UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
CIÊNCIA DA COMPUTAÇÃO

Artur Luiz Rizzato Toru Soda

Relatório - Trabalho 2 - Programação Lógica - Prolog
Kojum

Florianópolis
2024

1. INTRODUÇÃO

Este trabalho tem como objetivo aplicar os conhecimentos aprendidos na disciplina Paradigmas de Programação, criando um resolvedor para uns dos mesmos três puzzles mostrado no enunciado dos trabalhos anteriores, entretanto, dessa vez deveria ser implementado na linguagem Prolog.

Dessa forma, o problema escolhido neste trabalho foi o mesmo dos outros trabalhos, Kojum, cujo o funcionamento é da seguinte forma: Dado um tabuleiro dividido em diversas regiões, um número deve-se ser inserido em cada campo, de modo que: Cada região de tamanho N contenha números de 1 à N exatamente uma vez; Campo adjacentes ortogonalmente devem ter valores diferentes; E por fim, se dois campo são verticalmente adjacentes na mesma região, o valor de cima deve ser maior que o de baixo. O puzzle é resolvido quando o tabuleiro é completo, respeitando as três regras supracitadas.

2. DESENVOLVIMENTO

2.1. Programação de Restrições

A programação de restrição é uma técnica de resolução de problemas que envolve a definição de restrições ou condições que as soluções devem satisfazer. Assim, em vez de especificar um procedimento passa a passo para encontrar a solução, o problema é descrito em termos de variáveis e restrições, é um algoritmo de busca ou de propagação de restrições é usado para encontrar os valores que satisfazem todas as restrições.

Exemplos de como isso ocorre será mostrado mais à frente neste relatório, no tópico onde é mostrado a solução do problema.

2.2. Programação de Restrições: Vantagens e Desvantagens

Durante a resolução do trabalho, foi possível perceber algumas vantagens da programação de restrição para resolver puzzles, principalmente sobre a facilidade de implementação, pois permite descrever o problema de forma declarativa, especificando diretamente as condições e restrições que a solução deve satisfazer. Ademais, é bastante flexível, visto que é fácil adicionar, remover ou modificar restrições sem precisar redesenhar o algoritmo do zero.

Entretanto, foi observado, também, algumas desvantagens, em especial a programação funcional, paradigma utilizado nos trabalhos anteriores, pois a programação de restrições é mais adequada e especializada para esses tipos de problemas por restrições, enquanto a funcional é mais geral, podendo ser aplicada a uma maior variedade de problemas. Dessa forma a programação de restrição, em geral, possui uma menor performance para problemas genéricos, por possuir uma menor gama de problemas que se adequam ao seu paradigma.

2.3. Prolog - Solução

No começo de sua execução, o programa recebe como argumentos qual tabuleiro deve

ser resolvido, carregando então a matriz de valores e de regiões.

```
4   % Le arquivos e monta matrizes de valores e regioes
5   build_matriz(Path, ValuesMatriz, RegionsMatriz) :-
6       atom_concat('Entradas/', Path, P1),
7
8       atom_concat(P1, '/numbers.txt', ValuesPath), atom_concat(P1, '/regions.txt', RegionsPath),
9       open(ValuesPath, read, ValuesStr),           open(RegionsPath, read, RegionsStr),
10      read_matriz(ValuesStr, ValuesFile),          read_matriz(RegionsStr, RegionsFile),
11      close(ValuesStr),                            close(RegionsStr),
12      ValuesFile = [ValuesMatriz|_],               RegionsFile = [RegionsMatriz|_].
13
14
15  read_matriz(Stream, []) :- at_end_of_stream(Stream).
16  read_matriz(Stream, [H|T]) :-
17      read(Stream, H),
18      read_matriz(Stream, T).
19
20  main(Argv) :-
21      Argv = [Path|_], string_upper(Path, PathUpper),       % Recebe Qual tabuleiro resolver como
22      build_matriz(PathUpper, ValuesMatriz, RegionsMatriz), % Monta matriz de valores e regioes
23      solve_board(ValuesMatriz, RegionsMatriz),             % Resolve tabuleiro
24      halt.
```

