Lógica para Programação

Relatório do Projecto

2011/2012

Projecto
Metaforms

Grupo	**Nº:** 69632	Ricardo Leitão
55	**Nº:** 69905	Diogo Pinto
	Nº: 70227	Fábio Almeida

Descrição da implementação de cada funcionalidade e a motivação para as decisões tomadas

peca/2

O Functor peca, de aridade 2, é definido no nosso projecto através das afirmações do tipo peca(forma, cor), em que temos as constantes triangulo, circulo e quadrado para a forma e amarelo, vermelho e azul para a cor. São feitas nove afirmações, representativas de cada combinação para a cor e forma das peças.

tabuleiro

O tabuleiro de jogo é definido, de acordo com o enunciado, como uma lista de 9 posições, cada uma delas definida por um par (Linha, Coluna). São assim (top,left), (top,middle),(top,right), (center,left), (center,middle), (center,right), (bottom, left), (bottom, middle) e (bottom,right). O tabuleiro é, de forma geral aquilo que ao longo dos desafios se procura definir.

coloca/4

O predicado coloca(Peca,Linha,Coluna,Tabuleiro), de aridade 4, foi definido usando 9 afirmações, cada uma delas correspondente à colocação da peça na posição do tabuleiro. Assim, a título de exemplo, no caso da colocação de uma peça na posição (top,left) do tabuleiro, o objectivo vai unificar com a afirmação coloca(Peca,top,left,[Peca,_,_,_,_,_,_,_,_]). , ficando assim a peça desejada na primeira posição da lista Tabuleiro, correspondente à posição (top,left).

Esta implementação torna-se simples no caso do tabuleiro do jogo Metaforms, pois existem apenas 9 posições no tabuleiro, deixando de ser viável para um tabuleiro que tenha muito mais posições.

quadrado/4

O predicado quadrado(Peca,Linha,Coluna,Tabuleiro), de aridade 4, foi definido usando um “ou”, ou seja, analisando todas as possibilidade de colocação de peça, de acordo com a definição dada de quadrado. Assim, para uma dada linha e coluna, a peça pode ser sempre colocada em 4 posições distintas, sendo elas a posição (Linha,Coluna), (Linha, middle), (center,Coluna) e (center,middle) pois independentemente do quadrado que seja pedido, irá ser sempre possível colocar a peça na linha e coluna central, que é a partilhada por todos os quadrados possíveis de colocar no tabuleiro. Esta solução é suficiente tendo em conta que, para uma pista relativa a um quadrado, existe apenas uma posição livre para colocar a peça pois graças ao uso de “ou” entre as várias regras, o objectivo vai unificar com a posição livre. Esta solução não é, porém, a mais elegante e correcta, visto não corresponder a uma boa prática em Prolog.

rectanguloVertical/4

O predicado rectanguloVertical(Peca,Linha,Coluna, Tabuleiro), de aridade 4, foi mais uma vez implementado usando um “ou”, ou seja, analisando todas as possibilidades de colocação de peça. Para uma dada posição só existem duas possibilidades, correspondentes ao rectângulo ocupar a coluna esquerda e central ou a coluna central e direita. Assim, é trivial a definição das posições onde a peça pode ser colocada. Parte-se do mesmo princípio que se partiu para o predicado do quadrado e por isso não se trata da solução mais elegante.

rectanguloHorizontal/4

Este predicado é semelhante ao rectângulo vertical, tendo apenas em conta que agora o rectângulo correspondente às várias posições possíveis da peça pode ocupar as linhas de topo e centro ou centro e base, pelo que a solução adoptada é semelhante à do predicado rectanguloVertical.

linhaTriplaHorizontal/3

O predicado linhaTriplaHorizontal(Peca,Coluna,Tabuleiro) corresponde a colocar uma peça numa posição de uma linha de três posições definida por Coluna. Assim, recorreu-se ao predicado coloca e a um “ou”. As várias hipóteses para unificar o predicado coloca são com (top,Coluna), (center,Coluna) ou (bottom,Coluna). Assim, a peça vai ser colocada na posição livre correspondente à coluna Coluna, podendo ser na linha top,center ou bottom.

LinhaTriplaVertical/3

O predicado linhaTriplaVertical(Peca,Linha,Tabuleiro) segue a mesma implementação do predicado anterior, diferindo na orientação das hipóteses de colocação da peça, passando agora a ser (Linha,left), (Linha, middle) e (Linha,right).

LinhaDuplaHorizontal/3

O predicado LinhaDuplaHorizontal(Peca,Coluna,Tabuleiro) foi implementado recorrendo ao predicado LinhaTriplaHorizontal, unificando a regra LinhaDuplaHorizontal com LinhaTriplaHorizontal(Peca,Coluna,Tabuleiro) ou LinhaTriplaHorizontal(Peca,middle,Tabuleiro) pois inserir uma peça numa linha dupla horizontal é exactamente o mesmo que inseri-la numa linha tripla horizontal, apenas para duas colunas.

