Benjamin Plattner & Olivier Lischer

Build-your-own-[grep, Redis] in Haskell

Term Project

OST – Eastern Switzerland University of Applied Sciences
Campus Rapperswil-Jona

Supervision

Prof. Dr. Farhad D. Mehta

December 2022

Abstract

“How can one write a ’real’ application in Haskell” is a common question by students after completing a functional programming course. Following the ’build-your-own-x’ concept, we provide two partial, but fully working implementations of the grep and Redis applications in Haskell.

The principle of algebra-driven design, also known as denotational design, is closely followed for these implementations. Both code bases are accompanied with detailed and reviewed explanations which serve as a tutorial for experienced software engineers and students alike.

All code, along with the explanations are released open-source and, in addition, are used at OST as part of the functional programming lecture.

Keywords: Haskell, Algebra-Driven Design, grep, Redis, open source.

Executive Summary

Often, the best way to learn something new is to use it. Once you get past a minimum understanding of how 'this new thing' works, applying it in a real project is the key to master it.

As a software engineer you use many tools without ever really looking at the inner workings. When you embark on the adventure of learning a new programming language, this poses the ideal opportunity to apply the fresh knowledge to something you are very familiar with as a user.

This is the idea behind the build-your-own-x concept, where x stands for an application of your liking.

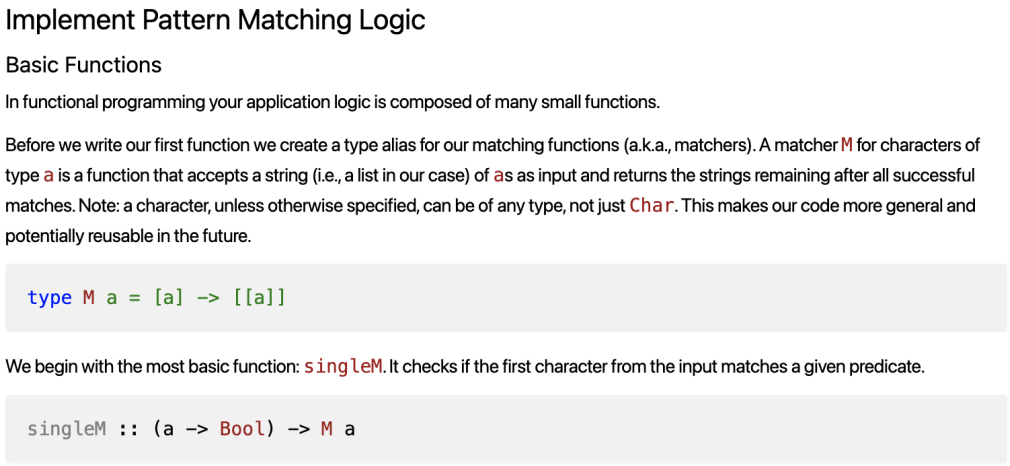
Many tutorials exist online which you can follow to implement your chosen tool in that new programming language you are just getting to know better. However, few of them are written in Haskell, a functional programming language known for its static and strong type system.

The purpose of this work is to provide detailed Haskell tutorials for two build-your-own-xs - grep and Redis. An example solution explanations is depicted in Figure 1.

Both tutorials are fully reviewed and publicly available as open-source projects.

This open-source repository is maintained by our project partner CodeCrafters.io, a start-up based in California that specializes in programming courses for experienced software engineers. In Figure 2 an instruction from one of their challenges is presented. The also provide a custom command line interface to submit solution attempts as illustrated in Figure 3.

The material is also used as part of the functional programming class at OST where students learn the fundamentals of Haskell.



Implement Pattern Matching Logic

Basic Functions

In functional programming your application logic is composed of many small functions.

Before we write our first function we create a type alias for our matching functions (a.k.a., matchers). A matcher `M` for characters of type `a` is a function that accepts a string (i.e., a list in our case) of `a`s as input and returns the strings remaining after all successful matches. Note: a character, unless otherwise specified, can be of any type, not just `Char`. This makes our code more general and potentially reusable in the future.

```
type M a = [a] -> [[a]]
```

We begin with the most basic function: `singleM`. It checks if the first character from the input matches a given predicate.

```
singleM :: (a -> Bool) -> M a
```

Figure 1: Example from a solution explanation for grep.

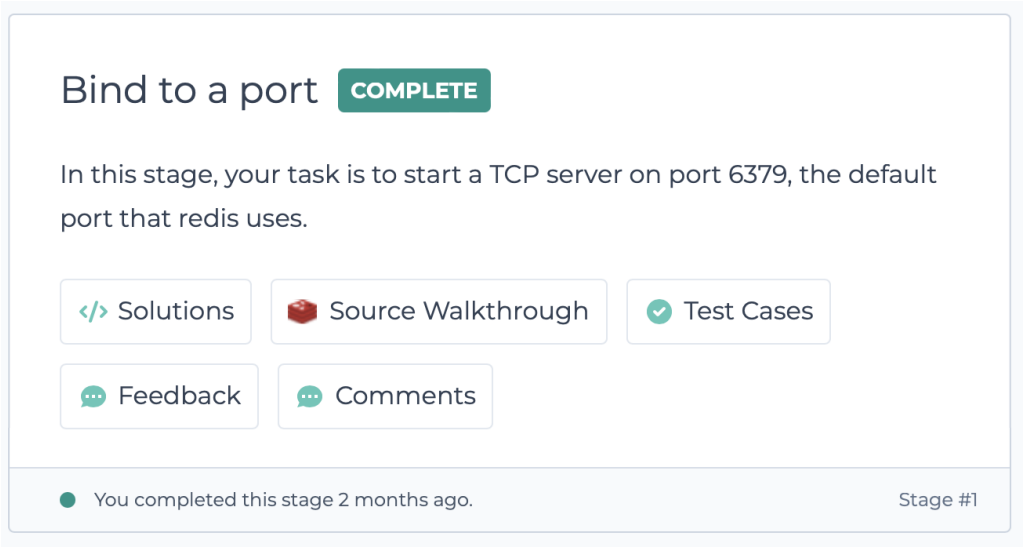


Figure 2: An example of a stage instruction for the Redis challenge on CodeCrafters.io's website.

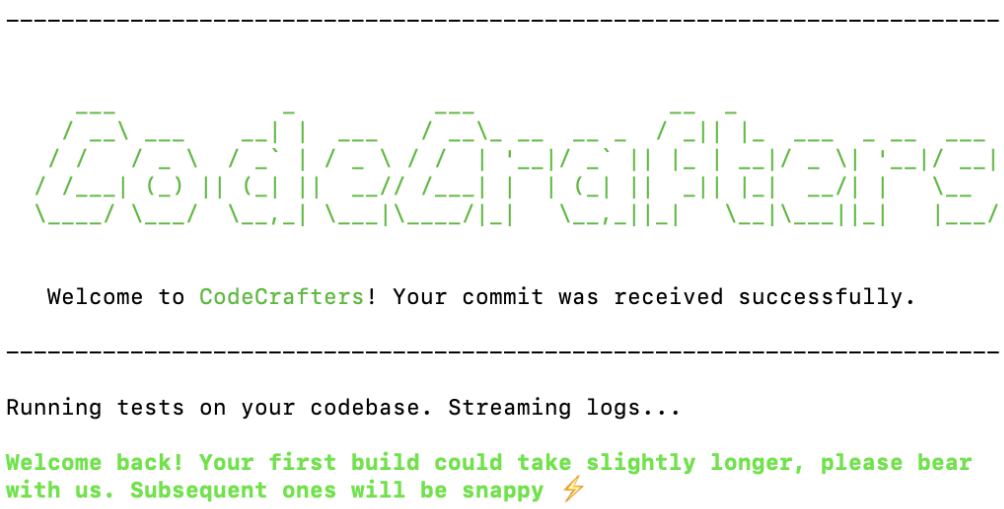


Figure 3: The command line interface when interacting with CodeCrafters.io's Git repository during a challenge.

