a text book example, the property that similarity in the call-by-name PCF is a pre-congruence using Howe's method (Howe 1996). Howes method is a powerful approach to show that a (bi)similarity is a congruence. In a nutshell, it reverses the problem: first define a relation, called Howes closure, that includes the (bi)similarity of interest and is almost a congruence. Second, show it is a (bi)simulation. As (bi)similarity contains every (bi)simulations, Howes closure is thus included in (bi)similarity. Third, conclude that the (bi)similarity and its Howes closure coincide, thus the former is a congruence. For simplicity, we concentrate on the notion of similarity from which we can obtain bisimilarity.

Proving congruence properties of (bi)similarity using Howe's method is a challenging benchmark for proof environments: the notion of (bi)similarity is typically defined coinductively and must be stated for closed and open terms and Howe's relation is an inductive definition on open terms. Modelling the scope of variables and reasoning correctly but succinctly about closed and open terms is therefore crucial. We mechanize this proof in Beluga. Our formal development relies on three key ingredients:

1. we give a HOAS encoding of lambda-terms together with their operational semantics as intrinsically typed terms, thereby avoiding not only the need to deal with binders, renaming and substitutions, but keeping all typing invariants implicit;

[*] This author acknowledges funding from the Fonds Québécois de Recherche sur la Nature et les Technologies (FQRNT).

2. we take advantage of Beluga's support for representing open terms using first-class contexts and simultaneous substitutions (Cave and Pientka 2012, 2013): this allows us to directly state a notion such as open simulation without resorting to the usual inductive closure operation and to encode very elegantly notoriously painful proofs such as the substitutivity of the Howe relation;
3. we exploit the possibility of reasoning by coinduction in Beluga's reasoning logic (Thibodeau et al. 2016a).

The end result is succinct and elegant, thanks to the high-level abstractions and primitives Beluga provides, as it can be seen by comparing this proof with a similar one, carried out in Abella (Momigliano 2012), and less related developments in Isabelle/HOL (Ambler and Crole 1999) and Coq (Honsell et al. 2001). We hence believe that this mechanization is a text book example that illustrates Beluga's strength at mechanizing challenging (co)inductive proofs using higher-order abstract syntax encodings. More generally, it demonstrates that the right abstractions and primitives are key in bringing down the cost of verification.

References

S. Ambler and R. L. Crole. Mechanized operational semantics via (co)induction. In Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Théry, editors, *TPHOLs*, volume 1690 of *Lecture Notes in Computer Science*, pages 221–238. Springer, 1999. ISBN 3-540-66463-7.

A. Cave and B. Pientka. Programming with binders and indexed data-types. In *39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'12)*, pages 413–424. ACM Press, 2012.

A. Cave and B. Pientka. First-class substitutions in contextual type theory. In *8th ACM SIGPLAN International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP'13)*, pages 15–24. ACM Press, 2013.

A. Cave and B. Pientka. A case study on logical relations using contextual types. In I. Cervesato and K.Chaudhuri, editors, *10th International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP'15)*, pages 18–33. Electronic Proceedings in Theoretical Computer Science (EPTCS), 2015.

R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, January 1993.

F. Honsell, M. Miculan, and I. Scagnetto. Π-calculus in (co)inductive type theories. *Theoretical Computer Science*, 2(253):239–285, 2001.

D. J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Computation*, 124(2):103–112, 1996.

A. Momigliano. A supposedly fun thing I may have to do again: A HOAS encoding of Howe's method. In *Proceedings of the Seventh International Workshop on Logical Frameworks and Meta-languages, Theory and Practice*, LFMTP '12, pages 33–42, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1578-4. URL `http://doi.acm.org/10.1145/2364406.2364411`.

A. Nanevski, F. Pfenning, and B. Pientka. Contextual modal type theory. *ACM Transactions on Computational Logic*, 9(3):1–49, 2008.

B. Pientka. A type-theoretic foundation for programming with higher-order abstract syntax and first-class substitutions. In *35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'08)*, pages 371–382. ACM Press, 2008.

B. Pientka and A. Cave. Inductive Beluga: Programming Proofs (System Description). In A. P. Felty and A. Middeldorp, editors, *25th International Conference on Automated Deduction (CADE-25)*, Lecture Notes in Computer Science (LNCS 9195), pages 272–281. Springer, 2015.

B. Pientka and J. Dunfield. Beluga: a framework for programming and reasoning with deductive systems (System Description). In J. Giesl and R. Haehnle, editors, *5th International Joint Conference on Automated Reasoning (IJCAR'10)*, Lecture Notes in Artificial Intelligence (LNAI 6173), pages 15–21. Springer, 2010.

D. Thibodeau, A. Cave, and B. Pientka. Indexed codata types. In J. Garrigue, G. Keller, and E. Sumii, editors, *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, ICFP 2016, Nara, Japan, September 18-22, 2016*, pages 351–363. ACM, 2016a. URL `http://doi.acm.org/10.1145/2951913.2951929`.

D. Thibodeau, A. Momigliano, and B. Pientka. A case-study in programming coinductive proofs: Howe's method. Technical report, McGill University, Nov 2016b.