Universidade Federal de Santa Catarina

Campus de Florianopolis

Relatório Vergleichssudoku

João Volpato

Thiago Bewiahn

Rafael Vargas

Florianópolis/SC

2022

Relatório Vergleichssudoku Prof. Maicon Zatelli

Resumo

O problema escolhido para ser resolvido neste trabalho foi o Vergleichssudoku. Ele é semelhante ao sudoku tradicional, com as mesmas restrições de não repetir os números em linhas, colunas e quadrados. Pórem a entrada do problema escolhido não contém números mas sim comparadores entre as células do tabuleiro. Com base nessas comparações (maior que e menor que), os números devem ser preenchidos no tabuleiro.

Sumário

1 Solução

1.1 Introdução

Para visualizar a solução, é necessário executar o arquivo main.hs, que imprimirá "No solution found"no caso de não haver solução ou "Found solution: "seguido das linhas da matriz-resposta, com cada célula separada por um espaço.

1.2 Estruturas

Como o problema possui dois elementos principais, o proprio tabuleiro do sudoku e a entrada de comparadores, decidimos criar duas estruturas separadas, uma para representar o tabuleiro e outra a entrada dos comparadores.
O tabuleiro é um Monad Maybe, que pode ser uma matriz de inteiros, ou Nothing. Utilizamos o Nothing para quando não achamos solução válida e precisamos reiniciar a célula para continuar o backtracking.
A entrada dos comparadores foi organizada numa matriz de strings, onde cada posição corresponde aos comparadores da célula correspondente na matriz de inteiros. A ordem dos comparadores em cada célula segue o sentido horário. Ex.: .<>. O primeiro caractere representa o comparador com o topo (O ponto representa a ausência de comparador), o segundo representa a comparação com a direita, terceiro o de baixo e quarto o da esquerda. Neste caso: (TOPO ., DIREITA <, BAIXO >, ESQUERDA .). As matrizes de comparadores estão hardcoded, representadas linha a linha como uma String e o | representa a divisão de cada célula. Os | são eliminados ao construir a matriz e a String é quebrada em Strings menores (1 String por célula).

```
row1 :: [Char]
row1 = ".<>.|.><>|..<<|.<>.|.<>>|..>>|.><.|.>><|..<<|"

takePipeOut :: [Char] -> [Char]
takeAllComparatorsFromRow :: [Char] -> [[Char]]
takeAllComparatorsFromRow [] = []
takeAllComparatorsFromRow rawRow = take 4 row : takeAllComparatorsFromRow
    (drop 4 row)
                                    where
                                      row = takePipeOut rawRow

getComparatorsGrid :: [[[Char]]]
```

```
getComparatorsGrid = map takeAllComparatorsFromRow allRowsSize9
```

1.3 Algoritmos

O principal algoritmo que resolve o problema do sudoku são as duas funções solveSudoku e solveSudokuWithValues

```
solveSudoku :: Maybe [[Int]] -> [[[Char]]] -> Int -> Int -> Maybe [[Int]]
solveSudoku sudokuGrid comparatorsGrid row column = do
  -- Verifica se chegou na ultima clula, retorna o tabuleiro
  if row == (sudokuSize - 1) && column == sudokuSize then trace "Found the
     solution: " sudokuGrid
  -- Verifica se chegou no final de uma linha, se sim passa para a prxima
  else if column == sudokuSize then solveSudoku sudokuGrid comparatorsGrid
     (row + 1) 0
  -- Verifica se o valor da clula atual j foi definido, se foi passa para a
     prxima clula
  else if getXY sudokuGrid row column > 0 then trace "Value already defined"
     solveSudoku sudokuGrid comparatorsGrid row (column + 1)
  else do
     -- Pega os possveis nmeros para a posio atual
     possibles <- Just (getPossibleOptions sudokuGrid comparatorsGrid row
        column)
     -- Aps validar a posio e adquirir os possveis nmeros chama o for
     solveSudokuWithValues sudokuGrid comparatorsGrid row column possibles 0
```