LinhaDuplaVertical/3

O predicado LinhaDuplaVertical(Peca,Linha,Tabuleiro) foi implementado recorrendo ao predicado LinhaTriplaVertical, unificando a regra LinhaDuplaVertical com LinhaTriplavertical(Peca, Linha, Tabuleiro) ou LinhaTriplaVertical(Peca, center, Tabuleiro) pois inserir uma peça numa linha dupla vertical é exactamente o mesmo que inseri-la numa linha tripla vertical, apenas para duas linhas.

Desafios avançados e predicados auxiliares

Para a implementação dos predicados referentes aos desafios avançados, tivemos a necessidade de definir predicados auxiliares pois deixamos de ter um conjunto de posições onde a peça pode estar no tabuleiro para termos uma lista de posições onde essa mesma peça não pode estar. A nossa implementação passa por avaliar todos os casos possíveis para negar as posições das peças e usar os predicados auxiliares para, no caso de essas posições não fazerem parte da lista das posições negadas, serem possíveis de colocar a peça. Para simplificar o relatório e pelo facto de toda a lógica usada para a implementação dos predicados negados ser semelhante, vamos apenas explicar os predicados matrizNeg, colocaAuxiliar, quadradoNeg e quadradoNegaAuxiliar, que são representativos da implementação realizada para todos os restantes predicados negados.

colocaAuxiliar/5

O predicado colocaAuxiliar(Peca,Lista,Linha,Coluna,Tabuleiro) verifica se a posição (Linha,Coluna) está na lista ou não e se não estiver coloca no tabuleiro a peça na posição (Linha,Coluna). Usamos este colocaAuxiliar para conseguir passar da lista das posições onde não podemos colocar a peça para as posições onde esta pode ser colocada. Para avaliar se a posição está na lista ou não, usamos o predicado do prolog member/2 que recebe um elemento e uma lista e verifica se esse elemento pertence à lista ou não. Assim, definimos que, se member((Linha,coluna),Lista) não unificar, colocamos a peça na posição (Linha,Coluna).

matrizNeg/3

O predicado matrizNeg(Peca,Lista,Tabuleiro) vai usar o predicado colocaAuxiliar e testar para todas as posições, se cada uma delas está na lista das posições onde a peça não pode ser colocada e caso essa posição não esteja na lista, vamos tentar inserir a peça nessa mesma posição. Como partimos do princípio que só é possível uma solução, só vamos conseguir unificar a regra uma vez para obtermos a posição onde a peça deve efectivamente ficar.

quadradoNegaAuxiliar/5

O predicado quadradoNegaAuxiliar(Peca,Lista,Linha,Coluna,Tabuleiro) verifica se a posição (Linha,Coluna) está na lista ou não e se não estiver colocamos essa peça na posição (Linha,Coluna) de um quadrado 2x2 que pode estar em qualquer lugar do tabuleiro. Usamos o predicado do prolog member/2 que recebe um elemento e uma lista e verifica se esse elemento pertence à lista ou não.

quadradoNeg/3

O predicado quadradoNeg(Peca,Lista,Tabuleiro) vai usar o predicado quadradoNegaAuxiliar e testar para todas as posições de um quadrado que pode estar colocado em qualquer local do tabuleiro, se cada uma delas está na lista das posições onde a peça não pode ser colocada e caso essa posição não esteja na lista, vamos tentar inserir a peça numa posição do quadrado. Como partimos do princípio que só é possível uma solução, só vamos conseguir unificar a regra uma vez para obtermos a posição onde a peça deve efectivamente ficar.

Principais problemas encontrados na realização do projecto

O principal problema encontrado na realização deste projecto foi o facto de se tratar de um novo paradigma de programação e numa maneira completamente diferente de raciocinar em relação à programação imperativa. A programação em lógica tem inúmeras vantagens e permite-nos simplificar muitas implementações que à primeira vista poderiam parecer complicadas. Porém, temos consciência que não aplicamos todo o potencial deste paradigma de programação. Poderíamos ter usado a recursividade para inúmeras implementações o que faria o nosso código mais funcional e poderoso. Temos algumas falhas no código, que não está preparado para, por exemplo, lidar com a possibilidade de para uma pista, haver possibilidade de colocar a peça em duas posições distintas. Ao longo de todo o projecto optámos por fazer uma verificação exaustiva de todas as possibilidades, quer de colocação da peça quer de impossibilidade de colocação, o que neste caso de um tabuleiro 3x3 é viável mas que poderia ser muito complicado caso as dimensões do tabuleiro fossem superiores, pelo que teríamos de seguir outra abordagem.