Proceedings of the
10th International Workshop on
Automated Verification of Critical Systems
(AVoCS 2010)

Integrating Automated and Interactive Theorem Proving in Type Theory

Karim Kanso and Anton Setzer

2 pages

Integrating Automated and Interactive Theorem Proving in Type Theory

Karim Kanso[1]* **and Anton Setzer**[2]†

[1] cskarim@swansea.ac.uk, [2] a.g.setzer@swansea.ac.uk, http://www.cs.swan.ac.uk/~csetzer
Department of Computer Science, Swansea University, UK

Abstract: We introduce an approach of integrating automated theorem proving techniques into the interactive theorem prover Agda. Our approach is generic and flexible, and can be combined with dependently typed programming. We have implemented the special cases of SAT solving and CTL model checking. The tool has been used for verifying the correctness of railway interlocking systems.

Keywords: type theory, automated theorem proving, interactive theorem proving, proof generation, Agda 2, SAT, CTL.

Agda is an interactive theorem proving (ITP) tool as well as a dependently typed functional programming language. It is based on Martin-Löf type theory. Proofs and programs are identified via the propositions as types paradigm, and therefore programmers can create proofs in the same way as writing programs. When teaching Agda even weak students were able to carry out proofs by induction by writing recursive termination checked programs. Another advantage of Agda is that it can be used for developing dependently typed programs. The second author has written programs involving graphics in Agda, and our long term goal is to write fully verified programs interacting with GUIs. This allows to write critical systems directly without translating them into a different language, which is especially useful for verified rapid prototyping.

At the moment there is no direct automated theorem proving (ATP) facility available in Agda. We show how to overcome this and how to use Agda in the context of real world railway interlocking systems. Our approach is different from the three main approaches for integrating ATP tools into ITP we have identified:

1. The use of an oracle, which is an operation which provides a proof of a theorem, and which when invoked calls an external tool. See for instance approaches by Tverdyshev, Müller and Nipkow [MN95] in Isabelle. The problem with this approach is that the result of the oracle does not reduce to head normal form, and therefore destroys the ability to execute programs. Another problem is that it is not clear whether the external tool is correct and whether the result of the external tool was interpreted correctly.

2. One verifies the correctness of an external tool and extracts from this proof a verified ATP tool. This program is called by a tactic. This approach has for instance been taken by Verma; and Lescuyer and Conchon [LC08]. This requires a correctness proof for an ATP tool in ITP, which might be difficult to achieve for state of the art theorem provers.

* Supported by Invensys Rail Systems, Chippenham, UK.
† Supported by EPSRC grant EP/G033374/1, theory and applications of induction-recursion.

3. One uses an ATP tool which provides in case of a positive answer a proof object, which can then be translated automatically into an ITP proof of the corresponding ITP formula. Since the ITP proof is now checked, the correctness relies entirely on the correctness of the ITP checker. Furthermore, a proof object is kept and therefore the ATP has to be executed only once. This approach has been used for instance by Paulson and Susanto; Böhme and Nipkow [BN10]; Dong, Ramakrishnan and Smolka; and Weber. Problems are that creating the proof object slows down the ATP tool, and that many ATP tools do not provide a proof object. Furthermore the proofs provided by the ATP tool might be very big (proof sizes of several hundreds of Megabytes have been reported for SMT solving [Stu09]). Therefore type checking translated ITP proofs might be infeasible.

We present here a fourth approach: We develop data types For of formulas and Model of models for the ATP tool. We define a function $\_ \models \_ : \text{Model} \to \text{For} \to \text{Set}$ which determines for a model $\mathscr{M}$ and formula φ the Agda formula $\mathscr{M} \models \varphi$ expressing that φ holds in $\mathscr{M}$. In case of SAT this relation refers to arbitrary instantiations of variables and in case of CTL to arbitrary infinite paths. Furthermore we write a decision procedure $\text{check} : \text{Model} \to \text{For} \to \text{Bool}$, which checks whether $\mathscr{M} \models \varphi$ holds. Then we prove that $(\text{check}\ \mathscr{M}\ \varphi)$ is true iff $\mathscr{M} \models \varphi$. We obtain a function $f : \text{check}\ \mathscr{M}\ \varphi == \text{true} \to \mathscr{M} \models \varphi$ and therefore, if $(\text{check}\ \mathscr{M}\ \varphi)$ reduces to true, then $(f\ \text{reflexivity}) : \mathscr{M} \models \varphi$ is the desired proof object. In order to make the correctness proof easy, check is introduced in a simple and therefore inefficient way. Then the call of check for closed arguments is replaced by a call to an external ATP tool. The above approach has been implemented for SAT solving and CTL model checking.

Note that $(\text{check}\ \mathscr{M}\ \varphi)$ returns true or false, so normalisation to head normal form is preserved. This approach can be used in combination with most ATP tools. A disadvantage is that the correctness and correct usage of the ATP tool is not verified. However, proofs in Agda are very explicit (no tactics are involved) and can be checked by hand. One can identify exactly the calls of the check function and needs then to rely on the correctness and correct usage of the ATP tools used. In ATP, even in the context of critical systems, it is quite common to use an ATP tool without having a formal correctness proof. Therefore, we believe that even though it is not ideal, it is acceptable to rely on an external tool without full verification.

Bibliography

[BN10] S. Böhme, T. Nipkow. Sledgehammer: Judgement Day. In Giesl and Hähnle (eds.), *Automated Reasoning*. Volume 6173, pp. 107–121. LNCS, 2010.

[LC08] S. Lescuyer, S. Conchon. A Reflexive Formalization of a SAT Solver in Coq. In *Proceedings of TPHOLs 2008*. 2008.

[MN95] O. Müller, T. Nipkow. Combining model checking and deduction for I/O-automata. *LNCS* 1019:1–16, 1995.

[Stu09] A. Stump. Proof Checking Technology for Satisfiability Modulo Theories. *Electronic Notes in Theoretical Computer Science* 228:121 – 133, 2009.