Thinking Outside the (Arithmetic) Box: Certifying RAHD Computations

Florent Kirchner[1] and Grant Olney Passmore[2]

[1] INRIA
Centre de Recherche
INRIA Rennes – Bretagne Atlantique
Campus universitaire de Beaulieu,
35042 Rennes Cedex France
`florent.kirchner@irisa.fr`
[2] LFCS
School of Informatics
Informatics Forum
10 Crichton Street
Edinburgh EH8 9AB
Scotland, UK
`g.passmore@ed.ac.uk`

Abstract. RAHD [8] is a tool that takes advantage of a heterogeneous collection of proof procedures to decide the satisfiability of formulas in the existential fragment of the theory of real closed fields. But system analysts can rarely restrict themselves only to arithmetic-based problems: they might want to use RAHD as part of a more diverse formal toolbox. In this paper, we present ongoing work on an architecture that enables the skeptical integration of RAHD into general-purpose proof assistants (PA). We will present a system-level view of how this integration can be performed, and examine efficiency concerns for this implementation.

1 Introduction

Given $p_{i,j} \in \mathbb{R}[\boldsymbol{x}]$, and $\bowtie \in \{=, >, \geq\}$, RAHD decides problems of the form:

$$\langle \mathbb{R}, +, -, \times, >, 0, 1 \rangle \models \exists \boldsymbol{x} \bigwedge_{i=1}^{c} \bigvee_{j=1}^{l_c} p_{i,j}(\boldsymbol{x}) \bowtie 0$$

RAHD is tailored to provide efficient solutions on industrially-derived workloads. In particular, it has been designed with the goal of addressing large and largely linear (L3) non-linear problems better than competing tools [9].

RAHD's pragmatic focus makes it a notable candidate for inclusion in formal verification toolchains. Take static analysis of programs for instance [7]: the technique generates sizeable mathematical models, and requires checking properties of these models in a fast and scalable fashion. In some cases, this involves using decision procedures for non-linear arithmetic. In addition, if a high level of trust is required, then proof assistants (PA) such as Coq are used to secure the analysis. The conjunction of both factors has created an increasing need for arithmetic verification in PAs [2,11,3]. Coq includes several commands to deal with arithmetic proofs [5,1,12] which are complementary to RAHD's aforementioned strength and features[3]. Our long-term goal is to allow bidirectional communication between these tools, with RAHD being called from Coq to solve L3 problems intractable with conventional methods, and Coq being invoked from RAHD to discharge non-arithmetic subgoals.

This short paper presents work in progress on the integration of RAHD with the Coq PA, via certificate generation and checking. In particular, it presents our take on the following questions:

Efficiency How can we ensure that the interfacing is efficient, both from the PA point of view (fast certification of external proofs), and from RAHD's perspective (least impact of certificate generation)?

Architecture Can we build a system that is well-integrated with both tools, and can be modularly adapted, should the need arise, to other use cases?

2 RAHD Preliminaries

Before we proceed with our contribution, we give a brief outline of the RAHD system architecture and introduce terminology which will be used throughout the rest of this short paper. More information can be found in [8,9].

– RAHD accepts as input an implicitly $\exists$-closed boolean combination of rational function equations and inequalities called a *goal*. A *case* is a conjunction of polynomial equations and inequalities. A *goalset* is a collection of *cases* whose disjunction is equisatisfiable with the goal.
– RAHD proof methods are embodied primarily in *case manipulation functions* (CMFs) which perform SAT-preserving transformations upon cases. The output of a CMF upon a case may be a *subgoal* which is a goal equisatisfiable with the case.
– RAHD has a fully-automatic mode called the *waterfall*. It is a heuristic combination of CMFs which calls itself recursively upon derived subgoals.
– RAHD also provides interactive facilities. In this mode, CMFs are applied manually through the use of *tactics*. This mode has proof-tree exploration machinery similar to general-purpose proof assistants.