A função solve sudoku, recebe de entrada o Grid do sudoku, o Grid dos comparadores a linha e a coluna. O primeiro teste é o caso de fim do backtracking onde achamos uma solução valida, então a função retorna o gridsudoku pronto. Depois a função testa se a chegamos ao final da linha no tabuleiro e caso positivo passa para a proxima linha, caso contrario verificamos se o valor atual da célula já foi definido. Se ja está definido, passamos para a próxima celula, se não está chamamos a função getPossibleOptions, e passa para a função solveSudokuWithValues.

```
getPossibleOptions :: Maybe [[Int]] -> [[[Char]]] -> Int -> Int -> [Int]
getPossibleOptions sudokuGrid sudokuGridChars x y = [a | a <- [1..sudokuSize],
    notInRow a, notInCol a, notInSquare a, inCompareOptions a]
    where
        notInRow a = a `notElem` getRow sudokuGrid x y
        notInCol a = a `notElem` getCol sudokuGrid x y
```

```
    notInSquare a = a `notElem` getSquare sudokuGrid x y sudokuSize
    inCompareOptions a = a `elem` getCompare sudokuGrid sudokuGridChars x y
```

A função getPossibleOptions recebe o tabuleiro do sudoku, o grid de comparadores o x e y que indexam a célula no sudokuGrid. A função retorna uma lista de números possiveis que aquela celula pode assumir, seguindo as regras do jogo, então realizamos uma list comprehension e só colocamos na lista os elementos que atendem os filtros notInRow, notInCol, notInSquare e inCompareOptions.

O filto inCompareOptions é a regra específica desse tipo de sudoku, ele utiliza a função getCompare que efetua a comparação com base no sudokuGridChars

```
-- A partir de uma lista de possveis nmeros para uma  posio  especfica  testa
    cada um deles at terminar a lista ou algum funcionar
solveSudokuWithValues :: Maybe [[Int]] -> [[[Char]]] -> Int -> Int -> [Int] ->
    Int -> Maybe [[Int]]
solveSudokuWithValues sudokuGrid comparatorsGrid row column possibles index =
    do
  -- Verifica se o index ultrapassou o tamanho da lista, nesse caso no h soluo
  if index >= getListLength possibles then Nothing
  else do
    -- Seta a grid com o valor encontrado na lista de possibilidades no index
        recebido
    sudokuGrid <- setXY sudokuGrid row column (getValueInList possibles index)
    -- Continua o fluxo para a prxima clula e verifica o retorno
    case solveSudoku (Just sudokuGrid) comparatorsGrid row (column + 1) of
      -- Caso seja Nothing, no houve  soluo  ir  resetar o valor e tentar o
          prximo da lista de possibilidades
      Nothing -> do
        setXY (Just sudokuGrid) row column 0
        solveSudokuWithValues (Just sudokuGrid) comparatorsGrid row column
            possibles (index + 1)
      -- Caso seja um tabuleiro  pq houve  soluo ,  ento  retorna a  soluo
      Just n -> Just n
```

A função solveSudokuWithValues funciona como um for que percorre a lista de possible options que foi passada como parametro, testando cada um dos elementos da lista (iniciando uma ramificação da árvore do backtracking). Caso o retorno do solveSudoku daquele número testado seja Nothing, ou seja não há solução com aquela tentativa, ele seta aquela célula para 0 e testa a proxima possibilidade (chamando o solveSudokuWithValues denovo com o próximo elemento a ser testado da lista)

1.4 Entrada e Saída

1.4.1 Entrada

Os arquivos Dataset.hs e Config.hs são responsáveis por guardar a entrada do programa (comparatorsGrid) e pela variavél global que guarda o tamanho do sudoku a ser resolvido, respectivamente.

