

# Projeto de Lógica para Programação 2013/2014

META-FORMS<sup>TM</sup>  
LEIC, Instituto Superior Técnico

## 1 Introdução

Este projeto visa a extensão da versão implementada em 2011/2012 pelos alunos de LP, em Prolog, do jogo META-FORMS<sup>TM</sup>, criado por Michael & Robert Lyons. O objectivo do jogo é, com base num conjunto de pistas, colocar 9 figuras distintas num tabuleiro 3x3 (Figura 1).

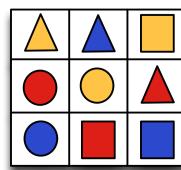


Figura 1: Solução de um desafio

Cada conjunto de pistas será, doravante, denominado *desafio*. Segue-se a ilustração de um desafio (Figura 2), do qual a Figura 1 é a solução.

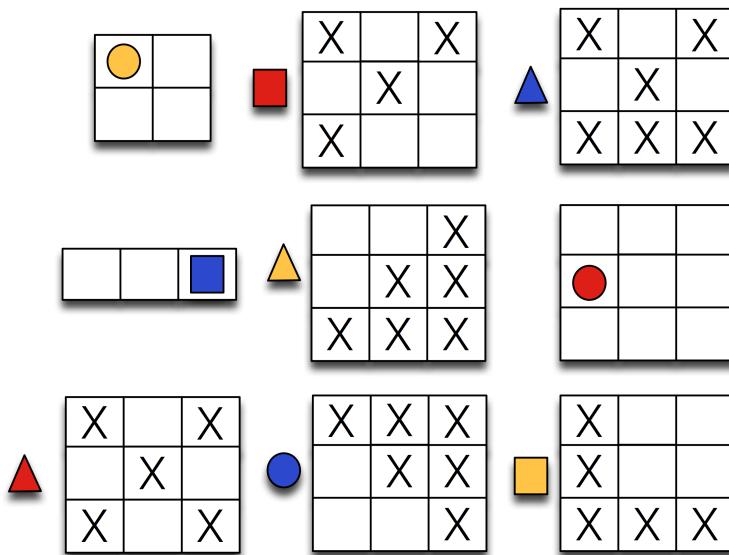


Figura 2: Exemplo de um desafio

A pista relativa ao círculo vermelho indica explicitamente a posição a ocupar no tabuleiro por esta peça. As pistas relativas ao círculo amarelo e quadrado azul são dadas por tabuleiros parciais. Poderá constatar, após uma breve reflexão, que a primeira indica que o círculo amarelo pode ocupar uma das quatro posições do quadrado 2x2 do topo esquerdo do tabuleiro 3x3 (Figura 3); a segunda que o quadrado azul pode estar em qualquer posição da última coluna do tabuleiro. Todas as outras pistas, explicitam onde é que as figuras **não** podem ocorrer.

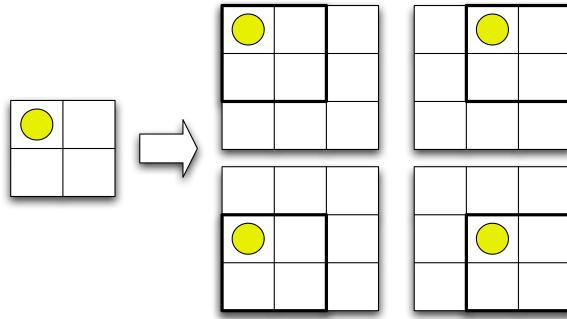


Figura 3: Pista relativa ao círculo amarelo

Na secção 2 apresentam-se o jogo e as pistas/predicados a implementar, na secção 3 precisam-se alguns detalhes da implementação e, na secção 4, explica-se como deve ser feita a entrega e como vai ser avaliado o projeto. No anexo A pode consultar uma descrição das pistas implementadas no projeto de LP de 2011/2012.

## 2 O META-FORMS<sup>TM</sup> em Prolog

### 2.1 O tabuleiro

Identifica-se cada casa do tabuleiro por um par, em que o primeiro elemento representa a linha e o segundo a coluna ocupadas por essa casa, sendo as linhas identificadas pelas constantes `top`, `center` e `bottom` e as colunas por `left`, `middle` e `right`, tal como mostra a Figura 4.

(top, left)	(top, middle)	(top, right)
(center, left)	(center, middle)	(center, right)
(bottom, left)	(bottom, middle)	(bottom, right)

Figura 4: Denominação das casas do tabuleiro

Quanto à representação do tabuleiro em Prolog, este é representado por uma lista com 9 elementos, em que as três primeiras casas correspondem à linha de topo, as três seguintes à linha do centro e as três últimas à linha inferior do tabuleiro (a correspondência deve ser feita da esquerda para a direita).

### 2.2 As peças

Consideram-se três formas de peças – triângulo, círculo e quadrado – e três cores – azul, amarelo e vermelho. As nove peças são combinações de uma forma com uma cor, sendo as peças únicas para cada par (forma, cor), ou seja, existe um e apenas um triângulo amarelo, um azul, etc.

