On the Proof Theory of Property-Based Testing of Coinductive Specifications, or: PBT to Infinity and beyond

Roberto Blanco
INRIA Paris, France

Dale Miller
INRIA Saclay
LIX, École Polytechnique, France

Alberto Momigliano
DI, Università degli Studi di Milano, Italy

Reasoning about infinite computations via coinduction and corecursion has an ever-increasing relevance in formal methods and, in particular, in the semantics of programming languages, starting from [19] (see [14] for a compelling example) and, of course, coinduction underlies (the meta-theory of) process calculi. This importance was acknowledged by researchers in proof assistants, who promptly provided support for coinduction and corecursion from the early '90s on: see [22, 11] for the beginning of the story concerning two popular frameworks.

It also became apparent that tools that search for refutations/counter-examples of conjectures before attempting a formal proof are invaluable: this is particularly true in PL theory, where proofs tend to be shallow but may have hundreds of cases. One such approach is *property-based testing* (PBT), which employs automatic test data generation to try and refute executable specifications. Pioneered by *QuickCheck* for functional programming [9], it has now spread to most major proof assistants [6, 21].

In general, PBT does not extend immediately to testing coinductive specifications (an exception being Isabelle's *Nitpick*, which is, however, a counter-model generator). Extending Coq's *QuickChick* [21] to deal with Coq's notion of coinduction via *guarded* recursion (which is generally seen to be a less than satisfactory approach to coinduction) is particularly challenging. We are not aware of applications of PBT to other forms of coinduction, such as *co-patterns* [1].

While PBT originated in the functional programming community, we have given in a previous paper ([7]) a reconstruction of some of its features (operational semantics, different flavors of generation, shrinking) in proof-theoretic terms using *focused proof systems* [2, 15] and *Foundational Proof Certificates* (FPC) [8]. An FPC can be used to define a range of proof structures, such as resolution refutations, Herbrand disjuncts, tableaux, etc. In the context of PBT, the proof theory setup is rather simple. Consider an attempt to find counter-examples to a conjecture of the form $\forall x[(\tau(x) \wedge P(x)) \supset Q(x)]$ where τ is a typing predicate and P and Q are two other predicates defined using Horn clause specifications. The negation of this conjecture is $\exists x[(\tau(x) \wedge P(x)) \wedge \neg Q(x)]$. In searching for a focused proof of this negation, the *positive phase* (which corresponds to the generation of possible counter-examples) is represented by $\exists x$ and $(\tau(x) \wedge P(x))$. That phase is followed by the *negative phase* (which correspond to counter-example testing) and is represented by $\neg Q(x)$. FPCs are simple logic programs that guide the search for potential counter-examples using different generation strategies; they further capture diverse features such as δ-debugging, fault isolation, explanation, etc. Such a range of features can be programmed as the *clerks and experts* predicates [8] that decorate the sequent rules used in an FPC proof checking kernel: the kernel is also able to do a limited amount of proof reconstruction.

There are at least two ways to address potentially infinite computations in logical terms. We can introduce infinite terms, rational or even irrational, in our semantics, as already accounted for in Lloyd's textbook and in [16]: this has recently been revisited in *coinductive logic programming*, see, for example, [24] and the definitive [5]. Or we can concentrate on modeling infinite *behavior* of finite terms, for

Submitted to:
LFMTP 2020

example, divergence of a given (finite) program. We choose the latter and this requires a logic stronger than a logic programming interpreter, namely one with explicit rules for induction and coinduction.

A natural choice for such a logic is the fixed point logic $\mathcal{G}$ [10] and its linear logic cousin μMALL [3], which are associated to the *Abella* proof assistant [4] and the *Bedwyr* model-checker. In fact, the latter has already been used for related aims [12].

For a concrete example, consider a *coinductive* definition for CBV evaluation in the λ-calculus with constants (following [14]). Using Bedwyr's concrete syntax, this is written as:

```
Define coinductive coeval: tm -> tm -> prop by
 coeval (con C) (con C);
 coeval (fun R) (fun R);
 coeval (app M N) V :=
   exists R W, coeval M (fun R) /\ coeval N W /\ coeval (R W) V.
```

As is well-known, co-evaluation fails to be deterministic, since a divergent term such as Ω co-evaluates to anything. We can confirm this by searching for a proof of the following formula:

```
exists E V1 V2, coeval E V1 /\ coeval E V2 /\ (V1 = V2 -> false)
```

Proving this query entails a way to generate such (finite) terms and then checking, with a crucial appeal to the coinduction, that they are in the required relation.

Other applications of PBT include separating the various notions of equivalences in the lambda-calculus and various process calculi: for example, applicative and ground similarity in PCFL [23], or analogous standard results in the π-calculus. While similar goals have been achieved in the literature for labeled transition systems (using, for example, the *Concurrency Workbench*), it is a remarkable feature of the proof-theoretic setting that we can generalize PBT from a system without bindings (say, CCS) to a system with bindings (say, the π-calculus). Such ease is possible since proof theory accommodates the *λ-tree syntax* approach to treating bindings [17]: in particular, both Abella and Bedwyr include the ∇ quantifier [18].

In our current setup, we attempt to find counter-examples using Bedwyr to execute both the generation of test cases (controlled by using specific FPCs [7]) and the testing phase. Such an implementation of PBT allows us to piggyback on Bedwyr's facilities for efficient proof search via tabling for (co)inductive predicates. The treatment of the negation in the testing phase is, as usual, a sticky point [20]. However, if we identify, as we do, the proof theory behind model checking as based on the linear logic μMALL [13], in that setting, occurrences of negations can be eliminated by using De Morgan duality and inequality.

