Create and Explore a graph from Agda Definitions to analyse projects and speed up compilation

Dissertation

Julian Camilo Vidal Polania

School of Computer Science

College of Engineering and Physical Sciences

University of Birmingham

2024-25

Abstract

Usually 100-300 words stating the salient points of the report. It should help your reader to decide whether the report is relevant to her or his interests.

I will mention:

- How I got the s-expressions and how they were parsed
- The uses of the graph and the queries you can make
- How much it can speed up compilation and how

Acknowledgements

Abbreviations

ACB	Apple Banana Carrot

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Background

When you first encounter a massive code base, containing years of development from different people all adding their own code with their own style and thought process, it can be overwhelming and difficult to parse through. On most code-bases you can ignore the fundamental functions that underpin a given library, but in Agda it is quite useful to have knowledge of the underlying definitions of the project.

Agda is a functional programming language that can be used as a proof assistant, following the propositions as types logic system [1]. Since proofs in Agda can be created almost entirely from scratch, defining the most basic and fundamental types and operations it means that projects will quickly balloon in size. While most of the time it is possible to use higher level definitions when developing a proof, knowing the how the used types are fundamentally defined can be quite beneficial.

1.2 Motivation

Due to the size of these projects and how complex type definitions can be it becomes difficult to fully grasp what the relationships between all the definitions are. This creates the need for a tool that can analyze Agda projects and give the user an easy to use interface to query the relationships between definitions, in order to better understand the proofs. A tool that would allow the user to explore these relationships would be of great benefit.

Also, large Agda projects tend to deal with long compilation times. Agda is a proof assistant, during "compilation", it type checks the entire project which can take a 5 minutes or more depending on the computer and project. Agda will keep track of what modules have already been compiled, so working on high level modules isn't an issue as Agda will only type check files that have changed in

the project and high level modules don't have dependents that would need to be re-checked. In contrast, low level files with many dependents, such as a module defining the natural numbers, will cause a significant portion of the project to re-compile. This effect will be particularly pronounced during refactoring where significant changes will be made to the structure of the project, causing Agda to re-check the whole project making most of the time spent refactoring, spent waiting.

Agda type checks the project sequentially, going through the dependencies of each modules, this is done to mantain safety as a module won't be compiled until its dependencies have. A tool that analyzes the relationship between modules and finds modules that are safe to compile in parallel could lead to significant time savings.

1.3 Problem Statement

This project aims to create a CLI tool that will analyze an Agda project and create a dependency graph from its definitions. Agda doesn't provide a built-in tool to extract the definitions from a project, but there is an s-expression extractor Agda version by Andrej Bauer [2] that will allows a better view into Agdas internal representation. With this graph the CLI will have commands that allow the user to explore and query this graph, in order to gain insight into the Agda project. As there aren't currently any tools that allows you to query the definitions of an Agda project.

The CLI tool will also get the module dependency graph and generate Make-files and index files that will optimize the order of the modules compiled to minimize compilation time.

1.4 Overview

CHAPTER 2

Research

This is often referred to the 'literature review' section. It is one of the most important section of the Project Proposal and the Project Report. It is where you demonstrate that you understand the state-of-the-art in the field you're working. Towards the end of this sections it's normally a good idea to explain how your aim / work / idea / contribution differs from the nearest work in the field.

Research papers to explore:

- s-expressions
- How graphs can be used to represent the definitions
- Most tools focus on data-flow for static analysis, I am looking for a simpler tools that allows the querying of the relationships between definitions.

Tools that do a similar job

- CTags
  - CTags indexes all the symbols of a project
  - This works only on definitions of functions, not where they are referenced
- Jedi - an awesome autocompletion, static analysis and refactoring library for Python
  - It finds function definitions and references which is what we are looking for
- Agda-Language-Server
  - An LSP for agda, it parses the files and analysis them, similar to what we want but it doesn't seem to index them

- Application-only Call Graph Construction
  - A call graph represents the relationships between functions
- Precise Interprocedural Dataflow Analysis via Graph Reachabilit
  - Data flow is a way to statically analyse code before running it, this is far more complex than what I want.
- CFL/Dyck Reachability: An Algorithmic Perspective
  - Analyses the decidability and complexity of this problem

CHAPTER 3

Legal, Social, Ethical and Professional Issues

Address the legal, social, ethical, and professional issues associated with your project. Find something to say for each. For example, listening for legal, BCS Code of Conduct for professional, etc.

