Using Dependent Types at Scale: Maintaining the Agda Standard Library

Matthew L. Daggitt
Heriot-Watt University
Edinburgh, Scotland
m.daggitt@hw.ac.uk

Guillaume Allais
University of St Andrews
Fife, Scotland
guillaume.allais@ens-lyon.org

Abstract

When creating a new language with real-world impact, implementing an advanced type-system is only the beginning. In order to attract new users, one must also have a comprehensive standard library that makes use of those advanced features. In this talk I will discuss the practicalities of using dependent types and other advanced language features in the Agda Standard Library.

The talk will be split into three parts. In the first I will cover how the upcoming version 2.0 of the library uses Agda's type-system to improve ergonomics and get a little closer to what a mathematician might write:

1. Using parametrised modules to significantly cut down the amount of boilerplate required for users.
2. Using instance search to fill in mechanistic proofs for dependently typed predicates.
3. Exploiting type-level computation to significantly reduce the number of cases that need to be considered when pattern matching.

The second part of the talk will focus on a rarely discussed topic in dependent type-systems, namely how the size of a dependently-typed library influences design decisions that, at first glance, appear unrelated. These effects include limiting the use of the dependently typed features.

The Standard Library has been growing at approximately 10kloc per year, and now stands at over 100kloc, putting it firmly into the realm of serious software projects by academic standards. Despite this, it contains surprisingly little computational content. For example we don't have real numbers or verified interfaces for maps, queues, stacks, sets etc. So what is taking up all the space? The answer is proofs, which outnumbers computation by a factor of about 10:1. The size of the library means that to type-check it all requires approximately 8GB of RAM. Consequently if a user wants to type-check their own code and have a web-browser open at the same time, they can't afford to type-check even half of the library. An apparently inexorable chain of consequences follows:

- We are now in a dependency hell where we not only have to worry about dependency cycles, we also have to minimise the *quantity* of code each module transitively imports. Correspondingly we are forced to favour small modules with minimal relationships between them.
- As proofs make up the vast majority of the library, it is immediate clear that the definition of and computation over data types must live in separate modules from the proofs about them.
- Consequently we lose the ability to hide implementation details behind abstraction barriers, as the 'private' and 'abstract' keywords in Agda currently only work on an intra-module level. Guaranteeing backwards compatibility is therefore near impossible and instead we're forced to rely on conventions, which new users are unlikely to be aware of.
- We also lose the ability to enforce dependently-typed pre-conditions to computations. For instance the function "deduplicate" can't take a proof that the underlying relation is an equivalence, because using it would then require importing proofs. This also significantly complicates the already labyrinthine hierarchy of mathematical structures, as we must have separate definitions, one with just the operations and one with both the operations and the laws.

In the final part of the talk I will cover some of the language features that would make the biggest difference to the library:

1. Improving how instance search handles irrelevancy.
2. Finer grained control over abstraction.
3. Better support for composing records when modelling inheritance diamonds.

WITS'22, 22 January 2022,
© 2021 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Matthew L. Daggitt and Guillaume Allais. 2021. Using Dependent Types at Scale: Maintaining the Agda Standard Library. In *Proceedings of Workshop on the Implementation of Type Systems (WITS'22).* ACM, New York, NY, USA, 1 page. https://doi.org/10.1145/nnnnnnn.nnnnnnn