Universidade de Évora

Nonograma

Programação Declarativa

-Miguel Azevedo, nº36975

Professor Salvador Abreu

-Vasco Crespo, nº37913

Introdução

Este trabalho consiste na implementação de um programa que resolva um nonograma, puzzle lógico com o objetivo de preencher espaços ou deixá-los em branco dependendo das restrições que são dadas.

Para este problema decidimos utilizar a linguagem Prolog pois utilizando constraints conseguimos restringir onde preencher os espaços de tal forma que conseguimos obter apenas uma solução.

Esta solução é obtida lendo linha a linha da matriz dada correspondente às linhas das restrições e aplicando as restrições dadas na nossa matriz final que contém as possíveis soluções e fazendo o mesmo para as colunas mas tratando-as como linhas fazendo a matriz transposta e desta forma restringimos de tal forma as possibilidades que obteremos apenas uma matriz final possível.

Implementação do problema

O nosso objetivo é limitar de tal forma as possibilidades de combinações possíveis que no final fica apenas uma solução possível para o nonograma.

Para apresentar o resultado final utilizamos e manipulamos uma matriz com n listas, sendo n o número de linhas existentes na lista dada como input.

Para isto temos vários predicados:

- matrizFinal(Lista1,Lista2,N) : este predicado recebe a lista com as restrições(Lista2) e coloca na Lista1 uma matriz com N listas e onde cada lista tem o número de colunas da Lista2 e que apenas toma os valores de 0 ou 1;
- contar(Lista1, N) : este predicado conta o número de ocorrências de 1´s na Lista1 e verifica se esse valor é N;
- primeiro_um(Lista1,Lista2): este predicado avança a Lista1 até à primeira ocorrência de um 1 verificando se a Lista2 é o resto dessa lista após esse 1 inclusive;
- n_seguidos(N, Lista1,Lista2) : este predicado verifica se há N 1's seguidos na Lista1 e verifica se a Lista2 é o resto dessa Lista1 mas sem esses 1's.
- espaco_obrigatorio(Lista1, Lista2) : este predicado aplica apenas a regra do nanograma que obriga no mínimo um espaço vazio após os espaços preenchidos.
- verifica_restricoes(Lista1,Lista2) : este predicado aplica os três predicados aplicando as restrições a uma lista completa comprovando se a Lista2 cumpre todas as restrições da Lista1;
- restricoes_lista(Lista1, N, Lista2) : este predicado aplica as restrições da Lista2 à Lista1 utilizando o predicado anterior mas desta vez com um tamanho especifico da lista que é dado através de N que é o número de colunas dadas nas restrições;
- restricoes_matriz(Lista1,Lista2,N) : este predicado aplica o predicado anterior a todas as listas da Lista1(lista de listas com as restrições para as linhas) e verifica se a Lista2 é alguma possibilidade válida aplicando as restrições dadas;
- transpose(Matriz1,Matriz2) : este predicado verifica se a Matriz2 é a matriz transposta da Matriz1;
- solve(Lista1, Matriz1) : este predicado recebe a lista de listas de listas com todas as restrições aplicando todas essas restrições utilizando os predicados anteriores e comparando se a Matriz1 aceita todas essas restrições e depois aplicando o predicado transpose para aplicar novamente as restrições às linhas(anteriormente as colunas) utilizando no fim o predicado pré-definido maplist que aplica o predicado fd_labeling a todos os elementos dessa Matriz1 para obtermos valores específicos;
- switch(Matriz1,Matriz2) : predicado que troca os valores da Matriz1 que são 1's e troca-os por 'X' e os valores que são 0's troca-os por '.' e verifica se a Matriz2 é a Matriz1 mas com os valores trocados;
- printFinal(Matriz1) : este predicado apenas mostra o output como é pedido no enunciado;
- puzzle(Matriz1) : este predicado utiliza os predicados anteriores e é o predicado principal e que vai ser utilizado.

Conclusão

Por fim, consideramos que este trabalho foi uma forma de colocar em prática as diferentes funcionalidades do **Prolog** e que demonstrou ser uma linguagem muito prática e que apesar das dificuldades sentidas ao longo do trabalho, é possível resolver problemas complicados de forma mais simples.

O nosso projeto executa os exemplos com menos restrições, mais simples, contudo os exemplos com mais restrições demoram um tempo considerável a serem resolvidos.