Compositional Coinduction with Sized Types

Andreas Abel

Department of Computer Science and Engineering, Gothenburg University,
Rännvägen 6, 41296 Göteborg, Sweden
andreas.abel@gu.se

Proofs by induction on some inductively defined structure, e. g., finitely-branching trees, may appeal to the induction hypothesis at any point in the proof, provided the induction hypothesis is only used for immediate substructures, e. g., the subtrees of the node we are currently considering in the proof. The basic principle of structural induction can be relaxed to course-of-value induction, which allows application of the induction hypothesis also to non-immediate substructures, like any proper subtree of the current tree. If course-of-value induction is not sufficient yet, we can resort to define a well-founded relation on the considered structure and use the induction hypothesis for any substructure which is strictly smaller with regard to the constructed relation. At a closer look, however, this well-founded induction is just *structural induction on the derivation* of being strictly smaller. This means that in a logical system that allows us to construct inductive predicate and relations, such as, e. g., Martin-Löf Type Theory (Nordström et al., 1990) or the Calculus of Inductive Constructions (Paulin-Mohring, 1993), structural induction is complete for any kind of inductive proof.

In all these flavors of induction, validity of induction hypothesis application can be checked easily and locally, independent of its context. In proof assistants, in principle a *structural termination checker* (Giménez, 1995; Abel, 2000) suffices[1] to check such inductive proofs, which looks at the proof tree, extracts all calls to the induction hypotheses, and checks that they happen only on structurally smaller arguments. In practice, mutual induction is supported as well, based on a simple static call graph analysis (Abel and Altenkirch, 2002; Barras, 2010; Ben-Amram, 2008; Hyvernat, 2014).

Dually to structural induction, in a coinductive proof of a proposition defined as the greatest fixed-point of a set of rules, we may appeal to the coinduction hypothesis to fill the premises of the rule we have chosen to prove our goal. For instance, two infinite streams may be defined to be bisimilar if their heads are equal and their tails are bisimilar, coinductively. Our goal might be to show that bisimilarity is reflexive, i. e., any stream is bisimilar to itself. To establish bisimilarity, we use the sole rule with the first subgoal to show that the head of the stream is equal to itself. After this breathtaking enterprise, we are left with the second subgoal to show that the tail of the stream is bisimilar to itself, which we solve by appeal to the coinduction hypothesis. At this point, it is worth noting that not the stream got smaller (the tail of an infinite stream is still infinite), but the coinductive hypothesis is guarded by a rule application. This means that the coinductive proof can unfold into a possibly infinitely deep derivation without getting into a "busy loop", meaning the proof is *productive*.

[1] Even for induction, type-based termination offers significant advantages for compositionality and robustness, as argued, e. g., by Barras and Sacchini (2013). However, at this point there is no mature implementation in proof assistants based on dependent types.

In a similar way as for induction, we seek to relax the criterion for well-formed coinductive proofs, which states that only the immediate subgoals of the final rule application can be filled by the coinductive hypothesis. We can allow several rule applications until we reach the coinductive hypothesis from the root of the derivation. This is dual to course-of-value induction and could be called *guarded coinduction*.[2]

In contrast to induction, checking the validity of calls to the coinduction hypothesis requires us to look at the *context* of the calls rather than the call arguments. We have to check that the calls to the coinductive hypotheses happen in a *constructor* context, i. e., a context of coinductive rule applications only. This lack of locality also leads to a loss of compositionality of proofs by guarded coinduction. For instance, consider a coinductive proof of the bisimilarity of two streams through a bisimilarity chain, i. e., via some intermediate streams and the use of transitivity of bisimilarity. Transitivity is not a constructing rule for bisimilarity,[3] but an admissible rule proven by coinduction. As transitivity is not a constructor, we cannot use the coinduction hypothesis under transitivity nodes in the proof tree. In practice, often a severe restructuring of a natural informal proof is necessary to make it guarded and please a structural guardedness checker. The resulting proofs may be highly non-compositional and bloated, especially if proofs of previous lemmata have to be inlined and fused into the current proof.

