Proof Reconstruction: Parsing Proofs

Alejandro Gómez-Londoño

Universidad EAFIT
agomezl@eafit.edu.co

Abstract. Automated theorem provers (ATP) and proof assistants are among the developed sub-areas on automated reasoning, despite their approaches being certainly opposite, many new developments combine both techniques allowing a sub-proof to be automated using an ATP from within a proof assistant. Acting as a bridge between proof assistants and ATP, these systems known as *hammers* tend to enhance the functionality of an existing proof assistant, adding ATP capabilities into the interactive logical reasoning process. Agda a well known dependently typed functional programming language that can also be use as a proof assistant lacks in some degree of a hammer-like tool, and hence our goal is to fill part of this gap with a tool that can translate from an ATP generated proof into idiomatic Agda code, and doing so provide a base for further development.

1 Introduction

Automated theorem provers (ATP) and proof assistants has been around for decades (Davis 2001; Geuvers 2009). Despite the obvious differences between the two, both approaches share a fundamental goal which is to aid humans with complex proofs in an automatic or interactive manner, is maybe due to this relationship that in recent years various tools have been developed using a mixture of both systems. Such tools, some times referred to as *hammers* (Blanchette, Kaliszyk, and Paulson 2014) allow the users to write proofs in an interactive manner from within a proof assistant, but with the option of sending sub-proofs to an ATP.

Hammers by themselves aren't ATP nor poof assistants, they act more like a plugin that sits on top of the proof assistant and allows the communication with various ATP for proof automation, hammer-like tools typically consists on three mayor components (Blanchette, Kaliszyk, and Paulson 2014):

1. *Premise selector*: it gathers relevant theorems from the available libraries that can help with the current proof.
2. *Translation module*: takes the premises and the goal and translates them into the ATP input syntax (a common syntax to represent ATP problems is TPTP(Sutcliffe 2009)) this translation in most cases involves mapping a subset of the proof assistant logic into the ATP logic.
3. *Proof reconstruction module*: processes the proof returned by the ATP reconstructing it in the proof assistants syntax/logic.

As shown in Figure 1, hammers act as a bridge between the interactive and the automated process, constituting a more robust system capable of a more smooth interaction with logical reasoning process.

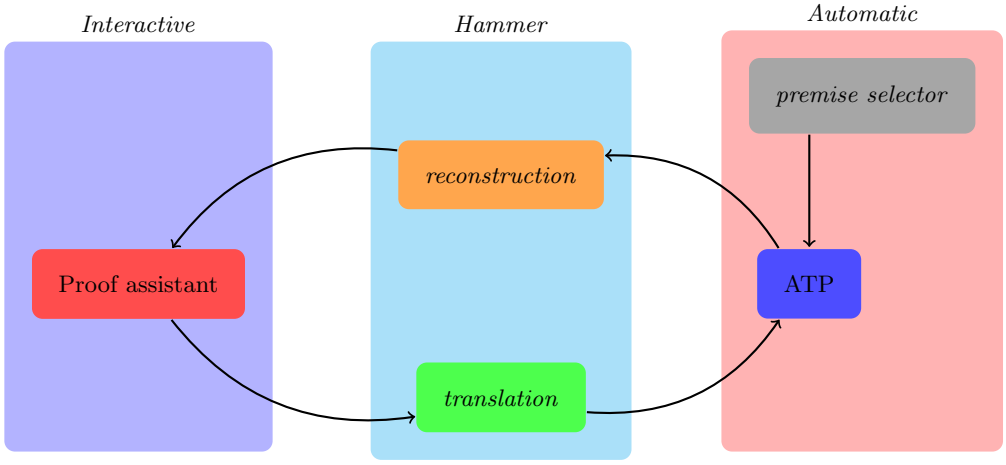

Fig. 1. Architecture of the various components involved in a hammer-like tool

2 Previous Work

As stated before, the development of ATPs and proof assistants dates from decades ago, but in comparison, the mixture of this two approaches (as *hammers*) is relatively new, nonetheless some exponents of this trend have been developed in the later years. Perhaps the best precedent in this category is Sledgehammer (Blanchette and Paulson 2014) a tool that sits on top of the Isabelle/HOL proof assistant (Nipkow, Wenzel, and Paulson 2002) and allows the translation/reconstruction of proofs to/from multiple ATPs. Another similar example for the Agda proof assistant is a work by Foster and Struth that proposes the integration of Agda with Waldmeister (Foster and Struth 2011) a theorem prover for equational logic (Buch, Hillenbrand, and Fettig 1996). The usefulness and convenience of the hammers can truly improve the way proof assistants work, taking most of the boilerplate and tediousness of proofs out of the way, an thus allowing to get more work done faster.