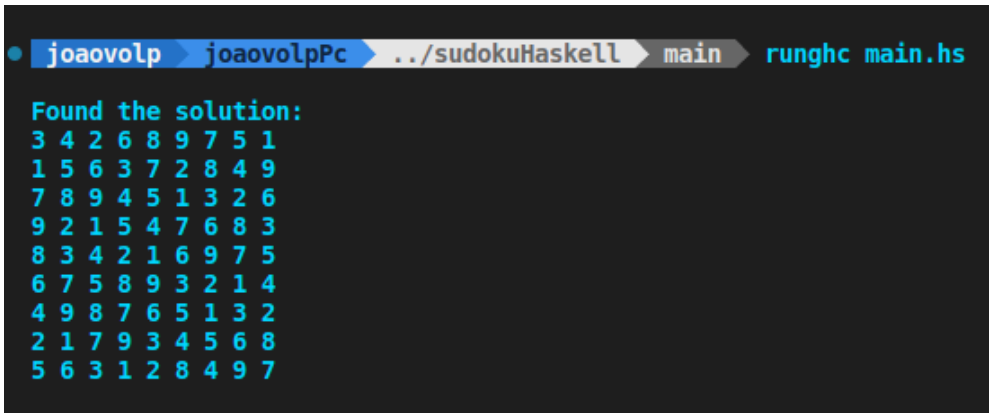
Exemplo de entrada:

```
row1 = ".<>.|.><>|..<<|.<>.|.<>>|..>>|.><.|.>><|..<<|"
row2 = "<<<.|><<>|>.<>|<<<.|<>>>|<.><|>>>.|<<><|>.>>|"
row3 = "><..|><.>|>..>|><..|<>.>|<..<|<>..|<<.<|<..>|"
row4 = ".>>.|.><<|..<<|.>>.|.<><|..>>|.<<.|.>>>|..<<|"
row5 = "<>>.|><<<|>.<>|<><.|<<<<|<.>>|>>>.|<>><|>.><|"
row6 = "<<..|>>.>|>..<|><..|>>.>|<..<|<>..|<<.<|<..>|"
row7 = ".<>.|.>>>|..><|.><.|.>><|..><|.<<.|.><>|..<<|"
row8 = "<><.|<<<<|<.>>|>>>.|<<><|<.<>|><>.|><<>|>.>>|"
row9 = "><..|>>.>|<..<|<<..|<<.>|>..>|<<..|>>.>|<..<|"
```

1.4.2 Saida

A saida do programa, o tabuleiro do sudoku resolvido caso tenha achado solução ou, No solution caso contrario, é feita por um arquivo Printer.hs por meio da função printMatrix

Exemplo de saida:



1.5 Organização

A comunicação do Grupo foi feita com "reuniões"no tempo de aula e em um grupo de whatsapp.

Primeiramente os tres membros do grupo discutiram a estratégia para iniciar o trabalho, ficou acordado que primeiro resolveriamos o problema em python e depois em haskell (sujestão do professor). Então montamos uma solução em python se basendo em um site da internet. Depois o Thiago implementou funções básicas em haskell que seriam úteis para a solução do problema, o grupo todo organizou como seria a entrada do programa e Thigo implementou a base do backtracking. Então o Rafael e o Thiago finalizaram o código corrigindo bugs. João ficou responsável pelo relatório.

1.6 Dificuldades

A principal dificuldade foi se adequar a uma linguagem com o paradigma funcional que era totalmente novo. Também por conta da linguagem ser fortemente tipada.

Uma dificuldade inicial foi a compreensão e visualização do backtracking, porém, após termos implementado a solução em python, ficou claro como funciona esse método.

Uma grande dificuldade foi fazer o debug, porque por conta da tipagem não fui possível utilizar a funções nativas com tipo de retorno IO. Para solucionar isso pesquisamos na internet e achamos a função trace da biblioteca debug que permite acrescentar prints mantendo a corretude do programa.

Houve também uma certa dificuldade em trabalhar com o Monad Mybe visto que a implementação inicial não utilizada Monads.