Cada peça será representada em Prolog através do functor `peça`, de aridade 2, sendo que as constantes `triangulo`, `circulo` e `quadrado` são usadas para representar, respectivamente, as formas triângulo, círculo e quadrado, e as constantes `azul`, `amarelo` e `vermelho` usadas para representar as cores com o mesmo nome. Assim, por exemplo, o termo `peça(quadrado, azul)` representa o quadrado azul.

## 2.3 As pistas

Como referido, no anexo A pode consultar uma descrição das pistas anteriormente implementadas (o código relativo a estas pistas é fornecido com o enunciado deste ano). Segue-se uma descrição do que terá de fazer este ano.

### 2.3.1 As pistas intermédias a implementar

As pistas a implementar são dadas através de representações parciais do tabuleiro, às quais chamamos pistas intermédias (representam indicações indiretas para a colocação de uma peça – ver apêndice A para as pistas indirectas anteriormente implementadas). Assuma que a variável `Peca` representa a peça em jogo, as variáveis `Linha` e `Coluna`, respectivamente, a linha e a coluna onde a peça deve ser colocada e `Tabuleiro` o tabuleiro em uso. As pistas a implementar são:

1. `trioLeft(Peca, Linha, Coluna, Tabuleiro)`
2. `trioRight(Peca, Linha, Coluna, Tabuleiro)`
3. `cobra(Peca, Linha, Coluna, Tabuleiro)`
4. `tSimples(Peca, Linha, Coluna, Tabuleiro)`
5. `tLeft(Peca, Linha, Coluna, Tabuleiro)`
6. `tRight(Peca, Linha, Coluna, Tabuleiro)`
7. `tInvertido(Peca, Linha, Coluna, Tabuleiro)`
8. `cantoTopLeft(Peca, Linha, Coluna, Tabuleiro)`
9. `cantoTopRight(Peca, Linha, Coluna, Tabuleiro)`
10. `cantoBottomLeft(Peca, Linha, Coluna, Tabuleiro)`
11. `cantoBottomRight(Peca, Linha, Coluna, Tabuleiro)`
12. `diagonalGrave(Peca, Linha, Coluna, Tabuleiro)`
13. `diagonalAguda(Peca, Linha, Coluna, Tabuleiro)`

A Figura 5 ilustra cada uma destas pistas.

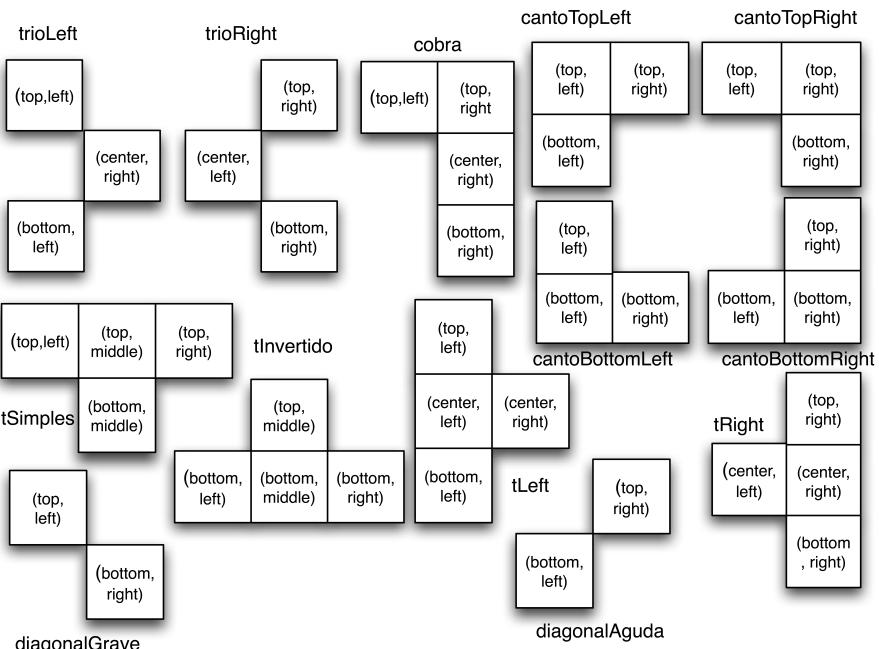


Figura 5: Figuras associadas às novas pistas a implementar.

Como exemplo, a pista da Figura 6 indica que o circulo amarelo pode ser encontrado no tabuleiro 3x3, nas posições (center, left), (center, middle), (bottom, left) ou (bottom, middle).

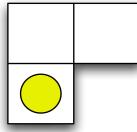


Figura 6: Exemplo de nova pista a implementar.

