



Universidade Federal de Santa Catarina

Departamento de Informática e Estatística (INE)

Curso: Ciência da Computação

Disciplina: Paradigmas de Programação (INE5416)

Professor: Maicon Rafael Zatelli

Estudante: Guilherme Adenilson de Jesus (22100620)

Relatório T1 - Haskell

• Problema

O presente relatório é referente ao desenvolvimento de um programa em Haskell para um solucionador do quebra-cabeça escolhido, Vergleichssudoku (Sudoku Greater Than). Esse *puzzle* é uma variante do Sudoku, em que cada posição do tabuleiro possui comparadores para as posições adjacentes a ela dentro do seu quadrado.

• Solução

O tabuleiro é separado em três matrizes. Uma para armazenar os valores de cada posição, uma para os comparadores horizontais e outra para os verticais. Essas estruturas utilizam o `Data.Vector` como auxílio, invés do padrão `Array` (`[Type]`). Os comparadores são obtidos por um arquivo `.txt`, detalhado melhor posteriormente.

As três matrizes são enviadas para função *solve*. Que executará cada alternativa possível. Ele pega a primeira opção possível para executar recursivamente a função *solve*. No momento que encontrar a solução (quando todas as posições do tabuleiro forem preenchidas), a função parará de chamar as próximas opções e retornará o resultado.

Nota-se no trecho de código abaixo (Figura 1) que há uma função auxiliar *findSolution*. Ela recebe um vetor com todas funções *solve* que foram avaliadas como possíveis (pela função *possible*). Graças ao *lazy evaluation* do Haskell, essas funções só serão executadas quando forem referenciadas, possibilitando a parada imediata quando encontrar uma solução.

```
solve :: Board -> Comparadores -> Comparadores -> Maybe Board
solve b compH compV
| full b = Just b
| otherwise = findSolution $ [ solve (insert b (x, y) n) compH compV |
    (x,y) <- take 1 empty,
    n <- [1..9],
    possible b compH compV (x,y) n]
  where empty = [(x,y) | x<-[0..8],y<-[0..8], isEmpty b (x, y)]
        findSolution [] = Nothing --Não há solução possível
        findSolution (Nothing:xs) = findSolution xs --Executa a próxima opção
        findSolution ((Just b):_) = Just b --Encontrou a solução, acabou a busca
```

Figura 1 - Função *solve*
Fonte: autoria própria

A seguir, há a função *possible*. Ela avaliará se a alternativa solicitada (k) será possível baseada na atual organização possível. Analisando o trecho de código abaixo (Figura 2), são feitas até 6 verificações: se a posição está fora do tabuleiro, se a posição já está preenchida, se k já é um elemento da linha, coluna e quadrado. Essas análises são iguais às realizadas no Sudoku, o que altera é a última, que utiliza os comparadores referentes à posição. Destaca-se que cada célula de um quadrado tem diferentes quantidades de comparadores, o que torna necessário a avaliação da posição em relação ao seu quadrado. Para realizar essa verificação final, é utilizada a função *comparing* para cada comparador.

```
possible :: Board -> Comparadores -> Comparadores -> Position -> Integer -> Bool
possible b compH compV coords@(i, j) k
    | i < 0 || i >= 9 || j < 0 || j >= 9 || k < 0 || k > 9 = undefined
    | b ! coords > 0 = False --posição já tem um valor
    | k `elem` getRow b coords = False --o número já está presente na linha
    | k `elem` getColumn b coords = False --o número já está presente na coluna
    | k `elem` getSquare b coords = False --o número já está presente no quadrado

    --As demais verificações são feitas para analisar cada uma das posições de um quadrado, visto
    --que cada posição tem uma quantidade diferente de comparações referentes a ela
    | i `mod` 3 == 0 && j `mod` 3 == 0 = comparing k (getComp compH (i, j)) (b!(i,j+1)) &&
      comparing k (getComp compV (i, j)) (b!(i+1,j))

    | i `mod` 3 == 0 && j `mod` 3 == 1 = comparing (b!(i,j-1)) (getComp compH (i, j-1)) k &&
      comparing k (getComp compH (i, j)) (b!(i,j+1)) &&
      comparing k (getComp compV (i, j)) (b!(i+1,j))

    | i `mod` 3 == 0 && j `mod` 3 == 2 = comparing (b!(i,j-1)) (getComp compH (i, j-1)) k &&
      comparing k (getComp compV (i, j)) (b!(i+1,j))

    | i `mod` 3 == 1 && j `mod` 3 == 0 = comparing k (getComp compH (i, j)) (b!(i,j+1)) &&
      comparing k (getComp compV (i, j)) (b!(i+1,j)) &&
      comparing (b!(i-1,j)) (getComp compV (i-1, j)) k

    | i `mod` 3 == 1 && j `mod` 3 == 1 = comparing k (getComp compH (i, j)) (b!(i,j+1)) &&
      comparing k (getComp compV (i, j)) (b!(i+1,j)) &&
      comparing (b!(i-1,j)) (getComp compV (i-1, j)) k
      comparing (b!(i,j-1)) (getComp compH (i, j-1)) k
```

