Universidade Federal de Santa Catarina - UFSC
Departamento de Informatica e Estatistica (INE)
Disciplina: INE5416 - Paradigmas de Programação
Aluno: Lucas Broering dos Santos (22100909)

Relatório Kojun T3

Problema:

O Kojun é um puzzle lógico onde o tabuleiro é dividido em regiões, cada região precisa ter suas casas preenchidas do 1 ao tamanho da região. Ex: Região de tamanho 5, em suas cédulas terão valores de 1 … 5. Seria uma variação muito distante do sudoku. Ele possui algumas regras básicas, como:

- Não pode haver casas com valores repetidos adjacentes.
- Dentro da região não pode haver números repetidos na linha.
- Os valores de uma coluna na mesma região precisam estar em ordem decrescente.

Estratégias de Solução:

Desenvolvi a solução utilizando a linguagem Prolog, como especificada na descrição do problema.

```
/* Define o domínio dos valores que cada célula pode assumir com base no tamanho da região. */
valor_maximo_regiao([R,X]) :- tamanho_regiao(R,T), X in 1..T.

/* Garante que os vizinhos à direita sejam distintos */
vizinhos_diferentes([_]).
vizinhos_diferentes([[_,X1],[R2,X2]|T]) :-
    X1 #\= X2, append([[R2,X2]],T,L), vizinhos_diferentes(L).

/* Garante que o valor acima é maior quando fazem parte do mesmo grupo */
superior_maior([_]).
superior_maior([[R1,X1],[R2,X2]|T]) :-
    (R1 #\= R2 ; X1 #> X2), append([[R2,X2]],T,L), superior_maior(L).

/* Agrupa os elementos que pertencem a mesma regiao */
agrupar_elementos(_, [], []).
agrupar_elementos(R, [[R1, X1] | T], [X1 | L]) :- R #= R1, agrupar_elementos(R, T, L).
agrupar_elementos(R, [[R1, _] | T], L) :- R #\= R1, agrupar_elementos(R, T, L).

/* Verifica se todos os elementos de uma regiao são distintos */
todas_regioes_diferentes([H]) :-
    all_distinct(H).
todas_regioes_diferentes([H|T]) :-
    all_distinct(H),
    todas_regioes_diferentes(T).
```

Funções que verificam e garantem todas as regras do kojun. A **primeira função(valor_máximo_região)** vai receber uma região, com seu identificador, um inteiro e seu tamanho e restringe que o valor X tem que está entre 1 e o tamanho da região.

Na **vizinhos_diferentes**, existe um caso base onde a lista só tem um elemento, não possui vizinhos. No caso recursivo, pega os dois primeiros valores da lista, caso sejam diferentes junta o segundo elemento, vai fazendo isso recursivamente até que haja um elemento.

Na **superior_maior**, tem o caso base onde a lista só tem um elemento, e não precisa comparar. No caso recursivo, Toma os dois primeiros elementos da lista e verifica a condição (R1 #\= R2 ; X1 #> X2)

- Se R1 e R2 são diferentes, não há restrição de ordem entre X1 e X2.
- Se R1 e R2 são iguais, X1 deve ser maior que X2.

Junta o segundo elemento e o restante da lista e chama a função recursivamente.

A função **agrupar_elementos,** o caso base quando a lista de entrada está vazia, a função termina e a lista de saída também é vazia. No primeiro caso recursivo, se a região R é igual à região R1 do primeiro elemento da lista, o valor X1 é incluído na lista de saída, e chama a recursão novamente. Segundo caso recursivo, se a região R é diferente de R1, o valor não é incluído na lista de saída e chama recursão.

