Structural Resolution For Automated Verification

By: Ekaterina Komendantskaya, Peng Fu and **Patricia Johann**

Abstract

We pose a research question: Can the newly-developed structural resolution be used to extend coinductive methods in automated theorem proving?

Ekaterina Komendantskaya, Peng Fu and Patricia **Johann**.(2015) "Structural Resolution for Automated Verification". *Automated Verification of Critical Systems* 2015 (AVoCS'15). ISSN 1863-2122
Version Available At www. www.AVoCS.com

1 Coinductive methods in ITP and ATP

Mathematical induction is pervasive in programming and program verification. It arises in data definitions (e.g., it describes some algebraic data structures), it underlies program semantics (e.g., it explains how to reason about finite iteration and recursion), and it gets used in proofs (e.g., it supports lemmas about data structures used in inductive proofs). Coinduction, too, is important in programming and program verification. It arises in infinite data definitions (e.g., lazily defined infinite streams), semantics (e.g., of concurrency), and proofs (e.g., of observational equivalence, or bisimulation, of potentially infinite processes). It is thus desirable to have good support for both induction and coinduction in systems for reasoning about programs.

The first implementations of coinduction were pioneered in interactive theorem proving (ITP) [Gim98], where the duality of inductive and coinductive methods was achieved by distinguishing inductive from coinductive types, recursive functions consuming inputs of inductive types from corecursive functions producing outputs of coinductive types, and methods for constructing inductive proofs from those for constructing coinductive proofs.

Recently, the rapid development of automated theorem proving (ATP) in general, and SAT/SMT solvers in particular, has opened the way to introducing induction and coinduction to ATP [LM14, RB15, S$^+$07], and thus to bridging the previously existing gap between coinductive methods in ITP and ATP. Some coinductive methods of ITP can be easily translated to ATPs. For example, definitions of (inductive and) coinductive types in ITP translate naturally into fixed-point definitions in ATP. However, some coinductive methods in ITP are much trickier to adapt to ATP. In particular, the notion of program and function *productivity* that is so central to the theory of corecursive functions in ITP has, until recently, been virtually absent from ATP.

2 Why ITP Theory of Productivity is a challenge for ATP?

In ITP, productivity plays a role for coinductive computations dual to that of termination for inductive ones. To safely use potentially non-terminating programs defined by corecursion, a

* This author is sponsored by the EPSRC Grant EP/K031864/1.

system must be able to guarantee that they are *productive*, in the sense that each part of their (potentially) infinite coinductive output will be generated in finite time. In the absence of productivity, the soundness of systems supporting corecursion is not ensured. Some systems, including Coq and Agda, guarantee productivity via syntactic guardedness checks [Gim98] which ensure that any recursive call to a corecursive function occurs under a call to a constructor of the program's output data type, and thus that the program's output yields finite observations by terminating recursive programs.

In ITP, productivity and guardedness depend crucially on types and type constructors, as well as on reductions by pattern matching computations. Indeed, the very definition of a productive function in ITP requires that the type of its output data be coinductive. But in the untyped setting of ATP there is no way to ensure that a function's output is coinductive, and thus that coinductive reasoning will be sound for it. In addition, ITP guardedness checks do not work if reductions by pattern matching are replaced by derivations via (SLD-)resolution, as commonly used in ATP.

Some approaches to coinduction in ATP, such as those of [RB15, S$^+$07], can construct coinductive proofs only for terms that can be represented as rational (or regular) trees. The corresponding regular corecursion is relatively easy to handle operationally via cycle detection [S$^+$07], and yields cyclic closed terms. This is crucial for SAT/SMT solvers [RB15]. Other approaches "guard" corecursion by imposing a recursive observational measure on corecursive functions — thus effectively viewing corecursion as a form of recursion [LM14]. All such additional restrictions are unnecessary in ITP, and, due to their *ad hoc* nature, work for only restricted cases of corecursion. In the general case, they do not actually capture the essence of ITP productivity.

3 Can Structural Resolution help?

Structural Resolution [JKK15, JKF$^+$15] is a newly-proposed resolution method that supports a very natural definition of productivity, as well as semi-decidable guardedness checks for it.

The propositional resolution rule underlying most modern ATPs is given by $\frac{C\vee A \quad \neg A\vee D}{C\vee D}$, where A,C,D are propositions. The standard (first-order) resolution rule is then (1) $\frac{C\vee A \quad \neg B\vee D}{\theta(C)\vee\theta(D)}$, where A,B,C,D are (first-order) terms and θ is a unifier of A and B (i.e., $\theta(A)=\theta(B)$), and the pattern matching reduction used in ITP is given by the restriction of the above rule to (2) $\frac{C\vee A \quad \neg B\vee D}{\theta(C)\vee D}$, where A,B,C,D are (first-order) terms and θ is a matcher of A and B (i.e., $\theta(A)=B$). The restricted rule (2) is of course incomplete relative to rule (1), and requires a further rule to emulate the effect of standard resolution by rule (1). This rule is given by (3) $\frac{C\vee A \quad \neg B\vee D}{C\vee A,\ \theta(\neg B)\vee\theta(D)}$, where A,B,C,D are (first-order) terms and θ is a unifier of A and B.

Subject to careful definitions, derivations comprising rules (2) and (3) can be shown to emulate the effect of standard resolution by rule (1) [FK15]. These are called derivations by *structural resolution*. Importantly, and perhaps surprisingly, structural resolution bears properties that are key to the theory of coinduction and productivity for resolution-based methods. Logic programs that correspond to productive corecursive functions in ITP are precisely those for which reductions steps by rule (2) always terminate (give finite observations), and reductions by rule (3) can be applied infinitely (thus accounting for the coinductive nature of a program's "output"). These two properties can be used to define productivity via structural resolution in LP. We ask: **Can structural resolution give a theory of productivity for other resolution-based ATPs?**

Bibliography

[FK15] P. Fu, E. Komendantskaya. Horn Formulas as Types for Structural Resolution. In *LOPSTR*. 2015.

[Gim98] E. Giménez. Structural Recursive Definitions in Type Theory. In *ICALP*. Pp. 397–408. 1998.

[JKF+15] P. Johann, E. Komendantskaya, F. Farka, P. Fu, M. Schmidt. A Structural Approach to Productivity and Guardedness in Logic Programming. In *submitted*. 2015.

[JKK15] P. Johann, E. Komendantskaya, V. Komendantskiy. Structural Resolution for Logic Programming. In *Technical Communications of ICLP*. 2015.

[LM14] K. R. Leino, M. Moskal. Co-induction Simply - Automatic Co-inductive Proofs in a Program Verifier. In *FM*. Pp. 382–398. 2014.

[RB15] A. Reynolds, J. C. Blanchette. A Decision Procedure for (Co)datatypes in SMT Solvers. In *CADE*. Pp. 197–213. 2015.

[S+07] L. Simon et al. Co-Logic Programming: Extending Logic Programming with Coinduction. In *ICLP*. Pp. 472–483. 2007.