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

3rd Year Project Specification

Toby Bailey

October 6, 2020

Contents

1 Introduction

In 2020, video games are more popular than ever. In the US alone, an ESA report [1] 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 News [2].

In recent years however, 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 types.

Haskell, in recent *Glasgow Haskell Compiler* (GHC) versions, has supported programming at the type level, allowing programmers to compute with types in the same way that languages like C or Python compute with values. Type-level computations are run at compile time, before an executable of the source code is generated. This type-level toolbox allows programmers to transform business logic errors into type errors—and this style of programming 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 games at the type-level, and ensure that programs can only be compiled when they follow the rules of the game.

2 Problem Statement

The project, named Chesskell, has the main aim of modelling the classic board game Chess in Haskell's type system. Chess is a complex logic-based game, with many moving parts, which contains move that can have side-effects (for instance, moving a Pawn to the opposite side of the board transforms that Pawn into a Queen).

Chess is for two players/teams, who are denoted Black and White after the colour of their numerous pieces, who act in alternating turns until one wins the game. A player cannot choose inaction for their turn; they must move one of their pieces, if they can. A player wins by putting the opposing team's "King" piece in a position whereby all moves it could make would put it in the direct path of another piece. All pieces have a certain area that they can "take" within, enabling them to remove other pieces from play. (For instance, Rook/Castle pieces can take any piece of the opposite team which is in the same horizontal or vertical axis on the board, with no other pieces in the way.)

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; 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.

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

[2] `https://www.bbc.co.uk/news/technology-50156033`

This type-level model will be interacted with via a Haskell-embedded DSL, for describing games of chess. This 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.

3 Project Requirements

3.1 Formal Requirements

The below list splits the requirements into *Functional* and *Non-Functional* requirements, employing the MoSCoW system to explain the scope of the project.

4 Methodology

4.1 Software Development Methodology

4.2 Evaluation and Testing

5 Timetable/Plan

6 Resources, Risks, and Ethical Considerations

6.1 Resources

6.2 Risks

6.3 Ethical Considerations