To regain compositionality, we have to relax the contexts of coinductive hypothesis applications to include admissible rules and lemma invocation in general, without jeopardizing productivity. Such contexts need to produce one more rule constructor than they consume, which must be easily verifiable by the productivity checker. Sized types (Hughes et al., 1996; Amadio and Coupet-Grimal, 1998; Barthe et al., 2004; Abel, 2008; Sacchini, 2013) offer the necessary technology. Coinductive types, propositions, and relations are parameterized by an ordinal $i \leq \omega$ which denotes the minimum *definedness depth* of their derivations. Semantically, this idea is already present in Mendler's work (Mendler et al., 1986; Mendler, 1991), and it is implicit in the principle of ordinal iteration to construct the greatest fixed point of a monotone operator F. We define the approximants $\nu^i F$ of the greatest fixed-point $\nu^\omega F$ by induction on ordinal i as follows:

$$\begin{array}{ll} \nu^0 & F = \top \\ \nu^{i+1} & F = F\,(\nu^i F) \\ \nu^\omega & F = \bigsqcap_{i<\omega} \nu^i F \end{array}$$

For monotone F, we obtain a descending chain $\nu^0 F \sqsupseteq \nu^1 F \sqsupseteq \cdots \sqsupseteq \nu^\omega F$. The greatest fixed point of F is reached at stage ω if F is continuous in the sense that $\bigsqcap_{i\in I} F(A_i) \sqsubseteq F(\bigsqcap_{i\in I} A_i)$. For instance, all strictly positive type transformers correspond to continuous operators (Abel, 2003, Theorem 1).

[2] Coquand (1994) calls it *guarded induction*, Giménez (1995) *guarded by constructors*.

[3] In fact, it is well-known that every coinductive relation with a transitivity rule is is trivial, i. e., the total relation. The proof of relatedness of arbitrary objects is just the infinite tree all of whose nodes are applications of the transitivity rule. This problem can be overcome with mixed coinductive-inductive types (Abel, 2007; Nakata and Uustalu, 2010), to allow only finitely many applications of the transitivity rule in a row (Danielsson and Altenkirch, 2010).

An alternative construction of the greatest fixed-point uses deflationary iteration (Sprenger and Dam, 2003; Abel, 2012; Abel and Pientka, 2013),

$$\nu^i F = \bigsqcap_{j<i} F\,(\nu^j F)$$

which gives a descending chain without the monotonicity of F. However, the same conditions on F are needed to reach the fixed point at stage ω.

Giving names to the approximants $\nu^i F$ of coinductive type $\nu^\omega F$, we can express through the type system when a term t, which acts as the context for the coinductive hypothesis, produces one more constructor than it consumes: it needs to have type $\forall i.\ \nu^i F \to \nu^{i+1} F$ polymorphic in "size" (depth) i. Such a context is called *guarding*. A weaker, but very common and useful property of a function t is to be *guardedness preserving*, i. e., having type $\forall i.\ \nu^i F \to \nu^i F$. For instance, consider bisimilarity on streams, which is defined using relation transformer $F(X)(x,y) = (\mathsf{head}\,x \equiv \mathsf{head}\,y) \times X\,(\mathsf{tail}\,x)\,(\mathsf{tail}\,y)$. The symmetry lemma of bisimilarity $\forall i.\ \nu^i F(x,y) \to \nu^i F(y,x)$ is guardedness preserving: to produce one constructor of the requested bisimilarity derivation, it only needs to inspect the last constructor of the given bisimilarity derivation. Analogously, transitivity of bisimilarity receives type $\forall i.\ \nu^i F(x,y) \to \nu^i F(y,z) \to \nu^i F(x,z)$. Here, to produce the last rule of the output derivations we only need to inspect the last rule of the two input derivations. This typing allows us to freely use transitivity in coinductive proofs without jeopardizing the validity of the coinductive hypothesis.

Tracking guardedness levels in the type systems through "sized" coinductive types gives us *compositional* coinduction, as we can freely abstract out, for instance, guardedness-preserving contexts without upsetting some structural productivity checker who can only deal with concretely given code.

Recently, guardedness-preserving functions have been rediscovered in the context of Isabelle/HOL as *friendly operations* (Blanchette et al., 2015). Prototypical implementations of type-based termination with sized types exist for Coq (Sacchini, 2015) and Agda. The latter has been subjected to a larger case study: an implementation of normalization by evaluation using the coinductive delay monad (Abel and Chapman, 2014). Compositional coinduction with sized types seems to be a promising alternative to parameterized coinduction (Hur et al., 2013) and up-to techniques (Pous and Sangiorgi, 2012).

