

Chesskell: Embedding a Two-Player Game in Haskell's type system

3rd Year Project Specification

Toby Bailey

October 14, 2020

Contents

1	Introduction	2
2	Objectives	2
2.1	Formal Project Requirements	3
2.1.1	Functional Requirements	3
2.1.2	Non-Functional Requirements	3
3	Methodology	4
3.1	Software Development Methodology	4
3.2	Evaluation and Testing	4
4	Timetable/Plan	4
4.1	Weeks 1-2: 5th to 18th October	4
4.2	Weeks 3-4: 19th October to 1st November	5
4.3	Weeks 5-6: 2nd to 15th November	5
4.4	Weeks 7-8: 16th to 29th November	5
4.5	Weeks 9-10: 30th November to 13th December	5
4.6	Weeks 11-14: 14th December 2020 to 10th January 2021	5
4.7	Weeks 15-18: 11th January to 7th February	6
4.8	Weeks 19-22: 8th February to 7th March	6
4.9	Weeks 23-24: 8th March to 21st March	6
4.10	Weeks 25-29: 22nd March to 25th April	6
5	Resources, Risks, and Ethical Considerations	6
5.1	Resources	6
5.2	Risks	7
5.3	Legal, Professional, and Ethical Considerations	7

1 Introduction

In 2020, video games are more popular than ever. In the US alone, an ESA report¹ estimates that there are above 214 million individuals who play games. Considering this, it's surprising how many games are released with major bugs in their software—some of which end up being so notable that news and footage of them appear on mainstream media, such as BBC News².

In recent years, programming languages have evolved to address runtime bugs at compile time. Features like optional types are being introduced to languages such as Java and C#, and languages like Rust have pioneered ways of safely handling dynamic allocation through the use of owner types³.

Recent versions of the *Glasgow Haskell Compiler* (GHC) support programming at the type level, allowing programmers to compute with types in the same way that languages like C or Python compute with values[1]. Type-level computations are run at compile time, before an executable of the source code is generated, allowing programmers to transform logic errors into type errors[2]. Programming in this way would enable game developers to eliminate many kinds of runtime problems in their games.

The aim of this project is to demonstrate a proof-of-concept; that it is possible to model Chess at the type-level, and that compiled programs comply with the rules.

2 Objectives

The project, nicknamed Chesskell, has the main aim of modelling the classic board game Chess in Haskell's type system. This type-level model will be interacted with via a Haskell-embedded Domain-Specific Language (DSL), for describing games of chess. This Embedded DSL (EDSL) will be modelled on Algebraic Notation, a method of writing down the moves associated with a particular match of chess.

The resulting program will take in as input a Haskell source file, describing one or more matches of chess in the EDSL. This file will be compiled, and in doing so, the type-level model of chess will be compared against the game described by the EDSL, and the compilation will fail if the described game breaks the International Chess Federation (FIDE) rules of chess.

The success of this project will be gauged against a data set of chess games, expressed via the EDSL; many of which will be incorrect and break the rules of chess in one way or another. The final program must successfully compile the games which do not break the rules, and must fail to compile those which break them.

The term "game" is rather loose, and encompasses many physical actions and software programs. Chess is complex, has a win condition, and involves multiple players;

¹https://www.theesa.com/wp-content/uploads/2020/07/Final-Edited-2020-ESA_Essential_facts.pdf

²<https://www.bbc.co.uk/news/technology-50156033>

³<https://doc.rust-lang.org/book/ch04-01-what-is-ownership.html>

modelling it at the type-level is a challenge, and would prove that type-level computation is fit for eliminating certain errors in video games.

2.1 Formal Project Requirements

2.1.1 Functional Requirements

- The system will contain a type-level model of chess, which will:
 - Support all rules within the FIDE official chess rules.
 - Have an internal representation of the board, the pieces, and other information relevant to the state of play.
 - Be able to detect when a rule of chess has been broken, and give out a compile-time error to explain which rule was broken and how.
- This type-level system will run on the Glasgow Haskell Compiler.
- This type-level system will be accessible via an Embedded Domain-Specific Language, which will:
 - Describe a game of chess, move by move.
 - Be written in Haskell source files, and be valid Haskell according to the Haskell grammar.
 - Be type-checked by the type-level model, and will cause errors if the game described breaks the established rules of chess.