References

[1] Andreas Abel, Brigitte Pientka, David Thibodeau & Anton Setzer (2013): *Copatterns: programming infinite structures by observations*. In: *POPL*, ACM, pp. 27–38.

[2] Jean-Marc Andreoli (1992): *Logic Programming with Focusing Proofs in Linear Logic*. *J. of Logic and Computation* 2(3), pp. 297–347, doi:10.1093/logcom/2.3.297.

[3] David Baelde (2012): *Least and Greatest Fixed Points in Linear Logic*. *ACM Trans. Comput. Log.* 13(1), pp. 2:1–2:44.

[4] David Baelde, Kaustuv Chaudhuri, Andrew Gacek, Dale Miller, Gopalan Nadathur, Alwen Tiu & Yuting Wang (2014): *Abella: A System for Reasoning about Relational Specifications*. *Journal of Formalized Reasoning* 7(2).

[5] Henning Basold, Ekaterina Komendantskaya & Yue Li (2019): *Coinduction in Uniform: Foundations for Corecursive Proof Search with Horn Clauses*. In: *ESOP*, *Lecture Notes in Computer Science* 11423, Springer, pp. 783–813.

[6] Jasmin Christian Blanchette, Lukas Bulwahn & Tobias Nipkow (2011): *Automatic Proof and Disproof in Isabelle/HOL*. In Cesare Tinelli & Viorica Sofronie-Stokkermans, editors: *FroCoS*, *Lecture Notes in Computer Science* 6989, Springer, pp. 12–27. Available at `http://dx.doi.org/10.1007/978-3-642-24364-6_2`.

[7] Roberto Blanco, Dale Miller & Alberto Momigliano (2019): *Property-Based Testing via Proof Reconstruction*. In: *PPDP*, ACM, pp. 5:1–5:13.

[8] Zakaria Chihani, Dale Miller & Fabien Renaud (2017): *A semantic framework for proof evidence*. *J. of Automated Reasoning* 59(3), pp. 287–330, doi:10.1007/s10817-016-9380-6.

[9] Koen Claessen & John Hughes (2000): *QuickCheck: a lightweight tool for random testing of Haskell programs*. In: *Proceedings of the 2000 ACM SIGPLAN International Conference on Functional Programming (ICFP 2000)*, ACM, pp. 268–279.

[10] Andrew Gacek, Dale Miller & Gopalan Nadathur (2011): *Nominal abstraction*. *Information and Computation* 209(1), pp. 48–73, doi:10.1016/j.ic.2010.09.004.

[11] Eduardo Giménez (1994): *Codifying Guarded Definitions with Recursion Schemes*. In P. Dybjer & B. Nordström, editors: *Selected Papers 2nd Int. Workshop on Types for Proofs and Programs, TYPES'94, Båstad, Sweden, 6–10 June 1994*, *Lecture Notes in Computer Science* 996, Springer-Verlag, Berlin, pp. 39–59.

[12] Quentin Heath & Dale Miller (2015): *A framework for proof certificates in finite state exploration*. In: *PxTP@CADE*, *EPTCS* 186, pp. 11–26.

[13] Quentin Heath & Dale Miller (2019): *A Proof Theory for Model Checking*. *J. Autom. Reasoning* 63(4), pp. 857–885.

[14] Xavier Leroy & Hervé Grall (2009): *Coinductive big-step operational semantics*. *Information and Computation* 207(2), pp. 284–304. Available at `http://gallium.inria.fr/~xleroy/publi/coindsem-journal.pdf`.

[15] Chuck Liang & Dale Miller (2009): *Focusing and Polarization in Linear, Intuitionistic, and Classical Logics*. *Theoretical Computer Science* 410(46), pp. 4747–4768.

[16] Michael J. Maher (1988): *Complete Axiomatizations of the Algebras of Finite Rational and Infinite Trees*. In: *3nd Symp. on Logic in Computer Science*, pp. 348–357.

[17] Dale Miller (2018): *Mechanized Metatheory Revisited*. *Journal of Automated Reasoning*, doi:10.1007/s10817-018-9483-3.

[18] Dale Miller & Alwen Tiu (2005): *A proof theory for generic judgments*. *ACM Trans. on Computational Logic* 6(4), pp. 749–783.

[19] Robin Milner & Mads Tofte (1991): *Co-induction in Relational Semantics*. *Theoretical Computer Science* 87(1), pp. 209–220.

[20] Alberto Momigliano (2000): *Elimination of Negation in a Logical Framework*. In: *CSL*, *Lecture Notes in Computer Science* 1862, Springer, pp. 411–426.

[21] Zoe Paraskevopoulou, Catalin Hritcu, Maxime Dénès, Leonidas Lampropoulos & Benjamin C. Pierce (2015): *Foundational Property-Based Testing*. In Christian Urban & Xingyuan Zhang, editors: *Interactive Theorem Proving - 6th International Conference, ITP 2015, Proceedings*, *Lecture Notes in Computer Science* 9236, Springer, pp. 325–343.

[22] Lawrence C. Paulson (1997): *Mechanizing Coinduction and Corecursion in Higher-order Logic*. *Journal of Logic and Computation* 7(2), pp. 175–204.

[23] A. M. Pitts (1997): *Operationally Based Theories of Program Equivalence*. In P. Dybjer & A. M. Pitts, editors: *Semantics and Logics of Computation*.

[24] Luke Simon, Ajay Bansal, Ajay Mallya & Gopal Gupta (2007): *Co-Logic Programming: Extending Logic Programming with Coinduction*. In Lars Arge, Christian Cachin, Tomasz Jurdziński & Andrzej Tarlecki, editors: *Automata, Languages and Programming*, Springer Berlin Heidelberg, pp. 472–483.