Na última função(**todas_regioes_diferentes**) existem dois casos, quando a lista de entrada contém apenas uma sublista H, all_distinct(H) garante que todos os elementos dessa sublista são distintos. E por último, quando verifica se todos os elementos da primeira sublista H são distintos usando all_distinct(H) e chama a recursão.

```
/* Lógica principal */
resolver(Puzzle) :-
    /* Concatena todas as sublistas de Puzzle em uma única lista Lista */
    append(Puzzle, Lista),
    /* Restringe os valores possíveis de cada célula com base no tamanho da região */
    maplist(valor_maximo_regiao, Lista),
    /* Garante que os valores das células vizinhas são diferentes, com as linhas */
    maplist(vizinhos_diferentes, Puzzle),
    /* Transpõe uma matriz (Puzzle) para obter suas colunas (Colunas) */
    transpose(Puzzle, Colunas),
    /* Garante que os valores das células vizinhas são diferentes, agora com as colunas */
    maplist(vizinhos_diferentes, Colunas),
    /* Garante que o valor de uma célula é maior que o valor da célula acima quando ambas pertencem à mesma região */
    maplist(superior_maior, Colunas),
    /* Agrupa os elementos de cada região */
    agrupar_elementos(0, Lista, Regiao0),
    agrupar_elementos(1, Lista, Regiao1),
    agrupar_elementos(2, Lista, Regiao2),
    agrupar_elementos(3, Lista, Regiao3),
    agrupar_elementos(4, Lista, Regiao4),
    agrupar_elementos(5, Lista, Regiao5),
    agrupar_elementos(6, Lista, Regiao6),
    agrupar_elementos(7, Lista, Regiao7),
    agrupar_elementos(8, Lista, Regiao8),
    agrupar_elementos(9, Lista, Regiao9),
    agrupar_elementos(10, Lista, Regiao10),
    Regioes = [Regiao0, Regiao1, Regiao2, Regiao3, Regiao4, Regiao5,
               Regiao6, Regiao7, Regiao8, Regiao9, Regiao10],
    todas_regioes_diferentes(Regioes), !.
```

A função **resolver**, vai fazer um append de todas as sublistas do tabuleiro em apenas uma, e depois vai aplicar todas as restrições do Kojun, chamando as funções exemplificadas acima, utilizando o maplist para aplicar a cada linha/coluna da matriz. Pode-se notar também a utilização do transpose para obter as colunas da matriz.

```
/* Chama a função para solucionar o tabuleiro */
solucao(ResultadoTabuleiro) :-
    tabuleiro(ProblemaTabuleiro),
    resolver(ProblemaTabuleiro),
    extrair_segundos_valores(ProblemaTabuleiro, ResultadoTabuleiro),
    imprimir_matriz(ResultadoTabuleiro).
```

Essa função vai chamar as demais funções para fazer a solução do tabuleiro, as duas funções chamadas no final servem para a impressão dos resultados.

```
/* Adquire a lista de cada linha da matriz */
extrair_segundos_valores([], []).
extrair_segundos_valores([Sublista | Resto], [SegundosValores | Resultado]) :-
    extrair_segundo(Sublista, SegundosValores),
    extrair_segundos_valores(Resto, Resultado).

/* Adquire o segundo elemento de cada lista adquirida */
extrair_segundo([], []).
extrair_segundo([[_, Segundo] | Resto], [Segundo | SegundosValores]) :-
    extrair_segundo(Resto, SegundosValores).

/* Imprime a matriz de forma legível */
imprimir_matriz([]).
imprimir_matriz([Linha|Resto]) :-
    writeln(Linha),
    imprimir_matriz(Resto).
```

Funções que servem para melhor visualização do resultado, a primeira transforma uma matriz de pares (região, valor) em uma matriz de valores e a segundo faz o mesmo mas com uma lista de pares (região, valor). Por último a função imprimir matriz, imprime cada linha na forma de uma matriz.

```
/* Esta definição representa o tabuleiro como uma lista de listas, onde cada célula é uma sub-lista com dois elementos:
a região e o valor (ou variável se o valor não estiver definido). */
/* https://www.janko.at/Raetsel/Kojun/003.a.htm */

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

/* Define a quantidade de elementos em cada região, sendo o primeiro valor a região e o segundo a quantidade de elementos.*/
tamanho_regiao(0,2).
tamanho_regiao(1,4).
tamanho_regiao(2,2).
tamanho_regiao(3,2).
tamanho_regiao(4,5).
tamanho_regiao(5,4).
tamanho_regiao(6,4).
tamanho_regiao(7,2).
tamanho_regiao(8,4).
tamanho_regiao(9,5).
tamanho_regiao(10,2).
```

Por último aqui está a definição do tabuleiro, onde o primeiro valor é a região e o segundo o valor que já está no tabuleiro inicial. Embaixo está a definição do tamanho de cada região do jogo.

Conclusão

A maior barreira para mim foi se adaptar ao paradigma de programação declarativo da linguagem Prolog. Porém a linguagem se mostrou muito eficaz e clara para resolver problemas lógicos, Tive dificuldade em entender a manipulação das listas, mas as aulas e exercícios foram muito importantes para esse quesito. Gostei muito de trabalhar com a linguagem Prolog, apesar de não ser muito utilizada, espero ter mais experiência com ela no futuro.