Contents

Part I

Product Documentation

Chapter 1

Vision

Learning a new programming language is a daunting task, even for experienced software engineers. Studying the core concepts and the specific syntax of a language provides a solid foundation and gets you usually pretty far. Nonetheless, often the best way to really deepen your knowledge is by applying the language in a concrete example.

This project achieves two separate goals of learning a new programming language. First, it is about practicing the Haskell programming language. Second, it achieves this by providing structured coding examples in Haskell so that others who are learning the language can benefit from it.

The examples cover partial re-implementations of two known applications: grep and Redis. Building such an application from scratch is done in stages, where every stage extends the functionality of the previous one. If a user is stuck, there are extensive solutions available.

This thesis' results are published as an open source project, but they will also be available on CodeCrafters.io[1], a platform that allows experienced programmers to learn and improve their programming skills. Currently, this platform is missing implementations of these two applications in Haskell.

In addition, the design and implementation of the coding examples are based on the concept of Algebra-Driven Design[2] [1].

[1] https://codecrafters.io
[2] https://algebradriven.design

Chapter 2

Introduction

Learning a new programming language is a great way to expand ones knowledge and skill set. There are many tutorials and courses online which help you on this journey.

In fact, this thesis was motivated to deepen the Haskell knowledge of the two authors after completing the functional programming class OST.

This work was inspired by the build-your-own-x concept, which proposes to learn a new programming language by applying it to a project of your own. The x stands for an application that a learner wants to implement using the new programming language. It may be a tool that the learner is often using and knows quite well how it works as a user. Often, the inner workings are not that important in production use, but it would be interesting to learn about them.

The build-your-own-x concepts is not new itself, but recently, a platform called CodeCrafters.io which is based in California, took it to the next level by providing a streamlined user experience.

This thesis is motivated and inspired by their approach.

2.1 Objective

The main objective of this project is to provide the CodeCrafters.io platform additional Haskell support including detailed and reviewed code solutions and explanations.

Two applications were chosen, grep and Redis in agreement with CodeCrafters.io's CEO and CTO.

The full functionality of each tool is beyond the scope of this thesis. However, the main guidance of which parts to implement is bound to the staged learning process of Code-

Crafters.io. It defines that a user builds their application stage-by-stage and gets feedback from the platform at every step.

In addition, an Algebra-Driven Design approach is used to build these two applications in Haskell.

2.2 Structure of the Thesis

This is the public version of the report. It does not contain the project plan, legal documents, personal statements and meeting minutes.

The thesis is organized as follows:

Product Documentation In the product documentation only a subset of the relevant information related to each application is presented.

The main contribution, however, are the code solutions and the corresponding explanations, which can be found in the respective repositories for build-your-own-grep[1] and build-your-own-redis[2].

Appendix The appendix contains code reviews and task description.

[1] https://github.com/codecrafters-io/build-your-own-grep
[2] https://github.com/codecrafters-io/build-your-own-redis

Chapter 3

Requirements

The requirements are defined generally, but also specifically for each course implementation.

3.1 Scope

The two applications, grep[1] and Redis[2], are large applications that grew over many years. While it is not feasible to implement all the available functions during this project nor is it likely that a learner would have time to implement everything, we stick to the stages that are currently implemented in other languages on the respective CodeCrafters.io learning tracks.

Each stage of the re-implementation track should contain a working implementation in Haskell along with tests which will run and provide feedback when a user submits a solution. In case a user gets stuck there are solutions provided that help to overcome hurdles or to verify how one's working approach may be differently implemented.

All contributions are first published on a private repository that is set up for this project. Once everything is implemented and tested in a satisfactory way it is pushed to the CodeCrafters.io repository as well to make it available to their user base.

In addition, the functional programming class at OST will be using the material for teaching. For this purpose we create a separate and dedicated repository that contains the solution code and explanations. A script will be provided to set up the stages for the student along with documentation on how this script and repository are organized.

[1] https://www.gnu.org/software/grep/
[2] https://github.com/redis/redis

3.2 General Functional Requirements

The goal is not to write a complete clone of the original grep or Redis but to provide solution code along with explanations for learning and teaching.

CodeCrafters.io is a platform that provides solutions and code explanations for various programming languages, but it lacks code and solutions for Haskell.

One goal of this thesis is to be compliant with CodeCrafters.io's course definitions published on Github:

- Grep Course Definition[3].
- Redis Course Definition[4].

The two courses contain at the time of writing 12 and 7 stages, respectively.

3.3 Grep Requirements

3.3.1 Functional Requirements

The Functional Requirements are documented in the Git Repository[5].

3.3.2 Non-Functional Requirements

ID	NFR-GREP-1
Requirement	The software must be written in a Haskell style.
Priority	High.
Measure(s)	The software consists of many small functions.
Process	Manual review.
Results	See Figure D.1.

ID	NFR-GREP-2
Requirement	The software must have good quality.
Priority	High.
Measure(s)	All defined properties during ADD are satisfied.
Process	Software should be tested.
Results	See Github Actions[6].

[3] https://github.com/codecrafters-io/build-your-own-grep/blob/main/course-definition.yml
[4] https://github.com/codecrafters-io/build-your-own-redis/blob/main/course-definition.yml
[5] https://github.com/SA-HS22/grep/blob/main/doc/01_requirements.md
[6] https://github.com/SA-HS22/grep/actions/workflows/test.yml

3.4 Redis Requirements

3.4.1 Functional Requirements

The Functional Requirements are documented in the Git Repository[7].

3.4.2 Non-Functional Requirements

ID	NFR-REDIS-1
Requirement	The software must be written in a Haskell style.
Priority	High.
Measure(s)	The software consists of many small functions.
Process	Manual review.
Results	See Figure D.2.

ID	NFR-REDIS-2
Requirement	The software must have good quality.
Priority	High.
Measure(s)	All defined properties during ADD are satisfied.
Process	Software should be tested.
Results	See Github Actions[8].

3.5 OST Course Builder Script

The course builder script produces a clean repository for students out of an arbitrary template repository.