Currently Agda unlike Isabelle lacks of a true *hammer*, but this is an issue that is being addressed by the aforementioned work by Foster and Struth, and by some developments like the Apia tool (Sicard-Ramírez, Bove, and Dybjer 2014), which allows to prove first-order theorems from within Agda translating the formulae to TPTP and then sending it to multiple ATPs.

3 Problem Description

Agda (Norell 2007)(Agda Team 2015) as a proof assistant lacks of a hammer-like tool, but programs like Apia which allows to prove first-order theorems from within Agda using ATPs are closing this gap. Unfortunately Apia only works as a *translation module* and as a front-end for the ATPs. Some further development has to be done to achieve an Agda-hammer tool and one of the missing pieces in this enterprise is a *proof reconstruction module*, this would allow proofs to be verified from within Agda and jointly with a tool like Apia it could provide a fully functional hammer for Agda. Our goal is then to build a *reconstruction module* for Agda in order to fill this gap.

4 Procedure

The construction of a proof reconstruction module in his essence is just the translation of the output from an ATP-generated proof into the native language of a proof assistant, thus, the selection of both systems constitute a crucial design decision. In our case a proof assistant has already been chosen, but a number of criterias were taken into account when deciding what ATP to choose:

- Good performance, in terms of time needed to perform a proof.
- TPTP input, since is the most broadly used format.
- TSTP output, as a relatively new and promising standard for representing proofs (Sutcliffe, Zimmer, and Schulz 2004).
- Concise proofs.

Thus the most convenient input and output format are (as shown in Figure 2) TPTP/TSTP since they stand as the more common format among many ATP, and the amount of useful tools and documentation is significant

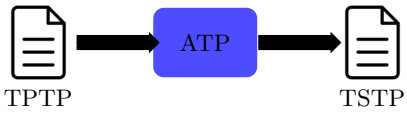

Fig. 2. ATP Input and Output formats

In terms of performance, a quick response time is an imperative requirement for any interactive system, and since the intended use for the chosen ATP will involve some level of interaction with the user, this criteria is of considerable importance. The last criteria refers to how concise a proof is in terms of how many rules or tactics are required to completed it, this helps for both the understanding of the proof, as for a more compact and simple implementation.

The reconstruction of a proof involve a series of steps (Figure 3) each of which perform some transformation over previous steps of the process, starting

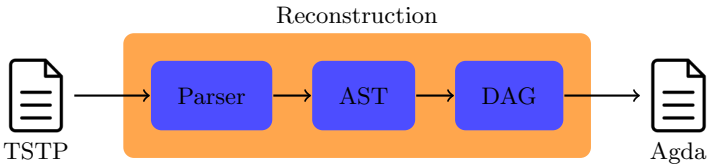

Fig. 3. Internals of a reconstruction module

with the TSTP input, and resulting in the final Agda code as output. The following sections describe the various steps planned for our implementation of a *reconstruction module*.

4.1 Parser and AST construction

In this two closely related steps the main objective is to analyze the TSTP input, in order to translate the proof into some suitable data structure allowing further manipulation from within the programing language used. This data structure is often called Abstract Syntax Tree (AST) and his main purpose is to represent only the relevant information from the input format in a concise and structured manner.

```
formula(s_0,plain,(x & y), conjunction(a_0,a_1)).

formula(s_1,plain,(z),modus_ponens(a_2,s_0)).

formula(r_0,plain,($true),simplify(s_1,c_0)).
```

Listing 1.1. TSTP-like proof fragment

```
F {name     = "s_0",
   role     = Plain,
   formula  = "x" (:&:) "y",
   annotations = Conjunction ["a_0","a_1"]
  }
```

Listing 1.2. Haskell data type representing a single TSTP formula

The conversion from TSTP format into a more concise data type is shown in Listings 1.1 and 1.2. After this translation, further analysis can be performed from within the programing language without using the original input file, since all the information is now represented in the corresponding data type.