Logo em seguida, a função *solve_board* é chamada, que tendo como parâmetro as matrizes de valores e regiões, define as restrições e imprime o tabuleiro antes de depois de ser resolvido.

```
5   solve_board(ValuesMatriz, RegionsMatriz) :-
6
7       print_matriz(ValuesMatriz, RegionsMatriz, false), nl,  % Imprime
8
9       ht_new(RegionHT), get_regions(RegionsMatriz, Regions), % Cria no
10      make_regionsHT(ValuesMatriz, RegionsMatriz, RegionHT), % Monta h
11
12      % Restricoes das:
13      %   - Regioes
14      regions_constraint(Regions, RegionHT),
15      %   - Linhas
16      rows_constraint(ValuesMatriz),
17      %   - Colunas
18      transpose(ValuesMatriz, TrVM),  transpose(RegionsMatriz, TrRM),
19      columns_constraint(TrVM, TrRM),
20
21      % Atribui valores a matriz
22      maplist(label, ValuesMatriz),
23
24      % Imprime resultado sem cor
25      print_matriz(ValuesMatriz, RegionsMatriz, false), nl,
26
27      % Imprime resultado com cor
28      print_matriz(ValuesMatriz, RegionsMatriz, true).
```

Dessa forma, correspondendo às regras do puzzle, as restrições foram divididas em 3 funções:

A primeira chamada é *rows_constraint(ValuesMatriz)*, que recebendo como parâmetros a matriz de valores delimita que valores vizinhos horizontalmente, devem ter valores diferentes.

```
57      % rows_constraint(ValuesMatriz)
58      %   Valida linhas
59      rows_constraint([]).
60      rows_constraint([Row|T]) :-
61          single_row_constraint(Row),
62          rows_constraint(T).
63
64      % single_row_constraint(ValueRow)
65      %   Metodo auxilidar de rows_constraint
66      %   Valores vizinhos devem ser diferentes
67      single_row_constraint([]).
68      single_row_constraint([_]).
69      single_row_constraint([V1, V2|T]) :-
70          V1 #\= V2,
71          single_row_constraint([V2|T]).
```

Em seguida vem a função *columns_constraint(TrVM, TrRM)*, tendo como parametros a matriz de valores e de regiões transpostas (para que as colunas fossem acessíveis mais facilmente), restringe que valores vizinhos verticalmente, se são da mesma região, então o valor de cima deve ser maior, caso contrário, apenas devem ser diferentes.

```
42      % columns_constraint(ValuesColumns, RegioesColumn)
43      %   Restricoes das colunas
44      columns_constraint([], []).
45      columns_constraint([VColumn|VT], [RColumn|RT]) :-
46          single_columns_constraint(VColumn, RColumn),
47          columns_constraint(VT, RT).
48
49      % single_columns_constraint(Column, Region)
50      %   Metodo auxiliar de columns_constraint (logica)
51      single_columns_constraint([], []).
52      single_columns_constraint([_], [_]).
53      single_columns_constraint([C1, C2|CT], [R1, R2|RT]) :-
54          (R1 == R2 -> C1 #> C2 ; C1 #\= C2),
55          single_columns_constraint([C2|CT], [R2|RT]).
```

Por fim, a última restrição chamada é *regions_constraint(Regions, RegionsHT)*, que

recebe a lista com as regiões existentes no tabuleiro, e a hash-table das regiões, onde as chaves são as regiões, e os valores são listas com os valores da região correspondente. Assim, a função define que que os valores de cada área variam de 1 até o tamanho da região, e todos são distintos.

```
30      % regions_constraint(RegionList, RegionHT)
31      %   Restricoes dos valores da regiao
32      regions_constraint([], _) :- !.
33      regions_constraint([R|T], RegionHT) :-
34          ht_get(RegionHT, R, RegionList),
35          length(RegionList, RegionLen),
36                                          % Valores
37          RegionList ins 1..RegionLen, %      - Va
38          all_different(RegionList),   %      - To
39
40          regions_constraint(T, RegionHT).
```