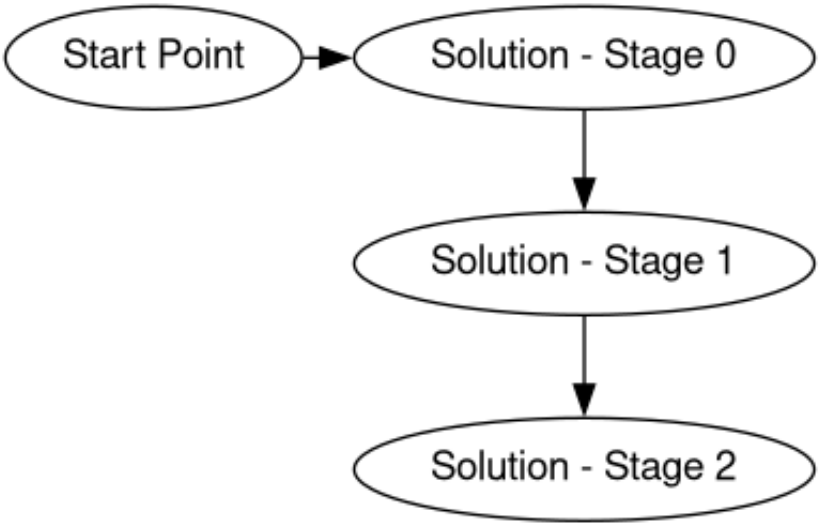
3.5.1 Functional Requirements

User Story 1 As a teacher I want to generate a Git repository from an existing repository. The existing repository contains the source code including the solutions for the different stages in different branches. The newly generated Git repository should have a history like in Figure 3.1.

User Story 2 As a student I want a simple workflow to view the example solution without leaving my coding environment.

[7] https://github.com/SA-HS22/redis/blob/main/doc/01_requirements.md
[8] https://github.com/SA-HS22/redis/actions/workflows/test.yml

Figure 3.1: Example Git History after generation



User Story 3 As a student I want to use the sample solution for a certain stage when I get stuck.

3.5.2 Non-Functional Requirements

There are no non-functional requirements specified for the Course Builder script.

Chapter 4

Architecture

In this chapter we describe the architecture for grep (section 4.1), Redis (section 4.2), the course builder script (section 4.3) and the CodeCrafters.io courses (section 4.4).

4.1 grep

grep is in its minimal form a RegEx engine. In the *Automaten und Sprachen* course at OST, Prof. Dr. Andreas Müller talks about RegEx and DFA. According to the script for the course [2] the most performant way to perform matching using RegEx is a DFA.

However, we decided to not implement a DFA for our implementation. Our goal is to provide an example solution how one could implement an application like grep in a typical Haskell way and not the most performant way.

The algebra is documented in the Git repository[1].

4.2 Redis

Redis is an in-memory data structure store.

The architecture of Redis itself is beyond the scope of this work, as we will focus on single functions only.

However, for this implementation we made some decisions, mainly that we use an abstract syntax tree to pass commands, keys and values between the parser and the execution function.

[1] https://github.com/SA-HS22/grep/

Similar to grep, we intend to demonstrate the advantages of Haskell, rather than to focus on the most performant implementation.

The detailed approach is documented in the Git repository[2].

4.3 OST Generator Script

The OST internal workflow should be as simple as possible. Therefore, we decided on using a “teacher” repository with a Python script.

The script is written in Python and automates the following steps:

1. Creation of a new repository for the students
2. Copy the files from the original to the new student repository
3. Create a Git history as specified in a config file

For each course a repository and a config file are required to generate a “student” repository. For more information, see the Git repository[3].

4.4 CodeCrafters.io Courses

Each course has its own Git repository (for example, build-your-own-redis[4]). The repository contains a starter template for each supported programming language. The starter template must be structured and formatted in a predefined way (subsection 4.4.1).

For each language there exists a subfolder containing the solutions for the stages. The solutions must follow a predefined layout (subsection 4.4.2).

4.4.1 Starter Template

The *starter_template* enables support for a new programming language. The source code in the *starter_template* must contain the following:

- A comment saying *“You can use print statements as follows for debugging, they’ll be visible when running tests.”* followed by a print statement.
- A comment out code section - the first line mustbe the *“Uncomment this block to pass stage 1”*

[2] https://github.com/SA-HS22/redis/
[3] https://gitlab.ost.ch/FP/Build_your_own_X/course-builder
[4] https://github.com/codecrafters-io/build-your-own-redis

The source code in the *starter_template* must still be valid after the code is commented.

The *starter_template* requires a bash script, which compiles and starts the application. To find the correct name for the script, check out an already existing *starter_templates* for other languages (for example for build-your-own-grep it is called “your_grep.sh”[5]).

4.4.2 Solutions

A solution consists of the following parts:

- *code* folder
- *diff* folder
- *definition.yml*
- *explanation.md*

The *code* folder contains the source code. The *diff* folder contains for each file a difference between the current stage solution and the previous one. The diffs must not be manually generated, but using a script that is part of CodeCrafters.io’s course SDK (see subsection 4.4.3).

During the testing using the SDK the diffs are compiled automatically. In the *definition.yml* file the author and reviewer for the solution are listed. Only one author is allowed (at the time of writing) but multiple reviewers are. An example of a *definition.yml* can be seen in Listing 4.1. The *explanation.md* contains an explanation in Markdown for the written code.

```
1  author_details:
2    name: Author Name
3    profile_url: https://github.com/author
4    avatar_url: https://github.com/author.png
5    headline: Student, Eastern Switzerland University of Applied Sciences ( ↵
   OST Rapperswil)
6
7  reviewers_details:
8    - name: Reviewer1 Name
9      profile_url: https://github.com/reviewer1
10     avatar_url: https://github.com/reviewer1.png
11     headline: Professor, Eastern Switzerland University of Applied Sciences ↵
    (OST Rapperswil)
12
13   - name: Reviewer2 Name
14     profile_url: https://github.com/reviewer2
15     avatar_url: https://github.com/reviewer2.png
```

[5] https://github.com/codecrafters-io/build-your-own-grep/blob/main/starter_templates/haskell/your_grep.sh

```
16    headline: Research Assistant, Eastern Switzerland University of Applied ↵
  Sciences (OST Rapperswil)
```

Listing 4.1: definition.yml example

The solution for *01-init* must not be created manually. The SDK will generate the solution for the first stage automatically based on the starter template.

4.4.3 Course SDK

The course-sdk[6] is a collection of scripts provided by CodeCrafters.io to test and validate a course. The SDK is in rapid development. Please see their course-sdk repository for up-to-date information about its current functionality.

[6] https://github.com/codecrafters-io/course-sdk

Chapter 5

Design

With Haskell one can build elegant functions that build on top of each other. The language lends itself very well to Algebra-Driven Design. This is a design principle that proposes small functions to have algebraic properties, such as identity or equality. Using the identity in ‘plus‘ function as an example, this would mean ‘plus 1 0 = 1‘, where ‘0‘ is the identity.

When designing an application in Haskell it is generally a good idea to start with ths principle in mind so that the more abstract the program gets, the more one can rely on small, algebraic functions.

This work relies to a significant degree on the work by Sandy Maguire and his book Algebra-Driven Design[1]. It explains in detail and with many examples how one can derive algebraic properties for functions, and how these properties can be implemented and also tested. To some, the algebra-driven design is also known as denotational design.

The documentation of the algebra-driven design approaches for both implementations can be found in the documentation sections in both repositories, grep[1] and Redis[2].

