Integrating an Automated Theorem Prover into Agda

Simon Foster and Georg Struth

Department of Computer Science, University of Sheffield, UK
{s.foster,g.struth}@dcs.shef.ac.uk

Abstract. Agda is a dependently typed functional programming language and a proof assistant in which developing programs and proving their correctness is one activity. We show how this process can be enhanced by integrating external automated theorem provers, provide a prototypical integration of the equational theorem prover Waldmeister, and give examples of how this proof automation works in practice.

1 Introduction

The ideal that programs and their correctness proofs should be developed hand-in-hand has influenced decades of research on formal methods. Specification languages and formalisms such as Hoare logics, dynamics logics and temporal logics have been developed for analysing programs, protocols, and other computing systems. They have been integrated into tools such as theorem provers, SMT/SAT solvers and model checkers and successfully applied in the industry. Most of these formalisms do not analyse programs directly on the code, but use external tools and techniques with their own notations and semantics. This usually leaves a formalisation gap and the question remains whether the underlying program semantics has been faithfully captured.

But there are, in fact, programming languages in which the development of a program and its correctness proof can truly be carried out as one and the same activity within the language itself. An example are functional programming languages such as Agda [7] or Epigram [15], which are based on dependent constructive type theory. Here, programs are obtained directly from type-level specifications and proofs via the Curry-Howard isomorphism. These languages are therefore, in ingenious ways, programming languages and interactive theorem provers. Program development can be based on the standard methods for functional languages, but the need of formal proof adds an additional layer of complexity. It requires substantial mathematical skill and user interaction even for trivial tasks. Increasing proof automation is therefore of crucial importance.

Interactive theorem provers such as Isabelle [17] are showing a way forward. Isabelle is currently being transformed into a versatile proof environment by integrating external automated theorem proving (ATP) systems, SMT solvers, decision procedures and counterexample generators [5,6,4]. Proof tasks can be delegated to these tools, and the proofs they provide are internally reconstructed.
to increase trustworthiness. But all this proof technology is based on classical logic. This has two main consequences. First, on the programming side the proofs-as-programs approach is not available in Isabelle, hence programs cannot be extracted from Isabelle proofs. Second, because of the absence of the law of excluded middle in constructive logic, proofs from ATP systems and SMT solvers are not generally valid in dependently typed languages. An additional complication is that proof reconstruction in dependently typed languages must be part of type-checking. This makes an integration certainly not straightforward, but at least not impossible.

Inspired by Isabelle we provide the first ATP integration into Agda. To keep it simple we restrict ourselves to pure equational reasoning, where the rule of excluded middle plays no role and the distinction between classical and constructive proofs vanishes. We integrate Waldmeister [10], the fastest equational ATP system in the world. Waldmeister also provides detailed proofs and supports simple sorts/types. Our main contributions are as follows.

- We implement the basic data-types for representing equational reasoning within Agda. Since Agda needs to manipulate these objects during the type checking process, a reflection layer is needed for the implementation.
- Since Agda provides no means for executing external programs before compile time, the reflection-layer theory data-types are complemented by a Haskell module which interfaces with Waldmeister.
- We implement equational logic at Agda’s reflection layer together with functions that parse Waldmeister proofs into reflection layer proof terms. We verify this logic within Agda and link it with the level of Agda proofs. This allows us to reconstruct Waldmeister proofs step-by-step within Agda.
- Mapping Agda types into Waldmeister’s simple sort system requires abstraction. Invalid proofs are nevertheless caught during proof reconstruction.
- We provide a series of small examples from algebra and functional programming that show the integration at work.

While part of the integration is specific to Waldmeister, most of the concepts implemented are generic enough to serve as templates for integrating other, more expressive ATP systems. Our integration can also be used as a prototype for further optimisation, for instance, by providing more efficient data structures for terms, equations and proofs, and by improving the running time of proof reconstruction. Such issues are further discussed in the final section of this paper.

Formal program development can certainly be split into creative and routine tasks. Our integration aims at empowering programmers to perform proofs at the level of detail they desire, thus making program development cleaner, faster and less error-prone.

This paper aims to explain the main ideas and features of our approach to a formal methods audience. Its more idiosyncratic aspects, which are mainly of interest for Agda developers, are contained in a technical report [8]; the complete code for our implementation can be found at our website [2].

1 http://www.cs.miami.edu/~tptp/CASC/ 15/02/2011
2 http://simon-foster.staff.shef.ac.uk/agdaatp