Trabalho I: Haskell

Arthur Barrichello, Gustavo Kundlatsch e Monique Bertan

Maio de 2019

1 O Problema

O problema proposto nesse trabalho é o Sudoku Vergleich, Sodoku Comparativo em uma tradução livre. Cada uma das células do sodoku original agora possuem de duas a quatro setas nas divisas entre as células, que representam qual comparação deve ser feita: maior que ou menor que. O problema pode ter dois tamanhos, 4x4 ou 6x6. No 4x4, o tabuleiro é separado em 4 blocos de 2 linhas e 2 colunas, e no 6x6 em 6 blocos de 3 linhas e 2 colunas. Além de repeitar a comparação apresentada, um número não pode se repetir numa mesma linha, coluna ou bloco.

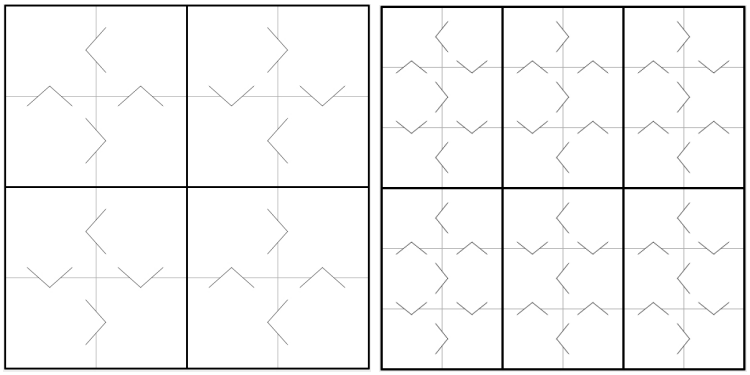

Figure 1: Sodokus comparativos em ambos os possíveis tamanhos.

2 Solução escolhida

Para resolver o problema apresentado, nosso grupo buscou um programa para resolver um sodoku tradicional para usar como base. Essa solução inicial está disponível *clicando aqui*. A ideia dessa solução é usar backtracking através de matrizes de arrays de possibilidades. Cada célula do sodoku possui ou um array com números que ainda podem ser usados, ou um número fixo. A cada iteração,

Caso	Possibilidades
<<	[1, 2]
<>	[2, 3]
>>	[3, 4]

Caso	Possibilidades
>>	[1 .. 4]
>>>	[1 .. 3]
<<	[3 .. 4]
<<<	[4 .. 6]
><<	[3 .. 5]
>><	[2 .. 4]
><	[2 .. 5]

Table 1: Tabelas de possibilidades.

o resolvedor fixa valores comparando os valores com a linha, coluna e bloco em que está. Quando todas as linhas tiverem sido percorridas, a matriz é invertida e o processo é repetido.

Para resolver o sodoku comparativo, resolvemos reduzir o problema das comparações para uma matriz de possibilidades e passar ela como entrada do algoritmo de resolução que já estava pronto. A lógica descrita leva em conta quantos "menores que" e "maiores que" uma célula possui, para tirar conclusões de quais possíveis valores podem ser aplicados.

2.1 Entrada de Dados

A entrada de dados do programa é feito através da função `signMatrix` da ordem correspondente (4x4 ou 6x6). Os elementos da matriz são quádruplas de inteiros, que representam cada uma das "paredes" da célula da matriz. O primeiro elemento da tupla é a parede da esqueda, e os elementos subsequentes são as paredes seguintes seguindo o sentido horário. Os valores 1 representam o sinal $<$ e os valores 2 representam o sinal $>$. Essas matrizes são classes criadas para o trabalho, que junto com a entrada de dados se encontra no módulo *Matrices.hs*. A implementação da Matriz é feita utilizando lista de listas de Ints. Abaixo, segue o código de como seria representado o exemplo 3 do site do Sodoku Comparativo dentro da função `signMatrix4x4`:

```
Mat [(0, 0, 2, 2), (1, 0, 0, 1), (0, 0, 2, 2), (1, 0, 0, 1),
(0, 1, 2, 0), (1, 2, 0, 0), (0, 1, 1, 0), (2, 2, 0, 0),
(0, 0, 1, 1), (2, 0, 0, 2), (0, 0, 1, 1), (2, 0, 0, 2),
(0, 2, 1, 0), (2, 1, 0, 0), (0, 2, 2, 0), (1, 1, 0, 0)] 4 4
```

2.2 Processamento de Dados

A entrada de dados não pode ser passada diretamente para o resolvedor de Sodoku escolhida, ela precisa ser processada na forma de matriz de possibilidades como descrito anteriormente. Para isso, criamos a função `sumSign` que retorna uma Matriz de inteiros, onde cada inteiro é a soma da tupla da

posição original. No Main, essa função é chamada como parâmetro da função `intMatrixToString`, e finalmente é utilizada como entrada de dados para o resolvedor de Sodoku.

A soma das tuplas é realizada para reproduzirmos as possibilidades mostradas na Tabela 1. Assim, quando tempos os símbolos >>, por exemplo, a tupla correspondente pode ser (0, 0, 2, 2), a soma será 4 que é passado para o resolvedor que foi alterado para entender que uma entrada '4' representa as possibilidades [3, 4]. Essa conversão de um inteiro para uma lista de possibilidades é feita na função `readGrid` no módulo *Reference* corresponde a ordem da matriz que está sendo trabalhada.

3 Problemas Encontrados

Para facilitar a implementação, separamos dois resolvedores, um para o puzzle de tamanho 4x4 e outro para o puzzle de tamanho 6x6. O principal problema da solução é que não temos, a princípio, como diferenciar a combinação de sinais '<>' de '><', e quando o valor 3, padronizado para ambos os casos como descrito na seção de processamento de dados, é passado para o resolvedor adotado, os casos são tratados como iguais pelo resolvedor e pode haver uma solução logicamente errada. O erro ocorre apenas entre os comparadores, sempre dois a dois. Não acontece o caso de um número se repetir em uma mesma coluna, linha ou bloco, pois isso é tratado no sodoku normal. Para diferenciar esses dois casos, criamos regras mais específicas para termos 4 valores diferentes sendo passados para o resolvedor, e criamos as listas de maneira diferente: [3, 4] e [4, 3]. Assim a solução funcionou para os casos em que foi testada.

No puzzle 6x6 o problema se torna maior, pois temos mais possibilidades diferentes. Apesar disso, conseguimos criar uma função que mapeia cada um dos casos para uma lista de possibilidades, assim como no 4x4. Entretanto, o resolvedor escolhido não foi desenvolvido para suportar matrizes com blocos não quadrados (3x2, nesse caso). Esse foi a única função que não consguimos adaptar adequadamente para cumprir o necessário.