[1] https://github.com/SA-HS22/grep/blob/main/doc/02_design.md
[2] https://github.com/SA-HS22/redis/blob/main/doc/02_design.md

Chapter 6

Implementation

The implementations consist of solution code for each implementation along with detailed explanations for each stage.

The explanations contain the rationale why the implementation was done in a certain way and they are complemented with code examples from the solution code.

The results are all stored in the Git repository for grep[1] and Redis[2]. They consist of *explanation.md* files under */solutions/haskell/[stage]/* plus Haskell code file(s) in the */solution/haskell/[stage]/code* directory.

Both, code and explanations, underwent a rigorous review at OST by our supervisor and a research assistant prior to publication as a pull request. See the feedback from the reviewers in Figure D.1 for grep and Figure D.2 for Redis.

In addition, pull request reviews were performed by the CTO of CodeCrafters.io before each merge, see pull request #17 for grep[3] and pull request #58 for Redis[4].

This ensures high-quality implementations from which future students can learn and benefit.

[1] https://github.com/codecrafters-io/build-your-own-grep
[2] https://github.com/codecrafters-io/build-your-own-redis
[3] https://github.com/codecrafters-io/build-your-own-grep/pull/17
[4] https://github.com/codecrafters-io/build-your-own-redis/pull/58

Chapter 7

Results

To assess the results of this thesis, they are compared with its goals which were defined prior to starting the project.

7.1 Goals

A summary of the goals from the task description:

1. This project will provide the following for each challenge considered:
   (a) Haskell support (a starter repository) in a form usable by CodeCrafters.
   (b) A solution and its explanation in a form usable by CodeCrafters and as a stand-alone web page that can be linked to “Build your own X”.
2. The following minimum set of challenges will be considered: Redis, grep
3. Depending on the progress of this project, support (with solutions and explanations) for other challenges will be provided in the order of their relevance and user interest.
4. Given that Haskell follows the pure functional programming paradigm, which differs from the existing solutions and explanations on CodeCrafters, care must be taken to arrive at high-quality solutions and explanations that are idiomatic to this style of programming, and not direct translations from existing solutions.
5. The results of this project will be released under the MIT license, giving all stake-holders the right to use and further develop its results.

7.2 Verification of the Results

Haskell support (a starter repository) in a form usable by CodeCrafters. Both implementations, grep and Redis, provide starter repositories that contain code for the initial stage of a challenge.

In the case of grep this required the complete setup of the initial stage, since no prior Haskell support existed.

For Redis this was slightly simpler as the initial file structure for the initial stage was already present. However, the existing code had to be adapted slightly to include Haskell libraries used in later stages, as well as a simplification of the network connection setup.

A solution and its explanation in a form usable by CodeCrafters and as a stand-alone web page that can be linked to "Build your own X". Solution code and explanations for each stage were submitted to CodeCrafters.io in the form of a pull request. The grep implementation has, at the time of writing, already been merged with the main branch of the build-your-own-grep challenge. While Redis was still in the review phase, no major issues are expected to complete a merge later on.

The stand-alone web page is intended for the functional programming class at OST. It was agreed that Git repositories should be used instead of a web page since it is easier for the students to setup the challenges themselves. To simplify the setup process for the grep and Redis student repositories a Python script is provided that creates a clean version for each student out of the repositories.

The following minimum set of challenges will be considered: Redis, grep. Both implementations were successfully completed and published in the respective repository for build-your-own-grep[1] and build-your-own-redis[2].

Depending on the progress of this project, support (with solutions and explanations) for other challenges will be provided in the order of their relevance and user interest. Since both implementations required a significant effort and time a Haskell implementation of another challenge was not considered.

[1] https://github.com/codecrafters-io/build-your-own-grep
[2] https://github.com/codecrafters-io/build-your-own-redis

Given that Haskell follows the pure functional programming paradigm, which differs from the existing solutions and explanations on CodeCrafters, care must be taken to arrive at high-quality solutions and explanations that are idiomatic to this style of programming, and not direct translations from existing solutions. By adhering to the Algebra-Driven Design principles and by implementing feedback from in-depth reviews, high-quality solution code and explanations were achieved.

This is also confirmed by CodeCrafters.io's CTO (see pull request reviews #17 for grep[3] and #58 for Redis[4]).

The results of this project will be released under the MIT license, giving all stakeholders the right to use and further develop its results. The license under which CodeCrafters.io releases its open-source projects is MIT, however, the Haskell templates are licensed under BSD3. It is very similar to the MIT license, with minor differences, except that BSD3 restricts the use of the organization or the contributors without prior written consent.

The BSD3 license might be an oversight from CodeCrafters.io. We are discussing a pull request at the time of writing to change this license to MIT.

The release for the functional programming course at OST on the OST Gitlab platform is done under the MIT license, so that other users (for example research assistants or students at OST) can further enhance and use the solutions.

[3] https://github.com/codecrafters-io/build-your-own-grep/pull/17
[4] https://github.com/codecrafters-io/build-your-own-redis/pull/58

Chapter 8

Conclusion

This chapter discusses our findings and possible alternatives and extensions.

8.1 Findings

The main findings are primarily documented in the personal reports. However, for a brief summary without personal opinions, a list of findings is presented here.

8.1.1 Haskell

Haskell Experience In order to design an application from start to finish in an algebraic way, deep Haskell experience is required. Additionally, a good knowledge about existing Haskell libraries is beneficial so that the various functions can be designed and laws correctly derived.

Parser Libraries In addition to the Haskell experience, if an application requires to parse input, it is good advice to settle on a parsing library early on. This is especially true if one implements a parser for the first time. The reason being, that the parser often connects various functions as well as in- and outputs. Haskell does provide excellent parsing libraries.

IO Considerations When an application requires impure interactions with the outside world, it is helpful to consider where the IO type should appear. It is easy to implement IO almost everywhere, since once it is present, it cannot be avoided. However, with careful design the spread of IO can be limited.

In addition, IO makes testing much more complicated. When it is possible to avoid having the IO type in a function signature, this function can be tested much easier and more thoroughly.

8.1.2 Software Engineering

Design Early For a software engineer, the implementation is often the most exciting and rewarding part of the application development. In spite of that, it can also be a frustrating experience having to re-write larger parts of code if limitations of the design appear.

Therefore, starting early on with even a simple design helps to avoid such pitfalls.

8.1.3 Project Management

RUP Planning Tools Our research has shown that planning tools for a high-level RUP are difficult to find. They are often too cumbersome to set up, because a very detailed plan and time estimation is required.

The paid version (or free trial) of Monday.com has show to be the right level of abstraction for this thesis. One can plan in days and weeks, plus it generates time lines and Gantt charts automatically. For future work, this tool can serve as good planning tool for RUP.

Task and Time Management YouTrack proved to be an excellent management tool for this thesis. Mainly because it is very flexible how tasks, user stories and epics can be defined and connected. It is also a good tool for time management, since for every single task the time can be tracked. Custom reports can then be generated within the tool itself.

This has massively reduced the overhead for the duration of the project, even though a bit more time was required to get used to the tool and to set it up. In the end, the overall gain from YouTrack easily outweighed the upfront effort.

8.1.4 Project Partner

CodeCrafters The idea behind CodeCrafters.io is compelling. It provides a structured way to learn a new language but also to learn about the tool one is implementing.