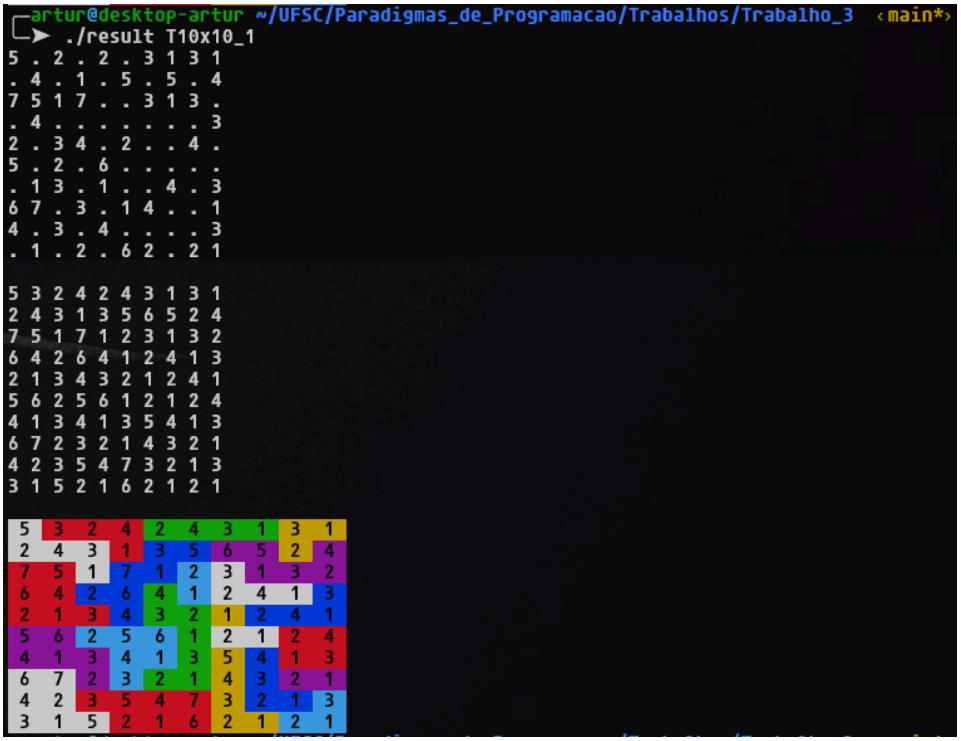
3. EXECUÇÃO

Como já dito anteriormente, o problema a ser resolvido é passado como argumento do executável criado na compilação, que é o nome do diretório que possui dois arquivos textos, um com os valores de cada campo, e outro indicando as regiões (indicadas por caracteres).

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

```
1     [[a, b, b, b, c, c, c, c, d, d],
2      [a, a, a, b, e, e, f, f, d, f],
3      [i, i, a, e, e, g, h, f, f, f],
4      [i, i, e, e, j, g, h, h, h, l],
5      [i, i, i, e, j, j, k, l, l, l],
6      [m, m, n, n, n, j, o, o, p, p],
7      [m, m, m, n, n, q, r, s, p, p],
8      [v, v, m, n, q, q, r, s, t, t],
9      [v, v, w, w, w, w, r, s, s, u],
10     [v, v, v, w, w, w, r, r, u, u]].
```

Por fim, com o tabuleiro resolvido é feita a impressão do estado inicial do quadro, depois uma impressão simplificada de cada campo, e por fim, uma impressão da solução com suas regiões separadas.



4. CONCLUSÃO E DIFICULDADES

Considerando os tópicos supracitados, é possível concluir que o projeto proposto, foi uma ótima prática, para o um melhor conhecimento da linguagem de restrições Prolog, nos permitindo pesquisar e ir atrás de diferentes soluções para o problema escolhido, e então analisarmos qual o melhor para o devido desafio, como por exemplo, a implementação do problema em paradigma de restrição, que no meu caso, foi onde mais encontrei dificuldade.

Dessa forma, tive dificuldade de implementar nesse paradigma, pois mesmo já tendo feito anteriormente esse puzzle, Prolog possui um paradigma bem diferente no quesito de implementação, algo bem fora da minha zona de conforto. Assim, foi preciso entender e se acostumar como as restrições deveriam ser feitas, para a resolução.