3 Effective Instrumentation in RAHD

The main concern when instrumenting automatic tools (decision procedures, SAT solvers, SMT solvers, etc.) is that logging cannot be allowed to hinder the

[3] It should also be noted that RAHD already has embryonic connections to other tools: it can be invoked from PVS using a blackbox tactic, and there are short-term plans to integrate it with MetiTarski [10], and the Yices SMT solver.

size and time characteristics of the space exploration process. Yet, to produce accurate and reasonably checkable certificates, it is essential for any complex tool to provide deeply-nested—and often performance-restraining—hooks into its core functions. Our solution to this conundrum follows a shrewd concept [13]: for each tool, have an optimized version and a verifying version run sequentially, with the former optionally guiding the latter.

This is the principle behind RAHD's *staggered proof refinement*:

0. while no certificate is needed, RAHD can be run without any performance hit in its unadulterated version;
1. when a trace of operations is required to produce a certificate, RAHD first produces a list of cases that have been modified by CMFs during the execution;
2. a second iteration of RAHD outputs deep, detailed logging of the CMFs, but *only* when they are applied to the modified cases as recorded in step 1., and *only* when logging is required to produce a certificate;
3. finally, these low-level logs are post-processed to produce non-redundant, precise traces.

Figure 1 describes this method. Progress monitoring, as it occurs in step 1., is an operation that has relatively low overhead. This is because it takes advantage of the CMF orchestrator, which in vanilla RAHD already computes (but does not record) the required information. At this point we are able to reconstitute the state of each case before and after the CMFs which contribute to the proof. Step 2 is expensive, so we limit it to occurrences that matter. Some certificates may only require information on cases, and no low-level logs, causing this step to be skipped too. Finally, post-processing does not take place in RAHD directly (see Section 4).

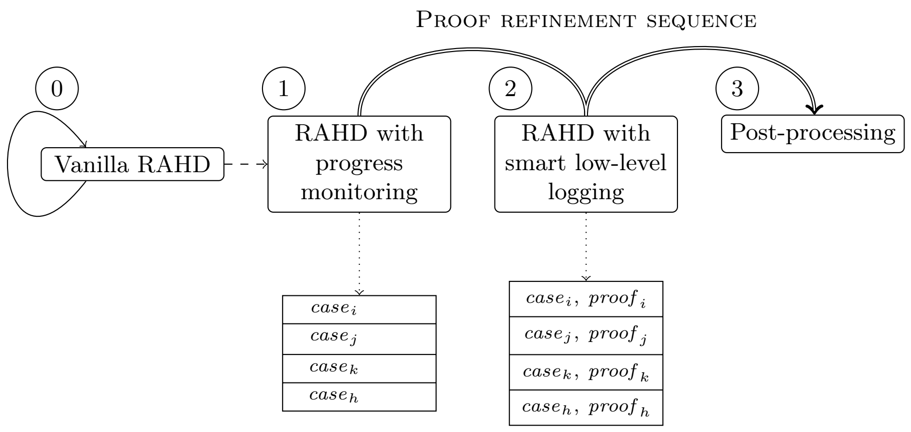

Fig. 1. Staggered Proof Refinement in RAHD.

We have introduced a low-level language to write the logs in step 2: since CMFs operate on cases in a deductive fashion, it is called the *proof command language*, or PCL. Proofs in this language are formed using three operations: `Init`, `Label`, and `Infer`. Let us illustrate on a simple example: take a CMF that proves unsat cases using Hilbert's Nullstellensatz. Informally, given a case as a conjunction of literals, some of which are equational constraints on polynomials, the function will use Gröbner Basis algorithms to saturate the related ideal, until it finds an element that satisfies the unsat criterion. The PCL log for such an process reads:

```
Init(poly1, P1)
Init(poly2, P2)
Init(poly3, P3)
Label(Ideal(P1,P2,P3),I)
Label(SPol(P1,P2),P4)
Label(SPol(P2,P3),P5)
...
Label(SPol(Pj,Pk),Pn)
Infer(UNSAT-BY-NULLSTELLENSATZ,I,Pn)
```

Here `poly1`, `poly2` and `poly3` are the polynomials in the equational fragment of the case. `I`, and `P1` to `Pn` are labels; `Ideal`, `SPol` and `UNSAT-BY-NULLSTELLENSATZ` are symbolic representations of CMF computations. PCL output is generated by every CMF that requires logging, from interval arithmetic to virtual term substitution. In this sense, it can be considered as a system-wide API for proof logging in RAHD.

