Integration between SMT Solvers and Lean

Tomaz Gomes Mascarenhas

1 Introduction

1.1 Context

One of the most desired properties of a software is its correctness. A system is said to be correct if its execution respects a given specification, regardless of the input it receives. For instance, one can specify that one important property for a sorting routine is that it should return the sorted permutation of the input list. Thus, a given function will be considered correct if it respects this rule for all possible input lists.

There are a variety of techniques to obtain correctness evidence for a software. The most common one is the development of tests. Besides being easy to write an efficient set of tests, there are many types of bugs that can be discovered with its execution. In fact, this approach is enough for a large amount of problems that are solved by software engineering. However, tests can't guarantee that a program doesn't have flaws, since the number of valid inputs is almost always exceedingly large, or infinite. This kind of guarantee is extremely important for systems that have critical responsibilities, such as the control of airplanes or medical equipment. This necessity motivates efforts to obtain a more general solution to the problem. In this context, a promising alternative is to carry out the formal verification of the software, that is, the development of a mathematical proof that it meets the predetermined specification. The present work will explore two of the main techniques to obtain this type of evidence: proof assistants and SMT solvers.

Proof assistants are tools that support programming languages by exploring deep relations between logic and computer science, in such a way that they can function as interactive theorem provers, which can refer to routines written in the language itself. In addition, the assistants are designed in a way that part of the type checking performed by the compiler is equivalent to verifying the validity of the provided proofs. Thus, it is possible to guarantee that these do not have errors. The ability of the assistants to automatically produce evidence for a given theorem is quite limited, therefore, their use cases are, in general, restricted to organizing the hypothesis and the proposition that must be proved in a given context, in addition to verifying the validity of the arguments given by the user. It is also important to highlight that these tools are widely used by mathematicians interested in certifying the correctness of their proofs. Indeed, recently the famous mathematician Peter Scholze helped the community of the Lean [7] proof assistant to formalize the proof of a very complex theorem that he was working on and had doubts about its correctness [6]. It turns out that Lean could point out some mistakes in his arguments, and, once they were fixed, the proof could be successfully certified by the assistant, bringing it a lot of attention, and convicing Scholze himself of the correctness of the theorem.

Satisfiability Modulo Theories (SMT) [3] is a generalization of the Boolean Satisfiability Problem (SAT), in which the underlying logic system is First Order Logic, instead of Propositional Logic. In addition to that, a set of theories is included, that is, a set of structures on which all the terms can be defined (as opposed to just the Booleans), and it is allowed to use operators of such structures. For instance, one could use as structure the real numbers and their comparison operators, making assertions about real variables. SMT solvers are systems specialized in solving this problem. Moreover, they do this automatically, with the only interaction with the user being the definition of the problem that they want to solve. It is possible to encode the problem of software verification as an instance of SMT, and that constitutes one of the main uses of SMT solvers.

On the one hand, proof assistants can rely on the creativity and previous experiences of the user to elaborate ingenious arguments. On the other hand, SMT solvers are highly optimized for performing extensive searches on the space of possible proofs, being able to find certain solutions that demand a long reasoning and the management of a large amount of information, which is really hard for human beings. This difference motivates the development of tools that could integrate the two techniques, making it possible to explore both of them simultaneously while writing a proof. Indeed, there are projects like Hammering Towards QED [4] that outline all the efforts that were already made in order to integrate proof assistants with automatic theorem provers. The document describes several tools, which are called hammers, that create this connection for a variety of proof assistants. It also describes the typical main components of them, providing an excellent guide to implement new hammers.

1.2 Goals

The goal of this project is to connect SMT solvers with Lean, which is currently emerging as a promising programming language and proof assistant. To achieve that, a tool will be implemented. With it, Lean users will be able to call SMT solvers to try to prove a theorem that is specified in the language of the assistant. For that, it will be necessary to translate each structure expressed on it into the SMT language, so that it will be possible to phrase the whole theorem in a way that the solver can work with it.

It is important to highlight that, at first, the reconstruction of the proof produced by the solver in the Lean language is outside the scope of this project. The tool will be limited to inform its user if the solver was able to solve that problem or not. One possible extension to this work could be the implementation of this reconstruction, as it was done in [5], for another assistant, called Isabelle [13].

2 Theoretical Framework

There are different types of APIs that provide an interface for using SMT solvers, in varied contexts. First of all, SMT-LIB [2] is an international initiative that, besides supporting research in this area in several ways, created a standard language for expressing instances of the SMT problem. This language is recognized by all SMT solvers, and can be used to communicate with all of them. Moreover, SMT-LIB provides a large amount of benchmarks that use the standard language to measure the efficiency of solvers. Indeed, facilitating the comparison between these systems is one of the main goals of this initiative.

At the same time, projects like PySMT [11] and SMT-Switch [12] seek to integrate the solvers with popular programming languages, namely Python and C++. The way they do this is by creating frameworks for these languages, in which it is possible to define variables and assertions in the SMT style and let the user choose a solver to try to prove the given theorem. The main advantage of this model is to facilitate the design of algorithms that are able to make decisions based on the output of the solver, besides defining dynamically what theorem will be sent to it. This kind of process is harder using just SMT-LIB, since it is necessary to write a file with all the details of the problem in order to use it.

Finally, there are APIs that connect SMT solvers with a specific proof assistant, as is the purpose of this work. For instance, there are interfaces developed for Agda[1], Isabelle[10] and Coq[1, 9]. This type of system directly translates the theorems expressed in the assistant to the standard language defined by SMT-LIB, or for the language of a tool such as PySMT. This is done completely by the mechanism that is creating the connection between the two systems, during the assistant's compilation time. This way, the user doesn't need to deal with this task. Once the SMT instance was produced, the API will feed the target solver with it and, if a proof is found, the API will send a message to the compiler of the assistant, allowing the compilation to succeed. Since there are a variety of theories over which the solvers operate, it is necessary to translate each one of them to the language of the assistant, using the native structures.