Also, it showed how much effort is required to set up such a platform from scratch. As many processes as possible have to be automated to make such a platform scalable. It has served as great inspiration for this work and for future endeavors.

8.2 Future Work

To build on top of the existing functions and solutions, or even improve the functionality and performance, some ideas are listed what could be done in a future thesis or work.

Alternative grep implementation As in section 4.1 described we implemented grep not as a DFA. This is a possible solution and for this work performance is not important. In another thesis it might be an interesting idea to try out a DFA implementation and add it as an alternative solution. The Stanford University has already written about this topic[3].

Algebra-Driven Design For our implementations of grep and Redis we deliberately chose two different approaches. With grep we started after a short design phase relatively quickly with the implementation. This was a good way to get out a proof of concept quite quickly. Also, it helped to get an understanding of how the whole implementation could (or should) work.

This was a main benefit compared to how we approached Redis since we did not have to guess how a parser is ideally implemented or what a parser exactly does.

In a future Haskell or functional programming project, it would be reasonable to begin with the Algebra-Driven Design approach in any case. It does require familiarity with Haskell and with the key libraries that are planned to be used. Again, the benefits of starting early and thoroughly with the design outweigh the initial efforts.

Discovering Laws The algebra-driven design approach is about finding algebraic laws. As a designer one may find many, but not all possible combinations and hence might miss a law or two. It would therefore be interesting to use such advanced tools like QuickSpec, which was out of scope for this thesis, since it requires a deeper understanding of Haskell and more time to set it up. It may have helped to find more laws that were not that apparent from simple reasoning.

Add More Functions Both tools, grep and Redis, have only those functions implemented that are relevant for the CodeCrafters.io stages. One could, as a further exercise, implement additional functions which build on top of the existing design. The respective product documentations contain ideas for further functions.

Improve Course Builder A course builder script for the teacher-student repositories was created to provide a solid and scalable solution.

Researching the ideal way to present and structure the course at OST was outside the scope of this thesis.

However, in a future thesis, a more elaborate and better maintainable way, either in the form of a Git repository or a course website, could be designed and implemented.

Part II

Appendix

Appendix A

Task Description

OST
Eastern Switzerland
University of Applied Sciences

Haskell Support for CodeCrafters.io

Task Description

1. Setting

CodeCrafters[1] allows experienced programmers to improve their programming skills by rebuilding well known open-source software tools such as Git, Docker, Redis, grep and SQLite (referred to by CodeCrafters as 'challenges') in a language of their choice. There are several tutorials on the internet that focus on rebuilding commonly used tools for educational and recreational purposes. The site "Build your own X" [2] maintains a list of such tutorials. CodeCrafters offers an improvement over such isolated tutorials by improving their user experience using automated feedback and well-tailored uniform guidance. Its offerings appear to be well accepted amongst experienced developers and have recently gained support from "Y Combinator", an American technology start-up accelerator.

Given current user interest, CodeCrafters would like to provide support to allow its users to use the programming language Haskell for its existing challenges. This would be beneficial for both, CodeCrafters, as well as the Haskell community. The only thing that currently exists in this area is a Haskell starter repository (referred to by CodeCrafters as 'Haskell support') for the Redis challenge without any solutions or explanations.

2. Goals

The main aim of this project is to provide Haskell support, with solutions and explanations, for some of the current programming challenges offered by CodeCrafters.

The following is a brief and unstructured list of initial requirements and notes:

1. This project will provide the following for each challenge considered:
    a. Haskell support (a starter repository) in a form usable by CodeCrafters[3].
    b. A solution and its explanation in a form usable by CodeCrafters and as a stand-alone web page that can be linked to "Build your own X".
2. The following minimum set of challenges will be considered: Redis, grep
3. Depending on the progress of this project, support (with solutions and explanations) for other challenges will be provided in the order of their relevance and user interest.
4. Given that Haskell follows the pure functional programming paradigm, which differs from the existing solutions and explanations on CodeCrafters, care must be taken to arrive at high-quality solutions and explanations that are idiomatic to this style of programming[4], and not direct translations from existing solutions.

[1] https://codecrafters.io
[2] https://github.com/codecrafters-io/build-your-own-x
[3] https://github.com/codecrafters-io/languages/blob/master/docs/adding_support_for_a_new_language.md
[4] For instance, using algebra-driven design https://algebradriven.design

OST
Eastern Switzerland
University of Applied Sciences

5. The results of this project will be released under the MIT licence, giving all stakeholders the right to use and further develop its results.

Further refinements and modifications to this list that serve the main aim of this project are possible during its course.

3. Deliverables

- All artefacts (source code, web pages, accepted pull requests to CodeCrafters, etc.) required to achieve the goals of this project.
- Product documentation in English that is relevant to the use and further development of the results (e.g., requirements, domain model, architecture description, code documentation, user manuals, etc.) in a form that can be developed further and is amenable to version control (e.g., LaTeX or Markdown). Ideally, all product documentation should be contained withing the artefacts required to achieve the goals of this project as stated in above.
- Project documentation that is separate from the product documentation that documents information that is only relevant to the current project (e.g., project plan, time reports, meeting minutes, personal statements, etc.).
- Additional documents as required by the department (e.g., poster, abstract, presentation, etc.)
- Any other artefacts created during the execution of this project.

All deliverables may be submitted in digital form.

4. Stakeholders

Industry Partner: Sarup Banskota, Rohit Paul Kuruvilla, CodeCrafters.io (San Francisco, USA)
Students: Olivier Lischer, Benjamin Plattner.
Supervisor: Farhad Mehta.

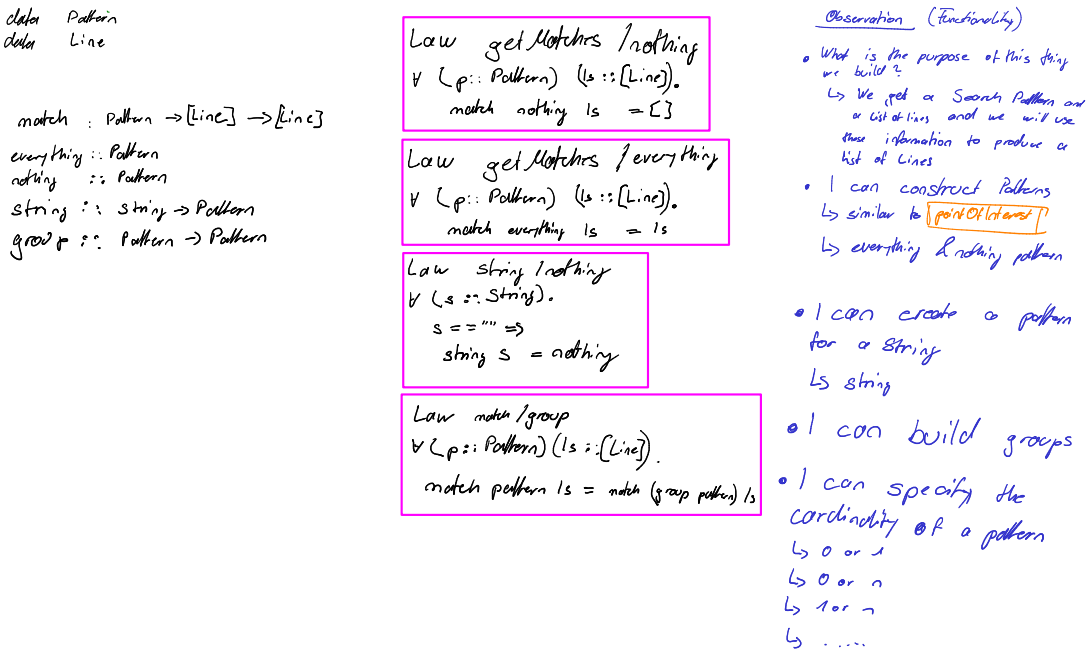
5. Other Project Details

Type of project: Study Project (de: Studienarbeit)
Duration: 19.09.2022 – 23.12.2022
Workload per student: 8 ECTS (1 ECTS = approx. 30 Hours)

Figure A.1: Task description for this thesis.