4.2 DAG

A *directed acyclic graph* (DAG) is intended to be used to represent the relationship between each step of the proof. The process for constructing a DAG takes a list of formulas from the AST, and uses some of the information on each proof to construct a graph that effectively resembles the logical steps in the original proof.

```
fof(a_0,axiom,x).
fof(a_1,axiom,y).
fof(a_2,axiom, ((x & y) => z)).
fof(c_0,conjecture, z).
```

Listing 1.3. TPTP problem

```
fof(s_0,plain,(x & y),
    inference(conj,[],[a_0,a_1])).

fof(s_1,plain,(z),
    inference(modp,[],[a_2,s_0])).

fof(r_0,plain,($true),
    inference(simplify,[],[s_1])).
```

Listing 1.4. TSTP proof

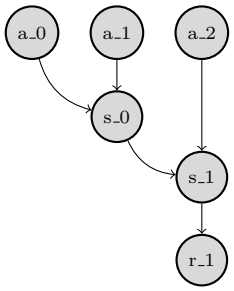

Fig. 4. DAG

$$\dfrac{\dfrac{x \qquad y}{x \wedge y} \qquad x \wedge y \Rightarrow z}{z}$$

Fig. 5. Actual proof

Listings 1.3 and 1.4 and Figures 3 and 4 present a complete example of how a DAG structure is capable of resemble the actual proof.

4.3 Agda code generation

Once the DAG is completed the next step is to translate the first order proof that it represents into a representation of FOL in agda. This step is perhaps the most complex of all and has not been fully designed yet.

5 Results

The current state of the project involves a number of design decisions as well as some development of the aforementioned steps of the process, all of which can be summarized as:

- Haskell and Agda has been chosen as the programing languages for the implementation. In Haskell, we will handle the parsing and AST construction, while in Agda, we will create and analyze the DAG.
- Metis (Hurd 2003) was chosen as our ATP, due to his simple kernel, good performance, and TPTP/TSTP support.
- A modified version of the `logic-tptp`[1] Haskell library has been used to implement a TSTP parser capable of analyze Metis proofs. This project is freely available on github[2].

References

Agda Team (2015). *The Agda wiki*. URL: `http://wiki.portal.chalmers.se/agda/pmwiki.php`.

Blanchette, Jasmin C., Cezary Kaliszyk, and Lawrence C. Paulson (2014). "Hammering towards QED". English. In: *Draft version*.

Blanchette, Jasmin Christian and Lawrence C. Paulson (2014). *Hammering Away. A User's Guide to Sledgehammer for Isabelle/HOL*. Institut für Informatik, Technische Universität München.

Buch, Arnim, Thomas Hillenbrand, and Roland Fettig (1996). "WALDMEISTER: High Performance Equational Theorem Proving". In: *Proceedings of the International Symposium on Design and Implementation of Symbolic Computation Systems*. DISCO '96. London, UK, UK: Springer-Verlag, pp. 63–64.

Davis, Martin (2001). "Chapter 1 - The Early History of Automated Deduction: Dedicated to the memory of Hao Wang". In: *Handbook of Automated Reasoning*. North-Holland, pp. 3 –15.

Foster, Simon and Georg Struth (2011). "Integrating an Automated Theorem Prover into Agda". English. In: *NASA Formal Methods*. Ed. by Mihaela Bobaru et al. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 116–130.

Geuvers, H (2009). "Proof assistants: History, ideas and future". English. In: *Sadhana* 34.1, pp. 3–25.

Hurd, Joe (2003). "First-Order Proof Tactics in Higher-Order Logic Theorem Provers". In: pp. 56–68.

Meredith, C. A. and A. N. Prior (1968). "Equational logic." In: *Notre Dame J. Formal Logic* 9.3, pp. 212–226.

[1] `https://hackage.haskell.org/package/logic-TPTP`

[2] `https://github.com/agomezl/tstp2agda`

Nipkow, Tobias, Markus Wenzel, and Lawrence C Paulson (2002). *Isabelle/HOL: A Proof Assistant for Higher-order Logic*. Berlin, Heidelberg: Springer-Verlag. ISBN: 3-540-43376-7.

Norell, Ulf (2007). "Towards a practical programming language based on dependent type theory". PhD thesis. Chalmers University of Technology and Göteborg University.

Sicard-Ramírez, Andrés, Ana Bove, and Peter Dybjer (2014). "Reasoning about Functional Programs by Combining Interactive and Automatic Proofs". Unpublished doctoral dissertion. PhD thesis. Uruguay: University of the Republic.

Sutcliffe, G (2009). "The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0". In: *Journal of Automated Reasoning* 43.4, pp. 337–362.

Sutcliffe, G., J. Zimmer, and S Schulz (2004). "TSTP Data-Exchange Formats for Automated Theorem Proving Tools". In: *Distributed Constraint Problem Solving and Reasoning in Multi-Agent Systems*. Ed. by Sorge V Zhang W. Vol. 112. Frontiers in Artificial In Itelligence and Applications. IOS Press.