References

Abel, A.: Specification and verification of a formal system for structurally recursive functions. In: Coquand, T., Dybjer, P., Nordström, B., Smith, J. (eds.) Types for Proofs and Programs, International Workshop TYPES'99, Lökeberg, Sweden, June 12-16, 1999, Selected Papers. Lecture Notes in Computer Science, vol. 1956, pp. 1–20. Springer (2000), http://dx.doi.org/10.1007/3-540-44557-9_1

Abel, A.: Termination and guardedness checking with continuous types. In: Hofmann, M. (ed.) Typed Lambda Calculi and Applications, 6th International Conference, TLCA 2003, Valencia, Spain, June 10-12, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2701, pp. 1–15. Springer (2003), http://dx.doi.org/10.1007/3-540-44904-3_1

Abel, A.: Mixed inductive/coinductive types and strong normalization. In: Shao, Z. (ed.) Programming Languages and Systems, 5th Asian Symposium, APLAS 2007, Singapore, November 29-December 1, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4807, pp. 286–301. Springer (2007), http://dx.doi.org/10.1007/978-3-540-76637-7_19

Abel, A.: Semi-continuous sized types and termination. Logical Methods in Computer Science 4(2:3), 1–33 (2008), http://dx.doi.org/10.2168/LMCS-4(2:3)2008, CSL'06 special issue.

Abel, A.: Type-based termination, inflationary fixed-points, and mixed inductive-coinductive types. In: Miller, D., Ésik, Z. (eds.) Proceedings of the 8th Workshop on Fixed Points in Computer Science (FICS 2012). Electronic Proceedings in Theoretical Computer Science, vol. 77, pp. 1–11 (2012), http://dx.doi.org/10.4204/EPTCS.77.1, invited talk

Abel, A., Altenkirch, T.: A predicative analysis of structural recursion. Journal of Functional Programming 12(1), 1–41 (2002), http://dx.doi.org/10.1017/S0956796801004191

Abel, A., Chapman, J.: Normalization by evaluation in the delay monad: A case study for coinduction via copatterns and sized types. In: Levy, P., Krishnaswami, N. (eds.) Proceedings 5th Workshop on Mathematically Structured Functional Programming, MSFP 2014, Grenoble, France, 12 April 2014. Electronic Proceedings in Theoretical Computer Science, vol. 153, pp. 51–67 (2014), http://dx.doi.org/10.4204/EPTCS.153.4

Abel, A., Pientka, B.: Wellfounded recursion with copatterns: A unified approach to termination and productivity. In: Morrisett, G., Uustalu, T. (eds.) Proceedings of the Eighteenth ACM SIGPLAN International Conference on Functional Programming, ICFP'13, Boston, MA, USA, September 25-27, 2013. pp. 185–196. ACM Press (2013), http://doi.acm.org/10.1145/2500365.2500591

Amadio, R.M., Coupet-Grimal, S.: Analysis of a guard condition in type theory (extended abstract). In: Nivat, M. (ed.) Foundations of Software Science and Computation Structure, First International Conference, FoSSaCS'98, 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. Lecture Notes in Computer Science, vol. 1378, pp. 48–62. Springer (1998), http://dx.doi.org/10.1007/BFb0053541

Barras, B.: The syntactic guard condition of coq. Talk at the Journée "égalité et terminaison" du 2 février 2010 in conjunction with JFLA 2010 (2010), http://coq.inria.fr/files/adt-2fev10-barras.pdf

Barras, B., Sacchini, J.L.: Type-based methods for termination and productivity in Coq. In: Mahboubi, A., Tassi, E. (eds.) The 5th Coq Workshop, A satellite workshop of ITP 2013, Rennes, July 22nd (2013), https://coq.inria.fr/coq-workshop/2013#Sacchini

Barthe, G., Frade, M.J., Giménez, E., Pinto, L., Uustalu, T.: Type-based termination of recursive definitions. Mathematical Structures in Computer Science 14(1), 97–141 (2004), http://dx.doi.org/10.1017/S0960129503004122

Ben-Amram, A.M.: Size-change termination with difference constraints. ACM Transactions on Programming Languages and Systems 30(3) (2008), http://doi.acm.org/10.1145/1353445.1353450

Blanchette, J.C., Popescu, A., Traytel, D.: Foundational extensible corecursion: a proof assistant perspective. In: Fisher, K., Reppy, J.H. (eds.) Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming, ICFP 2015, Vancouver, BC, Canada, September 1-3, 2015. pp. 192–204. ACM Press (2015), http://doi.acm.org/10.1145/2784731.2784732

Coquand, T.: Infinite objects in type theory. Lecture Notes in Computer Science, vol. 806, pp. 62–78. Springer (1994), http://dx.doi.org/10.1007/3-540-58085-9_72