Appendix B

Grep Whiteboard Sketches



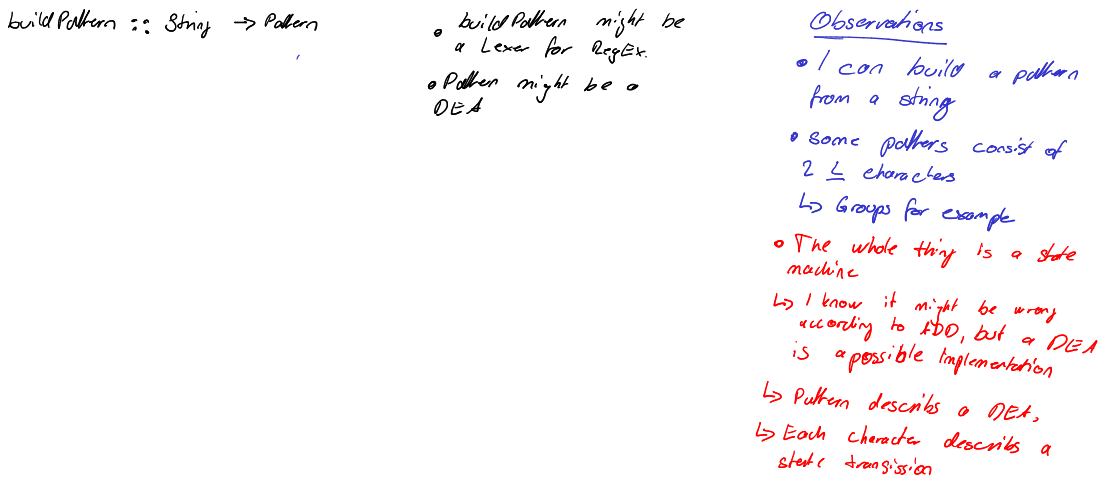


Figure B.1: A first attempt of using ADD with Redis.

Appendix C

Redis Whiteboard Sketches

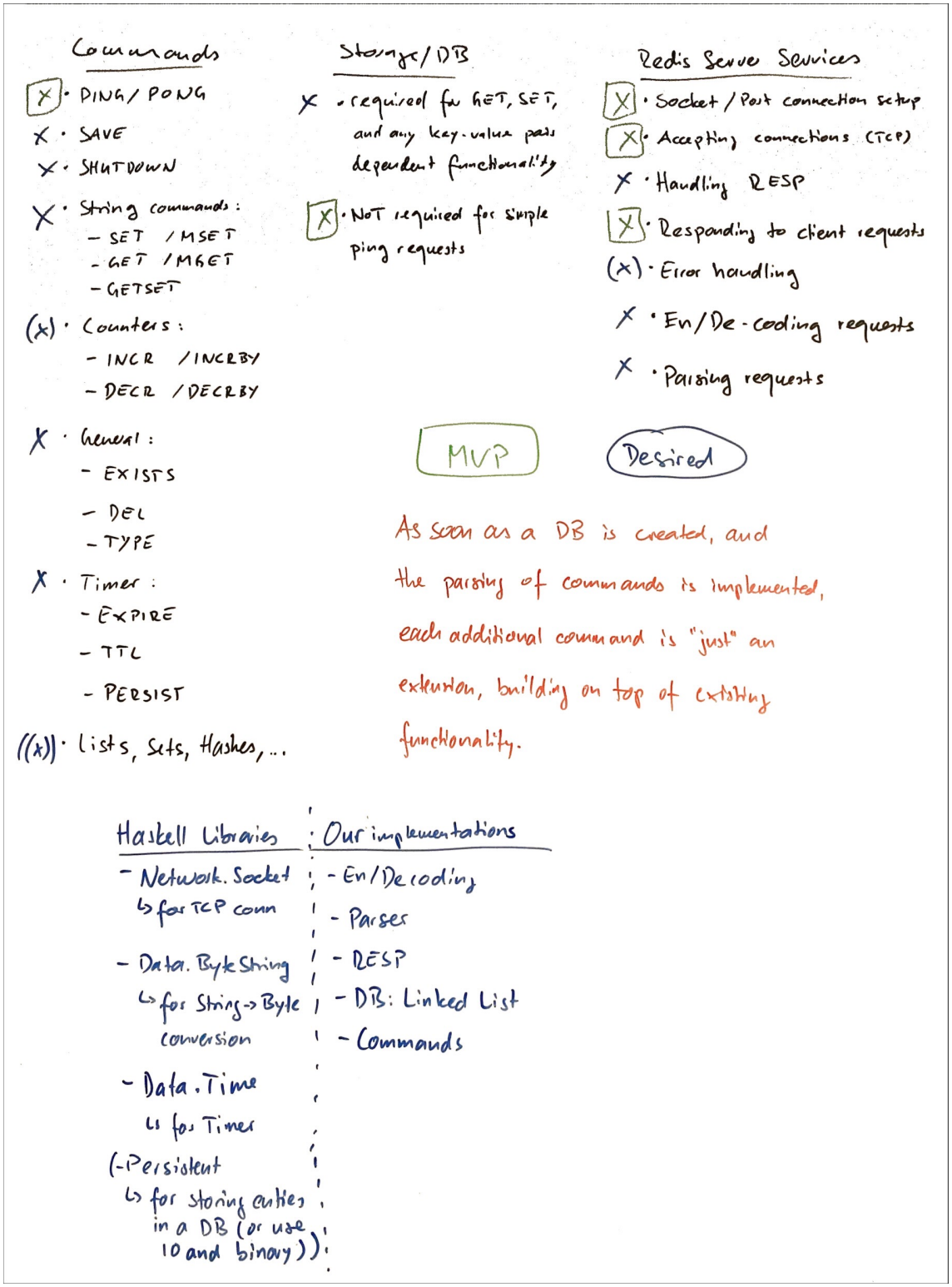


Figure C.1: Draft for scope of Redis implementation.

