Universidade Federal de Santa Catarina
Departamento de Informática e Estatística - INE
Curso de Ciências da Computação

Departamento de Informática e Estatística
CTC • UFSC

Alunos: Isaque Beirith e Luís Fernando Mendonça Junior
Disciplina: Paradigmas de Programação

Relatório

Análise do problema

Kojun é um quebra-cabeça lógico japonês, no qual o objetivo é completar as regiões do tabuleiro com número de 1 até N, sendo N o número de células da região. As regras definem que números em células ortogonais vizinhas devem ser diferentes e também que duas células adjacentes verticalmente na mesma região devem sempre conter o número maior na célula superior e o menor na inferior.

O problema consiste na implementação de um resolvedor de Kojun na linguagem Prolog. O código apresenta uma implementação para resolver um quebra-cabeça lógico, no qual números devem ser organizados em um tabuleiro seguindo certas restrições.

Solução implementada

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

region_size(0,5).
region_size(1,2).
region_size(2,3).
region_size(3,2).
region_size(4,6).
region_size(5,4).
region_size(6,4).
region_size(7,3).
region_size(8,2).
region_size(9,2).
region_size(10,3).
```

A estrutura do tabuleiro é definida pelo predicado **'board'**, que utiliza uma matriz onde cada célula contém informações sobre a região (identificada por números de 0 a 10) e um número associado. O tamanho de cada região é determinado pela função **'region_size'**.

```
solver(Puzzle) :-
    append(Puzzle, List),
    maplist(max_region_value, List),
    maplist(different_neighbors, Puzzle),
    transpose(Puzzle, Columns),
    maplist(different_neighbors,Columns),
    maplist(superior_greater, Columns),
    group_elements(0, List, Group0),
    group_elements(1, List, Group1),
    group_elements(2, List, Group2),
    group_elements(3, List, Group3),
    group_elements(4, List, Group4),
    group_elements(5, List, Group5),
    group_elements(6, List, Group6),
    group_elements(7, List, Group7),
    group_elements(8, List, Group8),
    group_elements(9, List, Group9),
    group_elements(10, List, Group10),
    Groups = [Group0, Group1, Group2, Group3, Group4, Group5,
              Group6, Group7, Group8, Group9, Group10],
    all_different_regions(Groups), !.
```

A lógica principal do solucionador está encapsulada no predicado **'solver'**. Ele utiliza a biblioteca CLPFD (Constraint Logic Programming over Finite Domains) para estabelecer restrições nas variáveis que representam os números do tabuleiro. As principais restrições são definidas pelos predicados **'max_region_value'**, **'different_neighbors'**, **'superior_greater'**, e **'all_different_regions'**, onde nelas temos a verificação de que o valor é válido em sua posição.

A restrição **'max_region_value'** limita os valores possíveis de cada região conforme o tamanho predefinido pela função **'region_size'**. A restrição **'different_neighbors'** garante que os valores dos vizinhos à direita em cada linha do tabuleiro são diferentes. A restrição **'superior_greater'** assegura que, se duas células pertencem à mesma região, o valor da célula superior é maior que o valor da célula inferior.

```
different_neighbors([[_,_]]).
different_neighbors([[_,X1],[R2,X2]|T]) :-
    X1 #\= X2, append([[R2,X2]],T,L), different_neighbors(L).

superior_greater([[_,_]]).
superior_greater([[R1,X1],[R2,X2]|T]) :-
    ((R1 #\= R2);
    (X1 #> X2)), append([[R2,X2]],T,L), superior_greater(L).

group_elements(_, [], []).
group_elements(R, [[R1, X1] | T], [X1 | L]) :- R #= R1, group_elements(R, T, L).
group_elements(R, [[R1, _] | T], L) :- R #\= R1, group_elements(R, T, L).

all_different_regions([H]) :-
    all_distinct(H).
all_different_regions([H|T]) :-
    all_distinct(H),
    all_different_regions(T).
```

O agrupamento de elementos é feito pelo predicado **'group_elements'**, que organiza os elementos do tabuleiro de acordo com a região a qual pertencem. A verificação da distinção dos membros em cada região é realizada pelo predicado **'all_different_regions'**.

```
solution(BoardResult) :-
    board(BoardProblem),
    solver(BoardProblem),
    extract_second_values(BoardProblem, BoardResult).

extract_second_values([], []).
extract_second_values([Sublist | Rest], [SecondValues | Result]) :-
    extract_second(Sublist, SecondValues),
    extract_second_values(Rest, Result).

extract_second([], []).
extract_second([[_, Second] | Rest], [Second | SecondValues]) :-
    extract_second(Rest, SecondValues).
```

O predicado **'solution'** coordena o processo de resolução chamando **'solver'** para encontrar uma solução e, em seguida, extrai a solução simplificada do tabuleiro através das funções **'extract_second_values' e 'extract_second'**.

Vantagens e Desvantagens

A implementação em Prolog apresenta vantagens como a sintaxe da linguagem, que facilita a implementação e leitura do código, já que apresenta uma estrutura declarativa. A utilização da biblioteca CLPFD também contribuiu para uma solução ficar mais simples, além de permitir a reusabilidade do código em problemas similares. No entanto, a eficiência pode ser limitada para tabuleiros muito grandes, e a forma um pouco atípica do Prolog pode ser um problema de início para debugar o código, além de não ser ideal para problemas que necessitam de uma manipulação de um número grande de dados.

Organização do trabalho

Como essa foi a terceira realização do jogo Kojun, o grupo já tinha a experiência de como funcionam as regras do jogo e como tentar implementá-las no programa, o desafio agora se encontrava em encontrar uma abordagem para implementação em Prolog.

Como recomendado pelo professor, começamos pesquisando pela biblioteca **'clpfd'**, e após isso fomos tentando realizar a implementação seguindo a mesma linha de raciocínio nos trabalhos anteriores.

O grupo realizou reuniões por meio de chamadas de voz para implementar o trabalho. Foi utilizada a extensão liveshare do aplicativo VSCode, que possibilita a edição de arquivos em conjunto, para que os membros pudessem escrever e editar o código de maneira síncrona.

Dificuldades encontradas

As maiores dificuldades encontradas foram as diferenças na abordagem que tivemos que tomar durante a implementação. Como não estávamos trabalhando mais com um paradigma funcional, embora nosso raciocínio para a resolução do problema fosse o mesmo, a implementação tomou uma forma bastante diferente das anteriores.

Junto dessa nova forma de abordar o problema, vieram também diversas dificuldades em debugar e descobrir quando algo estava errado em questões de lógica e sintaxe.