2.1.2 Non-Functional Requirements

- The compilation overhead incurred by the system must not cause the compile time of a Haskell file containing chess EDSL information to exceed 5 minutes on a consumer-grade PC.
- The EDSL will be based on Algebraic Notation, and must be intuitively readable by someone who is familiar with chess⁴.
- The errors reported by the system must be helpful and comprehensive, including information such as:
 - The specific move that broke the chess rule, where the first move in the game is known as Move 1.
 - The piece which broke the rule, as well as that piece's colour.
 - Any other relevant information as to why that move breaks the rule.

⁴More simply, it must not look like unintelligible Haskell.

3 Methodology

3.1 Software Development Methodology

The most appropriate methodology for developing this program is an incremental development approach. The software will be designed in broad strokes, before it is built piecewise. Function by function, the system will be developed and put together, with unit testing along the way, until the functions can be wired together to form the final product.

The above is a form of spiral methodology, adopting a waterfall-like process for each part of the project, before being tested as a whole.

3.2 Evaluation and Testing

As noted earlier, the final project's success will be evaluated against a curated data set of chess games. However, binary success is not the only helpful metric; the final project must compile within a reasonable time, and must output helpful errors that explain the reason for failure (should it occur).

Compiling within a "reasonable time" is nebulous; if the program compiles within 5 minutes on a consumer-grade laptop, then it will be enough. Errors must specify the exact move which breaks the rule, which piece broke it, and the way in which the rule was broken.

For example, "Error - broken rule" is unacceptable, and should be replaced with something more akin to the following:

Error on move 6:

Black Pawn has moved to d4, but it can only move to: d5, d6.

Of course, the specific error and relevant information will depend on the exact rule broken; but error messages which point to the exact transgression are *always* helpful.

4 Timetable/Plan

The plan for the project will be split into two- or four-week chunks, dating from October 2020 until April 2021. Certain portions of the project will be allocated as empty space; should something go awry, having breathing room to fix it would be ideal.

4.1 Weeks 1-2: 5th to 18th October

The initial research stage will take place over these two weeks; multiple approaches to type-level computation and modelling of programs will be considered, and an appropriate one will be selected. Work on modelling chess at the type-level will begin at the same time, to perform an initial general test of different approaches' feasibility.

The system will be designed in a broad-strokes level, with function⁵ signatures written out.

4.2 Weeks 3-4: 19th October to 1st November

Once the broad-strokes design is complete, the type-level model will be implemented in full. Unit tests will ensure each individual function works; at which point the model will be wired together.

4.3 Weeks 5-6: 2nd to 15th November

At this halfway point, either the type-level model is well underway (or even complete), or problems have been encountered. This is the initial opportunity to steer the project in a different direction, outlined in the Risks section below.

However, assuming that the project is going well, this is the time to draw out the formal grammar for the EDSL, planning out how it will function, how it will interact with the type-level model, and how it will be interacted with.

4.4 Weeks 7-8: 16th to 29th November

The EDSL should be fully planned out, and as such this section of time will be taken to implementing it in its' entirety, with internal daily deadlines for features.

Another module, CS325 Compiler Design, has a coursework due on the 23rd November. This project's progress report is due shortly afterwards, on the 30th November. As such, delays are very possible during this period; but as the EDSL will be specified in detail, implementing it should not take excessive time. The empty space allocated for the Christmas break should help with any issues.

4.5 Weeks 9-10: 30th November to 13th December

This space is set for testing of the (near-complete) system against a curated data set, and fixing any potential problems that arise. The test will involve piping the data set to the type-level model via the EDSL, and as such will be integration and system testing, with unit testing taking place during development.