- Legal consider licensing of third-party libraries and how they can be used. Data protection isn't an issue as it all runs locally.
- Social: the tool provides documentation so it can be used by a variety of developers. Tool runs in any terminal so it can be customized to fit any accessible need from text.
- Ethical: the source code for the tool is fully open for transparency and collaboration.
- Professional: Adherence to the BCS Code of Conduct.

CHAPTER 4

System Requirements

This section should detail your understanding of what you are planning to create. The section should aim to break down the overarching aims of the work into clear, measurable requirements that can be used in the evaluation of the project. This is why we often function of functional and non-functional requirements.

- Explain what the project should do and how it should do it
- Functional requirements
  - Create graph from agda project
  - Should allow for asked queries (the ones mentioned in the Mathodon thread)
  - Reduce compilation time by 10%?
- Non-Functional requirements
  - Installation shouldn't be difficult
  - Queries should be done in under 2 seconds
  - Creation of the graph shouldn't take more than 10 minutes

CHAPTER 5

Design

Communicating how you think about the composition of your system and how it works. You might detail the ways in which the overall system will be broken down into subsystems. Detail should then be provided on the design of each of these subsystems.

- A system that turns agda into s-expressions
- A system that reads s-expressions and turns it into a graph
- A system that queries that graph
- A way to store the graph for future use

CHAPTER 6

Implementation

This section should discuss how you went about developing a system that was consistent with your design to meet your stated requirements. The implementation of subsystems should be accurately documented, with any implementation difficulties being acknowledged. The Design and Implementation sections can be grouped in the Project Report, if these are tightly coupled. Likely omitted for the Project Proposal, though should mention your proposed implementation technologies somewhere.

- Explain the s-expressions extractor
- How they are loaded into python
- How they are stored for future use
- How the compilation uses graph dot

CHAPTER 7

Testing and Success Measurement

As well as documenting system testing, this section should also describe any unit testing or integration testing performed. If you are not familiar with unit, integration or system testing then it would be a good idea to investigate these notions and consider about how they relate to your project. This section might also detail any performance, reliability or usability testing performed, with quantification, i.e., numeric measurements, being used wherever possible. All those points are systems focused through. If you're doing something that's research focused or more of a social / analytical study then you need to think about how you'll measure success.

- Measure compilation time
- Measure time to build graph
- Measure time to make queries

CHAPTER 8

Project Management

What is the timeline for your project? What software development methodology will you use? Can you identify the major milestones? These types of questions are important for project planning and should be addressed directly.

- Gantt chart
- Weekly updates with supervisor
- Getting the s-expression extractor was a major milestone
- Using python instead of clojure was a significant improvement

CHAPTER 9

Evaluation

Usually you evaluate your project with regard to the functional and non-functional requirements you set out in the earlier chapter. This doesn't necessary mean that your project was successful but, if these requirements were appropriately specified, you it's likely that your project was successful. You might be reiterating some points from the Testing and Success Measurement chapter in your Project Report.

- Mention compilation time reduction, and improvements
- Mention how some queries take too long but were all implemented
- Mention how trees take some time but can be re-used instantaneously
- Creates graph from any agda project

CHAPTER 10

Conclusion

Generally speaking, section can take different forms - as a minimum you would normally provide a brief summary of your project work and a discussion of possible future work. You may also wish to reiterate the main outcomes of your project and give some idea of how you think the ideas dealt with by your project relate to real-world situations, etc.. For the Proposal, don't mention future work but do summarise your document.

- Compilation time could be further reduced
- A clever way of removing cycles could lead to faster queries
- Many useless nodes that are part of Agda's backend but not necessary for the user.
- Work on making the whole process more automatic and easily distributable

Bibliography

[1] Agda Developers. Agda documentation. https://agda.readthedocs.io/en/latest/overview.html.

[2] Andrej Bauer. Agda S-expression Extractor. https://github.com/andrejbauer/agda/tree/release-2.7.0.1-sexp?tab=readme-ov-file,.

CHAPTER 11

Appendix

Some information, for example program listings, is useful to include within the report for completeness, but would distract the reader from the flow of the discussion if it were included within the body of the document. Short extracts from major programs may be included to illustrate points but full program listings should only ever be placed within an appendix. Remember that the point of appendices is to make your report more readable. I don't expect to see apprentices in any Project Proposals.