As was mentioned, a great advantage of using the solvers in this way is the possibility of relying both on their automatic reasoning and on human creativity to prove theorems. In spite of that, even being used by many people, Lean doesn't have the tooling to make this integration. This project intends to fill that gap, making the assistant even more effective.

3 Methodology

In order to implement the API between SMT solvers and Lean, several steps must be completed. First of all, it's necessary to understand how to use metaprogramming[8] in Lean, since this tool will be implemented in the language of the assistant itself. Then, each theory in SMT will be matched with a native structure of Lean. This matching will be used in the translation process. Also, it's necessary to define what are the possible ways to communicate the Lean language to the SMT language. It shall be decided whether SMT-LIB will be used, or an intermediate language through PySMT or SMT-Switch, or another method. Once that is done, the tools that currently do the same integration with other assistants will be studied. In particular, the way they translate the theories in SMT to their particular language. Finally, a prototype will be implemented and tested. For that, a set of theorems to be used as test cases will be defined. It's also important to decide what metrics of the tool must be measured, and use theorems that will highlight those aspects. All the obtained results will be reported in a scientific article. One last important point is that the team that currently leads the development of Lean will participate in the project and all the decisions that were mentioned will be taken together with it.

[1]`https://github.com/wenkokke/schmitty`

4 Schedule

The following table shows the plan of courses to be taken in each semester:

Tabela 1: Initial Schedule of Classes

Semester	**Classes**
2021/02	DCC874: Language Theory
	DCC865: Algorithm Design and Analysis
2022/01	DCC831: Formal Methods
	DCC880: Formal Semantics
2022/02	DCC831: Informational Theory
	DCC831: Theoretical Cryptography

The next table presents the planning of activities that will be performed per semester related to the project development:

Tabela 2: Initial Schedule of the Project

Semester	**Goals**
2021/02	Understand the operation of existing tools that integrate SMT solvers with proof assistants.
2022/01	Start the implementation of the tool.
2022/02	Finish the implementation of the tool and start the writing of the final report.
2023/01	Finish the writing of the final report.

References

[1] Michael Armand, Germain Faure, Benjamin Grégoire, Chantal Keller, Laurent Théry, and Benjamin Werner. A modular integration of sat/smt solvers to coq through proof witnesses. In Jean-Pierre Jouannaud and Zhong Shao, editors, *Certified Programs and Proofs*, pages 135–150, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[2] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The SMT-LIB Standard: Version 2.6. Technical report, Department of Computer Science, The University of Iowa, 2017. Available at `www.SMT-LIB.org`.

[3] Clark Barrett and Cesare Tinelli. *Satisfiability Modulo Theories*, pages 305–343. Springer International Publishing, Cham, 2018.

[4] Jasmin C. Blanchette, Cezary Kaliszyk, Lawrence C. Paulson, and Josef Urban. Hammering towards qed. *Journal of Formalized Reasoning*, 9(1):101–148, Jan. 2016.

[5] Jasmin Christian Blanchette, Sascha Böhme, and Lawrence C. Paulson. Extending sledgehammer with smt solvers. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *Automated Deduction – CADE-23*, pages 116–130, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[6] Davide Castelvecchi. Mathematicians welcome computer-assisted proof in 'grand unification' theory. *Nature*, 595(7865):18–19, June 2021. Bandiera_abtest: a Cg_type: News Number: 7865 Publisher: Nature Publishing Group Subject_term: Mathematics and computing.

[7] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris Van Doorn, and Jakob von Raumer. The lean theorem prover (system description). *International Conference on Automated Deduction*, 2015.

[8] Gabriel Ebner, Sebastian Ullrich, Jared Roesch, Jeremy Avigad, and Leonardo de Moura. A metaprogramming framework for formal verification. *Proc. ACM Program. Lang.*, 1(ICFP):34:1–34:29, 2017.

[9] Burak Ekici, Alain Mebsout, Cesare Tinelli, Chantal Keller, Guy Katz, Andrew Reynolds, and Clark Barrett. Smtcoq: A plug-in for integrating smt solvers into coq. In Rupak Majumdar and Viktor Kunčak, editors, *Computer Aided Verification*, pages 126–133, Cham, 2017. Springer International Publishing.

[10] Pascal Fontaine, Jean-Yves Marion, Stephan Merz, Leonor Prensa Nieto, and Alwen Tiu. Expressiveness + automation + soundness: Towards combining smt solvers and interactive proof assistants. In Holger Hermanns and Jens Palsberg, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 167–181, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[11] Marco Gario, Andrea Micheli, and Fondazione Bruno Kessler. Pysmt: a solver-agnostic library for fast prototyping of smt-based algorithms, 2015.

[12] Makai Mann, Amalee Wilson, Yoni Zohar, Lindsey Stuntz, Ahmed Irfan, Kristopher Brown, Caleb Donovick, Allison Guman, Cesare Tinelli, and Clark Barrett. Smt-switch: A solver-agnostic c++ api for smt solving. In Chu-Min Li and Felip Manyà, editors, *Theory and Applications of Satisfiability Testing – SAT 2021*, pages 377–386, Cham, 2021. Springer International Publishing.

[13] Tobias Nipkow, Markus Wenzel, and Lawrence C. Paulson. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic.* Springer-Verlag, Berlin, Heidelberg, 2002.