Figura 2 - Função *possible*
Fonte: autoria própria

Como visto na Figura 3 abaixo, a função *comparing* recebe dois inteiros e uma string (comparador). Para acelerar as análises, é feito algumas avaliações das alternativas extremas (1 e 9). Se o valor 1 estiver ao lado de um comparador '>', é impossível que haja outro número menor que ele, então já é considerado False de imediato, mesma coisa para valor 9 com '<'. Essa verificação premeditada é importante devido ao valor 0 (posição vazia), que qualquer valor comparado a ele já se tornaria True, podendo considerar uma alternativa que nunca daria uma solução correta. Caso passe por todas essas avaliações, é feito a comparação em si entre a e b.

```
--Checa se a comparação é verdadeira
comparing :: Integer -> String -> Integer -> Bool
comparing 1 ">" _ = False --1 nunca será maior que qualquer outro número de 1 a 9
comparing _ "<" 1 = False
comparing 9 "<" _ = False --9 nunca será menor que qualquer outro número de 1 a 9
comparing _ ">" 9 = False
comparing 0 _ 0 = True --0 é um valor sem importância nesse contexto, então qualquer opção de 1 a 9 seria verdade
comparing 0 _ _ = True
comparing _ _ 0 = True
comparing a comparador b | comparador == "<" = a < b
                        | comparador == ">" = a > b
                        | otherwise = True
```

Figura 3 - Função *comparing*
Fonte: autoria própria

- **Entrada pelo usuário**

Como exposto na Figura 4 abaixo, o usuário enviará pelo terminal o arquivo de texto referente à configuração do quebra-cabeça. Caso não seja recebido, o programa optará pelo uso do arquivo padrão.

```
main = do
  args <- getArgs --Coleta os argumentos do terminal

  --Se não receber um arquivo pelo terminal, utiliza o arquivo padrão
  let textPath = if length args /= 1 then "defaultPuzzle.txt" else head args

  --Coleta o conteúdo do arquivo
  contentsText <- Text.readFile textPath
```

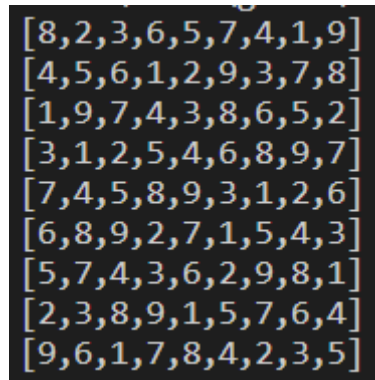
Figura 4 - Recebimento de entrada
Fonte: autoria própria

O arquivo de configuração é similar ao apresentado na Figura 5 a seguir. Começa pelos comparadores horizontais, linha a linha. Quando não houver um comparador, é utilizado o caractere "|", indicando troca de quadrado ou fim da linha do quebra-cabeça. Para facilitar a leitura e separação, antes da descrição dos comparadores verticais é colocado uma linha com um único caractere ".".

```
1  < > | > < | < < |
2  > < | < < | > < |
3  > < | < > | < > |
4  < < | > < | > < |
5  < > | > < | > > |
6  > > | > < | < > |
7  > < | < > | < < |
8  > < | > > | > > |
9  < > | < < | < < |
10 .
11 < > < < < > > >
12 < > > < > > < < >
13 | | | | | | | |
14 > > > < < < < >
15 < > < < < > > < >
16 | | | | | | | |
17 < < > < > > < > >
18 > < > > < < > < <
19 | | | | | | | |
```

Figura 5 - Configuração do quebra-cabeça
Fonte: autoria própria

O resultado do jogo é imprimido no terminal (Figura 6), mostrando os valores que estarão presentes em cada posição do quebra-cabeça. Caso não haja uma solução possível, será imprimido um *Nothing*.



```
[8,2,3,6,5,7,4,1,9]
[4,5,6,1,2,9,3,7,8]
[1,9,7,4,3,8,6,5,2]
[3,1,2,5,4,6,8,9,7]
[7,4,5,8,9,3,1,2,6]
[6,8,9,2,7,1,5,4,3]
[5,7,4,3,6,2,9,8,1]
[2,3,8,9,1,5,7,6,4]
[9,6,1,7,8,4,2,3,5]
```

Figura 5 - Resultado do quebra-cabeça
Fonte: autoria própria

- **Dificuldades encontradas**

Uma das dificuldades encontradas foi como fazer a leitura do quebra-cabeça a ser resolvido por um arquivo. Foi necessário o auxílio da biblioteca `System.Environment` para coletar o nome do arquivo pelo terminal. Além disso, foi preciso `Data.Text` e `Data.Text.IO` para ler o conteúdo do arquivo e separá-lo e convertê-lo propriamente para uma string para ser utilizada no restante do programa.

Outro problema foi a forma de representar os comparadores na matriz sem precisar de um extra-cálculo para determinar quais comparadores estão relacionados a cada posição. A solução foi adicionar um identificador de troca de quadrado no quebra-cabeça, além de separá-lo em horizontais e verticais. Cada posição na matriz de comparadores é a mesma posição da célula a esquerda ou acima do comparador referente.