Size-wise, the PCL representation is very compact: the labelling system allows implicit structure sharing, and computations are not stored. The latter might change in the future, as intermediary results could be added if additional certificate precision is deemed necessary. Additional schemes for enhancing structure sharing could then become necessary; preliminary investigations are being conducted in this regard.

4 The Extended Clause Database

There are a number of ways to interface proof assistants and decision procedures. Often, the PA more or less mimics the behavior of the procedure, so that it will systematically verify all of its operations. Instead, we opted for the so-called *skeptical approach*, where the PA delegates proof search to RAHD, then performs proof checking on the answer. This has a number of advantages: the checking routine is much faster than complete search space exploration, and the connection between the tools is based on a predefined format and thus is highly modular. In addition, the certificates can be used in proof-carrying code scenarios [6], or as part of the process to pass standardized evaluation requirements [4].

For the sake of modularity, the communication between the tools was given a storage-oriented twist. All the certificates transit via the *Extended Clause*

Database (ECDB), which provides a persistent store of the data involved in a RAHD proof. Table 1 illustrates the contents of the ECDB. The `case-id` and `goal-key` fields yield the structure of the proof tree. `case`, `cmf` and `status` respectively record the case (in s-expression form), the identifier of the CMF that modified it, and the eventual status of the case. Finally, `cert` contains the certificates generated by RAHD, and can also be used to include some information from the PA's checking algorithm. The database is directly fed by RAHD's staggered proof

case-id	goal-key	case	cmf	cert	status
0	0	$A \wedge B \wedge \cdots$	simp-real-null	psatz	:UNKNOWN
1	0	$C \wedge D \wedge \cdots$	rcr-svars	nil	(:UNSAT :BY-SUBGOAL)
0	0.1	$D' \wedge \cdots$	simp-tvs	auto	:UNSAT
2	0	$E \wedge F \wedge \cdots$	quick-sat	model	:SAT

Table 1. A sample table from the Extended Clause Database.

refinement sequence, and supports in-place certificate post-processing (step 3 of Section 3), on-the-fly translation of cases to PA syntax, and state-preserving shutdown and restart. Database queries can be made using trivial equational constraints: this affords the PA process additional flexibility, since proofs can be discharged regardless of case ordering. What is more, it enhances PA-side performance by allowing it to group non-consecutive certificate checks together, factoring potential overhead costs.

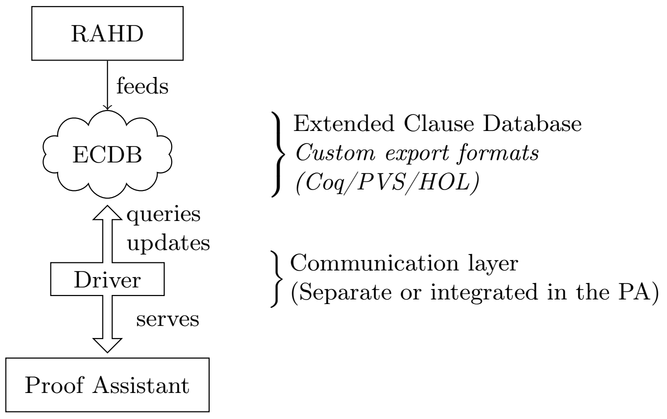

Fig. 2. System Architecture for the RAHD Interconnect.