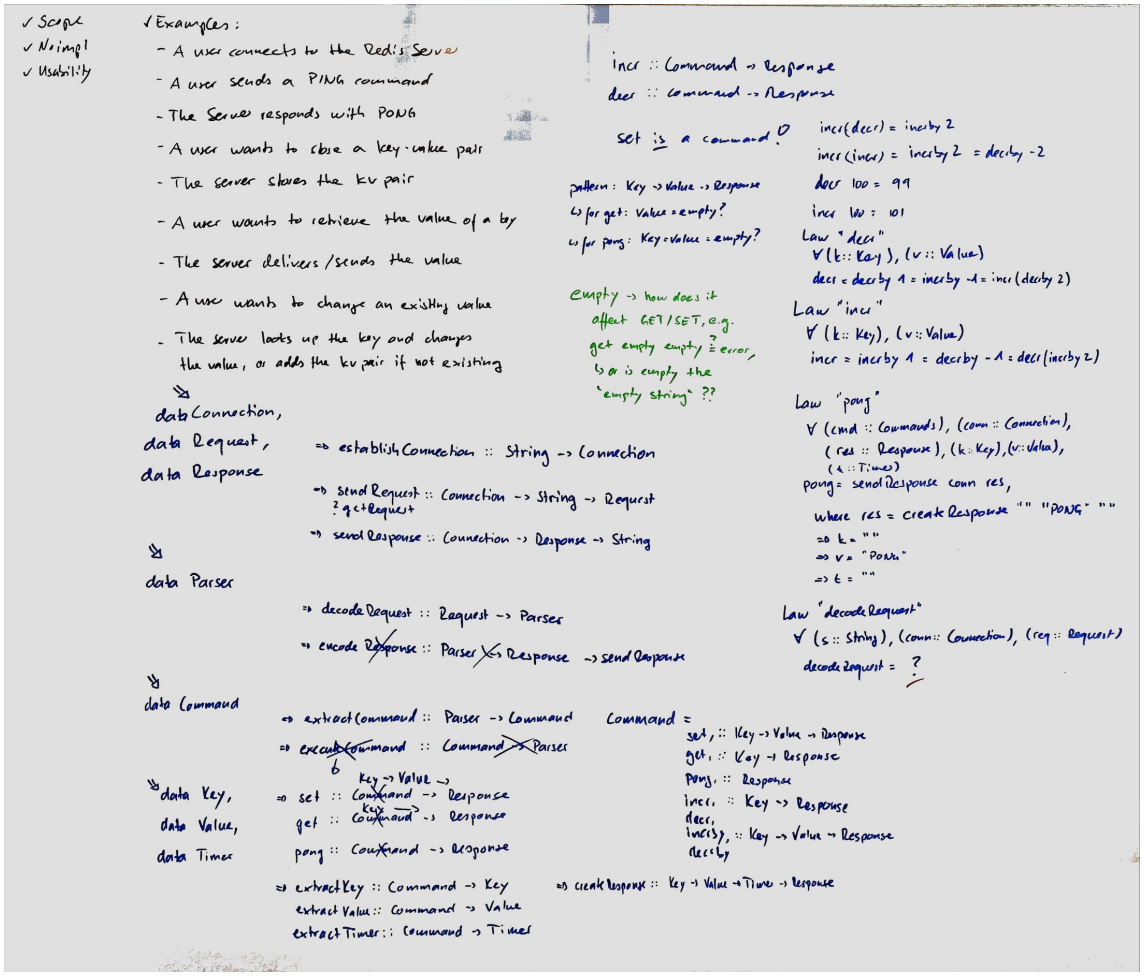


Figure C.2: A first attempt of using ADD with Redis.

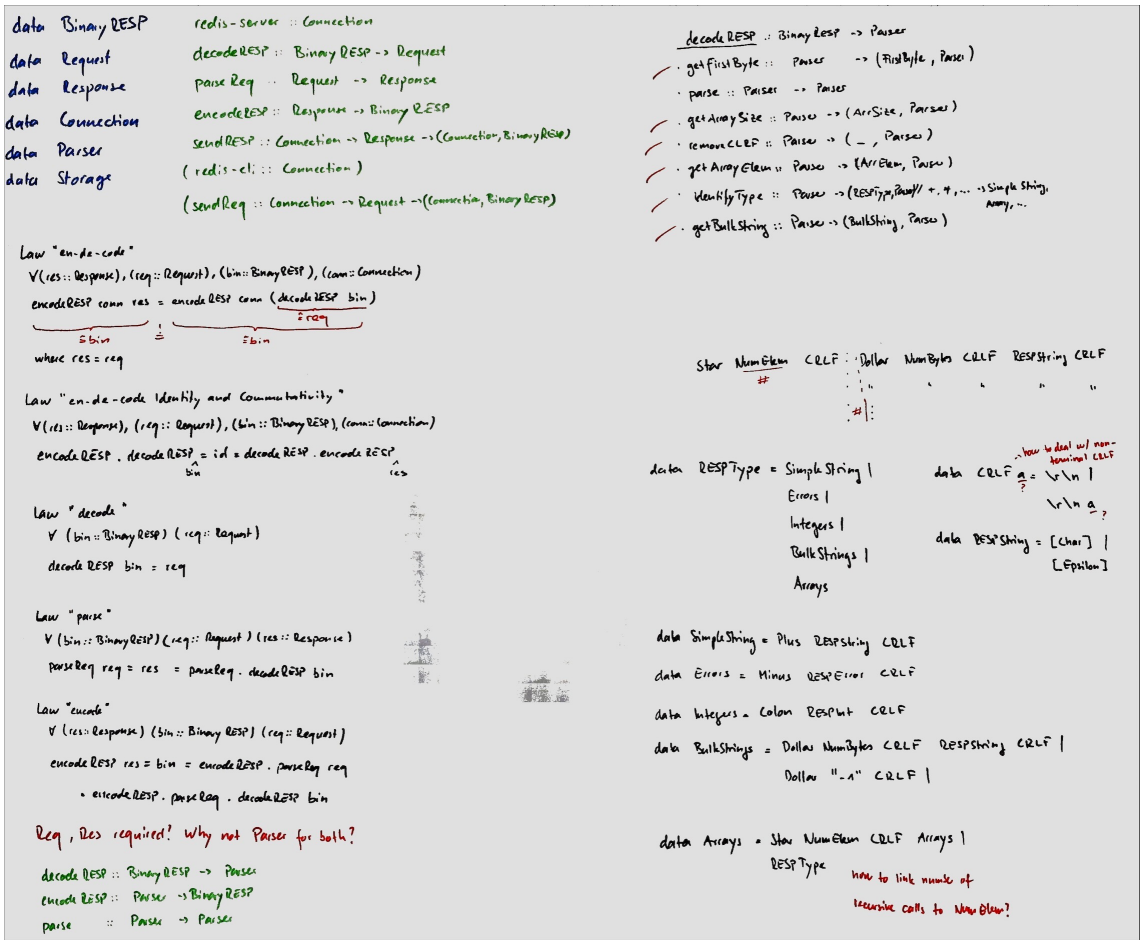


Figure C.3: An improved attempt with ADD and Redis.

Appendix D

Code Reviews

I looked at your grep code and overall I think the design choices and code are fine. Below are my thoughts and suggestions.