4.6 Weeks 11-14: 14th December 2020 to 10th January 2021

This is the first section of allocated empty space; should some of the risks in the below section materialise, this is when the project may be caught up on. Should the project be going well, then this extra time will be spent either starting early on the dissertation,

⁵At the type-level, it will be *type family* signatures. However, the word "function" is used here for simplicity.

or adding extensions to the project to further explore the modelling of games at the type-level.

4.7 Weeks 15-18: 11th January to 7th February

Writing the dissertation is planned for this stretch of time; it will be planned out, section by section, with any relevant graphs, figures, and citations gathered.

The module CS324 Computer Graphics has a coursework due on the 20th January; as such, planning the dissertation could take longer than expected. However, since a month is set aside for just planning, delays are unlikely.

4.8 Weeks 19-22: 8th February to 7th March

Once the previous detailed planning stage is complete, writing shall begin; with a detailed enough plan, this section should not take longer than a month. An initial draft will be completed by the 7th of March.

The project itself will be evaluated during this period, examined for any subtle bugs, and the code will be finalised and completed. Work on the project presentation should also start in earnest, writing out a script.

4.9 Weeks 23-24: 8th March to 21st March

These two weeks are more empty space; set aside to act as a buffer for delays in dissertation writing. The project presentation is expected on the 19th March; so this empty space will ideally be spent completing the presentation and rehearsing it.

4.10 Weeks 25-29: 22nd March to 25th April

Drafting, re-drafting, and refining the dissertation (with the help of the supervisor) will take place during this time. This is the final stretch, and will be spent ensuring that the final piece of writing is as good as it can be.

Revision for examinations is key during this period; however, since the dissertation should be complete and this period is for evaluation and not the main bulk of writing, there should be ample time.

5 Resources, Risks, and Ethical Considerations

5.1 Resources

Developing this program requires a computer (and Operating System) which can run the latest version of GHC (8.10.2, at the time of writing). GHC runs on most Linux distributions, as well as Windows and MacOS. In other words, it runs on almost all

consumer-grade PCs, and so development is unlikely to be a problem. The development environment will be the same as the "production" environment; it is a tool designed for those involved in the compilation process.

There are many Chess rulesets, casual or otherwise. The project will use the FIDE official ruleset, because it gives comprehensive, detailed rules that are widely used⁶.

5.2 Risks

The biggest risk with this project is that it is unachievable. Haskell's type-level computations are limited in scope, and other languages (such as Idris) have dependent types and so can express things at the type-level that Haskell may not be able to yet.

Should this be the case, the focus will switch from implementing all official chess rules, to a subset of them (to be selected at the time of failure).

If this also fails, the project will be re-attempted in a fully dependently-typed language, such as Idris; and if even that should fail, then there will be lots of work done on implementing Chess at the type-level, and there will be a specific reason as to why it is impossible. Future research could then build on this foundation, attempting to find ways around this problem.

There are, of course, other risks not directly associated with the project; in 2020, there is a large-scale pandemic going on, which could pose considerable personal risk. However, by following Government and University guidelines, the chances of contracting COVID-19 will be kept to a minimum, and the project should not be interrupted by sudden illness.

Planning out the project, as above, to account for other deadlines and busy periods during University life should also help mitigate the risk of delays.

5.3 Legal, Professional, and Ethical Considerations

As this project is undertaken alone, with no involvement from third parties (aside from the project supervisor), there are no additional legal, professional, or ethical considerations to consider. There will be no personal data processing, and no pre-existing Haskell type-level Chess implementation to plagiarise.

References

- [1] B. A. Yorgey, S. Weirich, J. Cretin, S. Peyton Jones, D. Vytiniotis, and J. P. Magalhães, "Giving haskell a promotion," in *Proceedings of the 8th ACM SIGPLAN workshop on Types in language design and implementation*, pp. 53–66, 2012.
- [2] S. Maguire, *Thinking With Types: Type-Level Programming In Haskell*. 2018.

⁶It also includes rules for the players themselves, but these meta-rules are beyond the scope of the project.