Figure 2 represents a high-level view of the RAHD–Proof Assistant interconnect. When called upon a formula f, RAHD spawns an ECDB process, creates a table named after f, and populates it with extended clauses through direct

LISP function calls. The PA, via internal code or a proxy driver, can then use TCP sockets to dialogue with the ECDB. It requests certificates and can update the database with proofs of certification. Overall, this architecture provides multiple degrees of flexibility for a wide combination of use cases: synchronous or asynchronous operations, in or out-of-order processing of certificates, local or remote connections, on-the-fly translation, etc. We feel confident that it will be suitable not only for Coq, but also for PVS, Isabelle, HOL, and perhaps see other non-formal uses such as proof visualization.

5 Conclusion

RAHD is a quickly maturing tool that now includes fast algorithms to generate and refine verifiable certificates for its computations. The ECDB constitutes a flexible and persistent storage structure for these certificates, and an innovative take on the architecture of connections between proof assistants and external decision procedures. Much work remains to be done, in particular on the Coq side, to complete the integration with RAHD: now that most of the facilities are in place, work can concentrate on writing proof checking tactics. *In fine*, we look forward to providing Coq users with access to RAHD's wide selection and robust implementation of decision methods over the theory of real closed fields.

References

1. Frédéric Besson. Fast Reflexive Arithmetic Tactics the Linear Case and Beyond. In Thorsten Altenkirch and Conor McBride, editors, *TYPES*, volume 4502 of *LNCS*, pages 48–62, Heidelberg, 2006. Springer.
2. Frédéric Besson, Thomas Jensen, and David Pichardie. Proof-Carrying Code from Certified Abstract Interpretation to Fixpoint Compression. *Theoretical Computer Science*, 364(3):273–291, 2006.
3. Frédéric Besson, Thomas Jensen, David Pichardie, and Tiphaine Turpin. Certified Result Checking for Polyhedral Analysis of Bytecode Program. In *Proc. of 5th Intl. Symp. on Trustworthy Global Computing*, LNCS, Heidelberg, 2010. Springer. In press.
4. ISO/IEC Standard 15408. *Common Criteria for Information Technology Security Evaluation*, version 2.0 edition, November 1998.
5. Assia Mahboubi. Programming and certifying a CAD algorithm in the Coq system. In Thierry Coquand, Henri Lombardi, and Marie-Françoise Roy, editors, *Mathematics, Algorithms, Proofs*, volume 05021 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2005.
6. George Necula. Proof-carrying code. In Neil Jones, editor, *Proc. of the 24th Annual ACM Symp. on Principles of Programming Languages*, pages 106–119. ACM Press, 1997.
7. Flemming Nielson, Hanne Riis Nielson, and Christopher Hankin. *Principles of Program Analysis.* Springer, corrected 2nd printing edition, 1999.

8. Grant Passmore and Paul Jackson. Combined Decision Techniques for the Existential Theory of the Reals. In Jacques Carette, Lucas Dixon, Claudio Sacerdoti Coen, and Stephen Watt, editors, *Calculemus/MKM*, volume 5625 of *LNCS*, pages 122–137, Heidelberg, 2009. Springer.
9. Grant Passmore and Paul Jackson. Short paper supplement to presentation-only version of Calculemus'09 paper "Combined Decision Techniques for the Existential Theory of the Reals", 2010. Logics for System Analysis.
10. Laurence Paulson and Paul Jackson. Automatic Proof Procedures for Polynomials and Special Functions, 2010. EPSRC Research Proposal.
11. André Platzer, Jan-David Quesel, and Philipp Rümmer. Real World Verification. In Renate Schmidt, editor, *CADE*, volume 5663 of *LNCS*, pages 485–501, Heidelberg, 2009. Springer.
12. Loic Pottier. Connecting Gröbner Bases Programs with Coq to do Proofs in Algebra, Geometry and Arithmetics. In Piotr Rudnicki, Geoff Sutcliffe, Boris Konev, Renate Schmidt, and Stephan Schulz, editors, *LPAR Workshops*, volume 418 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.
13. Natarajan Shankar. On trust by heterogeneous replayability, 2008. Personal communication.