- Writing the regex engine this way is an interesting alternative to building a finite automaton. returning a function from the parser is kinda interesting. As mentioned, there are some benefits though to building an AST.
- is it correct that you only grep one line? If so, why not read from stdin or add a -f flag for a filename and run grep in a loop on every line like the real grep? (This should be very easy to implement in your current version)
- I believe your unknown quantifier detection in `quantifier` on line 41 of Parser.hs doesn't work the way you expect (assuming I get your intent right). matching the regexp 'hel{2}o' on the string "hello" returns nothing, but should return "hello". Assuming you want to detect unsupported quantifiers, such as {5} or [0-9], you have to do it differently. I'm basing this on the fact that you require the "-E" flag, which suggest you intend to support a subset of the extended regexp syntax.
- the same can be said (again, assuming I get your intention right) about the pos/neg unsupported input string detection on lines 23 and 31 of Parser.hs.
- there is one more inconsistency regarding `.` extended regexps don't actually support `.` they only support []. the easiest way to fix this would be to just mention it in the SA report that you deviate from the standard there.
- just for the sake of it, it would have been an interesting approach to build a finite automaton in Haskell for this task. have you considered it or do you maybe mention it in the SA report?

I can't say much about code smells or anti patterns, as you follow the megaparsec style of building a parser and use the ebnf from the repo that you link to in the source file comment. All the functions are small and concise without much potential to mess things up, which makes your codebase clear and easy to understand.

1

Figure D.1: Code review by a Research Assistant at OST for grep.

I've looked at your redis implementation and written my feedback below. I suggest you primarily look at the "Code Smells" part and try to fix some of that because it should be quite easy. The "Design" part is more difficult to address and also requires restructuring the codebase to some extent. I've included it mostly for you as "things to think about" for future projects and topics to look into if you want to continue your Haskell journey.

Good

- code is easy to read & understand, even for beginners I think
- you've kept it simple, using simple haskell language constructs and still managed to craft a nice codebase
- because the codebase is small, the code-smells and design-issues are more tolerable

Code smells:

- consider packing all constants (lines 53,59,72,75,200) into an ADT (data Configuration…). optionally: read it from a config file
- the meaning of magic string "px" on line 142 is not obvious
- the meaning of magic string "$" on line 129 is not obvious
- the type synonym on line 87, `parseRequest :: Response...` reads strange. maybe replace `Response` with `ByteString`
- `crlfAlt` on lines 100 & 101 looks redundant
- `echo` and `ping` on lines 183-187 are equivalent

Design:

- IO exception handling is missing. Consider using bracket. As it stands now, you technically have a resource leak in your socket connection, but since the whole app probably crashes completely when the connection throws an exception, cleanup isn't as crucial.
- the error string "-ERR unknown command"" on line 84 seems to be the only application error in the app. Using it in `fromRight` suggests it's a meaningful default value though, while in reality it's an error masking itself as a default value. This isn't as clean as error handling could be. Using a simple ADT `data ApplicationError = UnknownCommand` could make your intent clearer, but requires a bit of a redesign.
- the `Parser Command` parsers all hold a reference to the DB inside `Command` and execute the IO logic *within* the parsing logic. Parsers should be pure and not do IO. The IO part is only used at the end of those functions though, when you call the corresponding redis actions (get, set etc.). Returning a pure ADT that encapsulates your commands (data Command = Get Key | Set Key Value (Maybe Time) …) would make the parser pure and pose the added benefit of having a clean API description in the form of a `Command` ADT.

- The `Responses` from your redis API are raw bytestrings, but they conform to the redis API spec. Consider encapsulating them inside a `data Response = GetResponse | SetResponse | PingResponse...` ADT to make the API more explicit.
- To summarize the points in the "Design" part (If you were to fix them e.g. for a version 2 of the project):
  - use bracket for IO exceptions & resource handling/cleanup
  - use an `ApplicationError` ADT to structure your invalid states
  - make the parser pure
  - use a `Command` ADT to structure your redis api requests
  - use a `Response` ADT to structure your redis api responses
  - make the parser return a simple "AST" in the form of the `Command` ADT
  - use the results from the pure parser in your redis actions (get, set, echo)

2

Figure D.2: Code review by a Research Assistant at OST for Redis.

Glossary

Algebra-Driven Design Algebra-Driven Design takes a heavy focus on designing leak-free abstractions, on understanding programs so well that the code and tests can be largely generated automatically, and on finding performance improvements not via intuition, but through algebraic manipulation of the program's underlying equations. [1] 2, 4, 13, 17, 20, 38

BSD3 This version of the BSD license allows unlimited redistribution for any purpose as long as its copyright notices and the license's disclaimers of warranty are maintained. The license also contains a clause restricting use of the names of contributors for endorsement of a derived work without specific permission. [Wikipedia] 17

CodeCrafters.io Web platform to recreate existing tools like Redis, Git, Docker in your favorite language with the goal to learn and improve your programming skills. See also `https://codecrafters.io` iii, iv, 2, 3, 5, 6, 9, 11, 12, 14, 16, 17, 19, 20, 39

Deterministic Finite Automata A finite state machine running deterministically through the states. [Wikipedia] 38

grep grep is a command-line utility for searching plain-text data sets for lines that match a regular expression. [Wikipedia] 2, 3, 5, 6, 9, 10, 13, 14, 16, 17, 20, 33, 39

MIT The MIT License is a permissive free software license originating at the Massachusetts Institute of Technology (MIT)in the late 1980s. As a permissive license, it puts only very limited restriction on reuse and has, therefore, high license compatibility. [Wikipedia] 17

Monday.com Shared workspace for collaboration and project planning. See also `https://monday.com` 19

QuickSpec QuickSpec takes your Haskell code and, as if by magic, discovers laws about it. You give QuickSpec a collection of Haskell functions; QuickSpec tests your functions with QuickCheck and prints out laws which seem to hold. [Hackage] See also the QuickSpec library on Hackage
https://hackage.haskell.org/package/quickspec 20

Rational Unified Process The Rational Unified Process (RUP) is an iterative software development process framework created by the Rational Software Corporation. [Wikipedia] 38

Redis Redis (Remote Dictionary Server) is an in-memory data structure store, used as a distributed, in-memory key–value database, cache and message broker, with optional durability. [Wikipedia] 2, 3, 5, 6, 9, 13, 14, 16, 17, 20, 29, 35, 39

Regular Expression Regular Expression is a sequence of patterns that specifies a search pattern. [Wikipedia] 38

YouTrack A project management tool that can be adapted to processes to help deliver products. See also https://www.jetbrains.com/youtrack/ 19

Abbreviations

ADD Algebra-Driven-Design (See: Algebra-Driven Design) 6, 7, 27, 30, 31, 39

DFA Deterministic Finite Automata (See: Deterministic Finite Automata) 9, 20

RegEx Regular Expression (See: Regular Expression) 9

RUP Rational Unified Process (See: Rational Unified Process) 19

List of Figures

List of Tables

List of Listings

Bibliography

[1] S. Maguire, *Algebra-Driven Design*. Leanpub.com, 05 2021.

[2] A. Müller, “Automaten und Sprachen,” accesssed on 26 November 2022. [Online]. Available: https://github.com/AndreasFMueller/AutoSpr

[3] “Haskell - Regex Calculus,” accessed on 26 November 2022. [Online]. Available: https://theory.stanford.edu/~blynn/haskell/re.html