Report

I have tried to create a parser which uses functors, monads, applicative, Either data type and do notation. To parse beans gambit and BGN notation, I started out with parsing pieces and piecetype. Using pattern matching and case of construct, I mapped different characters to different piecetypes. Since for a BGN notation (first) move to be valid or invalid, the corresponding must be able to be performed on the starting position, therefore I attempted to parse board, parse row and parse each piece on each cell. Choice was used since parseEmpty, parseRed, parseBlue can be considered as booleans which are then passed onto parsePiecetype parser. Monadic for all operator is used in the definition of parseRed and parseBlue functions to apply the Just constructor to the result of parsing a PieceType using parsePieceType parser, applying the function to the result of parsing within a monad. Hence, the use of the for all operator allowed to chain together operations. To run the program, stack run *filename* must be entered. ParseRow resplicates parsePiece 4 times, whereas parseBoard replicated parseRow 4 times alongside checking if the length of the board is 16 and returning the board. ParseStartingPos has the type String -> Either String Board, which means if the board successfully parsed (right constructor) , it would return the board otherwise (Left constructor) it would return an error message. ParseInteger is a parser for integers, which will read 1 or more digits (used some), therefore for it to be successful, there must be at least 1 digit in the input. In the main function, I get the name of the file that was entered in the terminal, using the getArgs and then use the readFile function to read its file contents. Then the startingPos is printed. Also, to check if the line numbers are sequential, follow the correct pattern(number followed by the dot) , I used isDigit, takeWhile, zip functions to define order and number. If the line numbers are correct, then the parsing occurs with the parseInteger method, otherwise relevant error messages are returned. Moreover, unnecessary characters from the input are removed such as line numbers, new line characters and spaces. Then it is translated to a coordinate or a draw or lose. A coordinate parser is also used. The hash symbol parser and draw parser return invalid locations of the board, indicating inability to make move. Then using the Either data type's left and right, either the parsed tuples are returned or error is returned if unsuccessful. It is then passed onto produceNewBoard function along with starting position. If the coordinates are (30,30) or (40,40), game is drawn or the relevant player wins message is printed. If there is a valid coordinate, makeMove function is used to make the move according to the parsed input , and the new board which is returned from makeMove is shown. Otherwise, invalid is returned.

I used megaparsec as the parsing library. I also used 'Haskell Tutorial - 12 - Writing Parsers From Scratch' by James Hobson[1], to get an insight of what parsers are, how they work and what they are supposed to do. [1] Using this video, I learnt more about functor, applicative, monad fail and monad with parsers type classes and instance declarations. Moreover, I used Mark Karpov's Megaparsec tutorial to understand how to consume white spaces with the library. [2]

This is not a complete implementation. Improvements need to be made. The board after makeMove is not being returned successfully. If I had more time, I would have been able to produce a better implementation than the one submitted.

[1] https://www.youtube.com/watch?v=LeoDCiu_GB0

[2] https://markkarpov.com/tutorial/megaparsec.html#white-space