UNIVERSIDADE FEDERAL DE SANTA CATARINA – UFSC

CENTRO TECNOLÓGICO

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

CURSO DE GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

RELATÓRIO - TRABALHO 3 – PARADIGMAS DE PROGRAMAÇÃO

EDUARDO DIAS GUTTERRES

ARTHUR MOREIRA RODRIGUES ALVES

BRYAN MARTINS LIMA

FLORIANÓPOLIS, 02 DE DEZEMBRO DE 2019

O programa desenvolvido apresenta uma solução para o jogo Wolkenkratzer, ou em inglês, Skyscraper. O jogo consiste em um tabuleiro quadrado dividido em posições, linhas e colunas. Em um tabuleiro 5x5, é possível que os números de 1 a 5 apareçam nas linhas e colunas, representando as alturas das “torres”, não havendo repetições e sendo possível haver espaços em branco. Em volta do tabuleiro, ao lado de algumas das posições, pode haver um número, representando quantas “torres” é possível enxergar estando naquele lugar e olhando por aquela linha ou coluna. Por exemplo, se for possível visualizar 2 torres, uma solução possível para tal linha ou coluna é uma torre de altura 4 logo à frente e em seguida uma torre de altura 5, com as outras em qualquer ordem logo atrás, que não serão vistas pois são mais baixas que as duas primeiras.

O programa consiste em: dados os números existentes em volta do tabuleiro, apresentar uma solução para ganhar o jogo. A solução se dá por meio de um arquivo na linguagem Prolog, este contendo uma regra para soluções 5x5 e uma para 6x6.

A entrada do programa se dá executando o arquivo através do swipl, onde a pessoa invoca a regra problem (passando um 1 como primeiro parâmetro caso queira resolver um 5x5 ou um 2 caso seja 6x6) seguido da regra sudoku_5x5 ou sudoku_6x6. Nestas regras ela informa os números em volta do tabuleiro através de quatro listas: Uma para os números parte de cima, na parte de baixo, ao lado esquerdo e ao lado direito. A saída do programa, sendo a primeira variável da regra, é uma representação em forma de matriz do tabuleiro resolvido, apresentada no console do sistema operacional.

Para o desenvolvimento do programa, foi utilizado e modificado um programa de resolução do jogo Sudoku na linguagem Prolog, que funciona de forma semelhante ao Wolkenkratzer, mas com algumas diferenças.

O programa funciona da seguinte maneira: originalmente, o programa resolveria o Sudoku através de uma única regra, que utiliza algumas regras do Prolog, como length, maplist, append e transpose. Esta regra então receberia o tabuleiro parcialmente preenchido e resolveria o Sudoku. Para adaptar ao Wolkenkratzer, foi passado sempre um tabuleiro vazio através da regra problem. Adicionalmente,

foi retirada a parte onde se fazia a verificação das sub-áreas e adicionadas algumas regras para a verificação as alturas das torres.

A primeira delas é a regra visibility:

```
visibility(_, [], 0) :- !.
visibility(Pivot, [H|T], N) :-
    (
        H > Pivot -> visibility(H, T, N_), N is (N_ + 1);
        visibility(Pivot, T, N)
    ).
```

Esta regra realiza a contagem de quantas torres se é possível visualizar através de uma linha ou coluna, retornando este valor na variável N.

Depois, foram ciradas as regras check_visibility e check_visibility_reversed:

```
check_visibility([], []) :- true, !.
check_visibility([A|B], [H|T]) :-
    A =:= 0, check_visibility(B, T);
    visibility(0, H, N), N =:= A, check_visibility(B, T);
    false.


check_visibility_reversed([], []) :- true, !.
check_visibility_reversed([A|B], [H|T]) :-
    A =:= 0, check_visibility_reversed(B, T);
    reverse(H,Reversed_H), visibility(0, Reversed_H, N), N =:= A, check_visibility_reversed(B, T);
    false.
```

Estas duas regras servem para fazer a verificação de alturas. Ambas utilizam a regra visibility para a contagem de alturas e então as compara com a lista fornecida na entrada, retornando true quando forem compatíveis. Ambas as regras são muito parecidas, com a diferença de que a reversed serve para inverter as alturas da solução para que fiquem possíveis de serem comparadas com as alturas fornecidas na entrada.

Por fim, estas duas regras são adicionadas como condição à regra da resolução do sudoku, sendo chamadas como no exemplo abaixo:

```
sudoku_5x5(Rows, Left, Right, Top, Botton) :-
    length(Rows, 5), % retorna True se o numero de elementos na linha é n
    maplist(same_length(Rows), Rows), % verifica se cada linha tem um numero de colunas igual ao numero de linhas
    append(Rows, Vs), Vs ins 1..5, % estipula um intervalo no qual uma variavel Vs adicionada ao final de Rows deve estar
    maplist(all_distinct, Rows), % verifica se os elementos de todas as linhas são distintos
    transpose(Rows, Columns), % transpoe a tabuleiro
    maplist(all_distinct, Columns), % verifica se os elementos de todas as colunas sao distintos
    maplist(label, Rows), % aplica a função label a cada linha para obter os valores presentes em cada posicao
    check_visibility(Left, Rows),
    check_visibility_reversed(Right, Rows),
    check_visibility(Top, Columns),
    check_visibility_reversed(Botton, Columns).
```