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 ([17]) a reconstruction of some of its features (operational semantics, different flavors of generation, shrinking) in proof-theoretic terms using focused proof systems [2] 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 \[\forall x \left( \tau(x) \land P(x) \right) \Rightarrow Q(x)\] where \(\tau\) is a typing predicate and \(P\) and \(Q\) are two other predicates defined using Horn clause specifications. The negation of this conjecture is \[\exists x \left( \tau(x) \land P(x) \right) \land \neg Q(x)\]. In searching for a focused proof of this negation, the positive phase (which corresponds to the generation of possible counter-examples) is represented by \(\exists x \left( \tau(x) \land P(x) \right)\). 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 \(\delta\)-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
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 $G$ [10] and its linear logic cousin $\mu$MALL [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 $\lambda$-calculus with
constants (following [14]). Using Bedwyr’s concrete syntax, this is written as:

Define coinductive coeval: tm $\rightarrow$ tm $\rightarrow$ prop by

coeval (con C) (con C);
coeval (fun R) (fun R);
coeval (app M N) V :=
exists R W, coeval M (fun R) $\lor$ coeval N W $\lor$ coeval (R W) V.

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

exists E V1 V2, coeval E V1 $\lor$ coeval E V2 $\lor$ (V1 = V2 $\rightarrow$ 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 $\pi$-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 $\pi$-calculus). Such ease is possible since proof theory accommodates the
$\lambda$-tree syntax approach to treating bindings [17]: in particular, both Abella and Bedwyr include the $\nabla$ quantifier [18].

In our current setup, we attempt to find counter-examples using Bedwyr to execute both the genera-
tion 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 $\mu$MALL [13],
in that setting, occurrences of negations can be eliminated by using De Morgan duality and inequality.

References

pp. 2:1–2:44.
soning 7(2).
Corecursive Proof Search with Horn Clauses. In: ESOP, Lecture Notes in Computer Science 11423,


