Relatório de implementação de resolvedor de puzzle Kojun

Gabriel de Vargas Coelho, 20200400

Florianópolis, 2022

Análise do Problema

O jogo escolhido para esse trabalho foi o kojun. A escolha se deu simplesmente por gosto pessoal, uma vez que não gosto de jogos no estilo sudoku, o kojun foi, dentre as opções, o mais divertido.

O kojun se trata de um jogo simples, em que o objetivo é conseguir completar um tabuleiro com números, respeitando 3 regras:

1. Casas adjacentes ortogonalmente devem possuir números distintos;
2. O número máximo para uma casa é determinado pelo número de casas do grupo, e dentro de um grupo não pode haver repetição de números;
3. Casas de um mesmo grupo e de mesma coluna, devem apresentar numeração decrescente no sentido de cima para baixo.

Assim, o desafio para a implementação de uma solução ao kojun se dá pelo fato de haverem muitas "respostas" corretas para um tabuleiro. Uma vez traçada a estratégia para o preenchimento das casas do tabuleiro, o segundo desafio é a verificação da corretude da solução. Na próxima seção são explicitados ambas as estratégias.

Solução Proposta

A primeira coisa a ser destacada aqui é a modelagem do tabuleiro. Optou-se por utilizar a estrutura de chave e valor para modelar uma casa do tabuleiro, da seguinte forma: Int-Int, em que o primeiro valor é o indicador do grupo ao qual aquela casa pertence, e o segundo valor é destinado para o valor daquela casa. Uma vez definida a modelagem de uma casa do tabuleiro, o tabuleiro em si é somente uma matriz, em que cada posição é do tipo Int-Int. Para sinalizar uma casa vazia é usado uma variável anônima _.

```
board(6, [[0-_, 1-_, 1-4, 1-_, 3-2,  4-_],
          [0-_, 2-_, 1-3, 3-_, 3-_,  3-_],
          [0-1, 0-4, 5-_, 3-4, 10-_, 10-_],
          [6-_, 7-5, 5-_, 9-_, 9-_,  10-2],
          [6-_, 7-_, 7-_, 8-_, 8-3,  10-_],
          [7-6, 7-2, 7-_, 8-2, 8-_,  8-5]]).
```

Imagem 1: Tabuleiro 6x6 usando a modelagem proposta

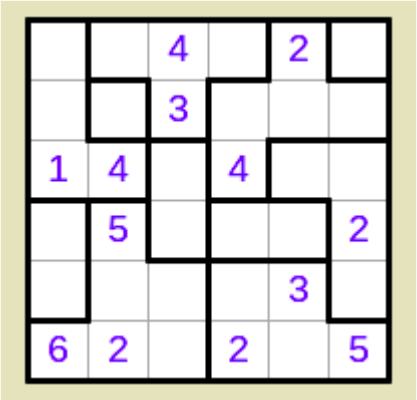

Imagem 2: Tabuleiro modelado pela imagem 1. Fonte: [1]

Para a solução do tabuleiro foi utilizado a programação de restrições, em que são definidas as regras do jogo e as restrições para o valor de uma casa do tabuleiro, e a própria linguagem em tempo de execução é responsável por achar os valores que respeitem as regras e restrições. Uma vantagem dessa estratégia é a simplicidade do algoritmo, há apenas declarações que devem ser satisfeitas e isso facilita a leitura e entendimento do código, por outro lado há a desvantagem de se conhecer os operadores e ser minucioso no momento da declaração da restrição, pois geralmente erros não serão acusados. Um exemplo do uso da programação de restrições se encontra nas imagens abaixo:

```
% Verifica se os grupos não possuem valores repetidos
flatten(R,Rflat),
keysort(Rflat, RflatSorted),
group_pairs_by_key(RflatSorted, RflatGrouped),
pairs_values(RflatGrouped, RValues),
maplist(all_distinct, RValues),

% Verifica se vizinhos não possuem valores repetidos
maplist(validNeighbours, R),
transpose(R, Columns),
maplist(validNeighbours, Columns),

% Verifica se células na mesma coluna e de mesmo grupo tem valor decrescente
maplist(decreaseByGroup, Columns).
```

Imagem 3: Trecho de código que exemplifica a programação de restrições.

```
% Verifica na lista se itens adjacentes de mesma chave (mesmo grupo) estão em ordem decrescente
decreaseByGroup([_]).
decreaseByGroup([H1,H2|T]) :-
  pairs_keys([H1],K),
  pairs_keys([H2],K),
  pairs_values([H1],[V1]),
  pairs_values([H2],[V2]),
  V1 #> V2,
  decreaseByGroup([H2|T]).
decreaseByGroup([H1,H2|T]) :-
  pairs_keys([H1],[K1]),
  pairs_keys([H2],[K2]),
  K1 \== K2,
  decreaseByGroup([H2|T]).
```

Imagem 4: Programação de restrições para verificação de grupos com valores decrescentes por coluna.

Na Imagem 3, as regras gerais do jogo são descritas, por exemplo: um grupo não deve ter valores repetidos, que é verificado pelo predicado all_distinct. Já a imagem 4 ilustra a regra de que um grupo deve obedecer a regra de valores decrescentes caso os valores se encontrem na mesma coluna.

Para obter a resposta de um tabuleiro, o usuário deve fazer a seguinte consulta: kojun(x), sendo x o número de um tabuleiro, que deve ser definido pelo predicado board, como ilustra a Imagem 1. A Imagem abaixo mostra a saída do programa para o tabuleiro da Imagem 1.

```
?- kojun(6).
3, 2, 4, 1, 2, 1
2, 1, 3, 5, 1, 3
1, 4, 2, 4, 3, 4
2, 5, 1, 2, 1, 2
1, 4, 3, 4, 3, 1
6, 2, 1, 2, 1, 5
true .
```

Imagem 5: Solução possível do tabuleiro das Imagens 1 e 2.

Paradigma lógico x Paradigma funcional

Primeiramente vale destacar que o paradigma lógico torna a tarefa de solucionar um puzzle, através de um programa, muito simples e intuitiva. A programação por restrições permite que a maior parte do trabalho seja feito pela própria linguagem, assim é necessário apenas escrever as regras do jogo de modo adequado.

Por outro lado, a programação funcional pode facilitar durante o desenvolvimento por apresentar mensagens de erro mais claras. Erros de digitação e até mesmo de lógica podem

ocorrer e são melhor indicados pelos compiladores e interpretadores de linguagens funcionais, assim é mais fácil fazer a correção de problemas nesse paradigma.

Referências

[1] JANKO.AT. **Kojun**. Disponível em: <https://www.janko.at/Raetsel/Kojun/index.htm>. Acesso em: 08 out. 2022.