Danielsson, N.A., Altenkirch, T.: Subtyping, declaratively. In: Bolduc, C., Desharnais, J., Ktari, B. (eds.) Mathematics of Program Construction, 10th International Conference, MPC 2010, Québec City, Canada, June 21-23, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6120, pp. 100–118. Springer (2010), http://dx.doi.org/10.1007/978-3-642-13321-3_8

Giménez, E.: Codifying guarded definitions with recursive schemes. In: Dybjer, P., Nordström, B., Smith, J. (eds.) Types for Proofs and Programs, International Workshop TYPES'94, Båstad, Sweden, June 6-10, 1994, Selected Papers. Lecture Notes in Computer Science, vol. 996, pp. 39–59. Springer (1995), http://dx.doi.org/10.1007/3-540-60579-7_3

Hughes, J., Pareto, L., Sabry, A.: Proving the correctness of reactive systems using sized types. In: Boehm, H.J., Jr., G.L.S. (eds.) Conference Record of POPL'96: The 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Papers Presented at the Symposium, St. Petersburg Beach, Florida, USA, January 21-24, 1996. pp. 410–423. ACM Press (1996), http://doi.acm.org/10.1145/237721.240882

Hur, C., Neis, G., Dreyer, D., Vafeiadis, V.: The power of parameterization in coinductive proof. In: Giacobazzi, R., Cousot, R. (eds.) The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'13, Rome, Italy, January 23 - 25, 2013. pp. 193–206. ACM Press (2013), http://doi.acm.org/10.1145/2429069.2429093

Hyvernat, P.: The size-change termination principle for constructor based languages. Logical Methods in Computer Science 10(1) (2014), http://dx.doi.org/10.2168/LMCS-10(1:11)2014

Mendler, N.P., Panangaden, P., Constable, R.L.: Infinite objects in type theory. In: Proceedings, Symposium on Logic in Computer Science, 16-18 June 1986, Cambridge, Massachusetts, USA. pp. 249–255. IEEE Computer Society (1986)

Mendler, N.P.: Inductive types and type constraints in the second-order lambda calculus. Annals of Pure and Applied Logic 51(1–2), 159–172 (1991), http://dx.doi.org/10.1016/0168-0072(91)90069-X

Nakata, K., Uustalu, T.: Resumptions, weak bisimilarity and big-step semantics for while with interactive I/O: an exercise in mixed induction-coinduction. In: Aceto, L., Sobocinski, P. (eds.) Proceedings Seventh Workshop on Structural Operational Semantics, SOS 2010, Paris, France, 30 August 2010. Electronic Proceedings in Theoretical Computer Science, vol. 32, pp. 57–75 (2010), http://dx.doi.org/10.4204/EPTCS.32.5

Nordström, B., Petersson, K., Smith, J.M.: Programming in Martin Löf's Type Theory: An Introduction. Clarendon Press, Oxford (1990), http://www.cs.chalmers.se/Cs/Research/Logic/book/

Paulin-Mohring, C.: Inductive definitions in the system Coq - rules and properties. In: Bezem, M., Groote, J.F. (eds.) Typed Lambda Calculi and Applications, International Conference on Typed Lambda Calculi and Applications, TLCA '93, Utrecht, The Netherlands, March 16-18, 1993, Proceedings. Lecture Notes in Computer Science, vol. 664, pp. 328–345. Springer (1993), http://dx.doi.org/10.1007/BFb0037116

Pous, D., Sangiorgi, D.: Enhancements of the bisimulation proof method. In: Sangiorgi, D., Rutten, J. (eds.) Advanced Topics in Bisimulation and Coinduction. Cambridge University Press (2012)

Sacchini, J.: Coq^: Type-based termination in the Coq proof assistant (2015), project description, http://qatar.cmu.edu/~sacchini/coq.html

Sacchini, J.L.: Type-based productivity of stream definitions in the calculus of constructions. In: 28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013. pp. 233–242. IEEE Computer Society Press (2013), http://dx.doi.org/10.1109/LICS.2013.29

Sprenger, C., Dam, M.: On the structure of inductive reasoning: Circular and tree-shaped proofs in the μ-calculus. In: Gordon, A.D. (ed.) Foundations of Software Science and Computational Structures, 6th International Conference, FOSSACS 2003 Held as Part of the Joint European

Conference on Theory and Practice of Software, ETAPS 2003, Warsaw, Poland, April 7-11, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2620, pp. 425–440. Springer (2003), http://dx.doi.org/10.1007/3-540-36576-1_27