Em Prolog, esta pista seria dada por:

```
cantoTopLeft(peca(circulo, amarelo), bottom, left, Tabuleiro)
```

### 2.3.2 Predicado check/2 a implementar

Ao contrário do projeto anterior, em que todas as peças tinham uma pista associada, este ano **poderão não ser fornecidas pistas sobre alguma peça**, devendo o programa implementado inferir a sua localização. Também ao contrário do projeto anterior, desta vez podem ser dadas pistas parciais relativas às peças (por exemplo, considerando de novo a Figura 6, podia ser indicado que a peça em causa era um circulo sem indicação da cor, ou que era amarela, sem indicação da forma).

No entanto, tal como no projeto anterior, **a solução final será o tabuleiro totalmente preenchido, em que nenhuma peça se repete e todas estão representadas no tabuleiro**. Com este fim deve implementar o predicado `check/2` que recebe um tabuleiro eventualmente incompleto e devolve o mesmo tabuleiro, mas com as peças em falta. Como exemplo, a resposta a

```
check([peca(triangulo, azul), peca(circulo, amarelo),
      peca(circulo, azul), peca(triangulo, amarelo),
      peca(triangulo, vermelho), peca(quadrado, vermelho),
      _, peca(quadrado, azul), peca(circulo, vermelho)],
      TabuleiroCompleto).
```

seria

```
TabuleiroCompleto = [peca(triangulo, azul), peca(circulo, amarelo),
                     peca(circulo, azul), peca(triangulo, amarelo),
                     peca(triangulo, vermelho), peca(quadrado, vermelho),
                     peca(quadrado, amarelo), peca(quadrado, azul),
                     peca(circulo, vermelho)].
```

No caso de haver mais do que uma solução, qualquer solução apresentada pelo aluno será aceite desde que as pistas sejam respeitadas.

## 3 Detalhes de implementação

### 3.1 Ferramenta

Recomenda-se o swi-prolog, dado que é o que vai ser usado na avaliação do projeto (em <http://www.swi-prolog.org/> encontrará ferramentas para as várias plataformas).

### 3.2 Testes e formato dos dados

Os alunos têm à sua disposição uma bateria de testes representando vários desafios, servindo os ficheiros `testes-alunos.pl`, `resultados.txt`, `correprojeto.pl` e `metaforms12.pl` para levar a cabo esses testes.

O ficheiro `testes-alunos.pl` contém 5 desafios e o ficheiro `resultados.txt` a sua solução; o ficheiro `metaforms12.pl` contém uma sugestão de implementação do projeto do ano de 2011/2012. O ficheiro `correprojeto.pl` contém o código abaixo. TODO o código dos alunos respeitante ao projeto deste ano (isto é, a implementação dos predicados referentes às pistas descritas na secção 2.3.1, bem como o predicado `check/2`) deve estar no ficheiro `metaforms14.pl` e é esse (e apenas esse) o ficheiro que deve ser entregue.

```

start :- carregaprojeto, desafios.
carregaprojeto :- ['metaforms12.pl'], ['metaforms14.pl'], ['testes-alunos.pl'].
desafios :- correTodosDesafios(1, 6).

```

Quando tiver implementado o seu projeto e para o testar sugere-se a utilização da seguinte linha de comando<sup>1</sup> que permitirá guardar os resultados do programa no ficheiro `myresultados.txt`:

```
swipl -s correprojeto.pl -g start -t halt > myresultados.txt
```

A avaliação será feita em moldes semelhantes aos testes e é muito importante que sejam respeitadas todas estas indicações. A título de exemplo, após o seguinte desafio de `testes.pl`:

```

desafio(2, TabuleiroFinal) :-
    coloca(peca(triangulo, vermelho), top, right, Tabuleiro),
    tSimples(peca(circulo, azul), top, right, Tabuleiro),
    tLeft(peca(circulo, vermelho), center, right, Tabuleiro),
    trioRight(peca(triangulo, amarelo), center, left, Tabuleiro),
    cantoTopLeft(peca(quadrado, vermelho), top, right, Tabuleiro),
    cantoTopLeft(peca(circulo, amarelo), top, left, Tabuleiro),
    rectanguloHorizontal(peca(triangulo, azul), bottom, right, Tabuleiro),
    diagonalAguda(peca(quadrado, amarelo), bottom, left, Tabuleiro),
    check(Tabuleiro, TabuleiroFinal).

```

A lista `TabuleiroFinal` deverá ter o seguinte conteúdo:

```
[peca(circulo,amarelo),peca(quadrado,vermelho),peca(triangulo,vermelho),
 peca(triangulo,amarelo),peca(circulo,vermelho),peca(circulo,azul),
 peca(quadrado,amarelo),peca(quadrado,azul),peca(triangulo,azul)]
```

## 4 Entrega e avaliação

O ficheiro `metaforms14.pl` deve ser entregue via Fénix até às 23h 59m do dia **20 de Maio**. Dado que não haverá relatório, o código deverá estar devidamente comentado, de modo a que seja comprehensível o racional da implementação das pistas em jogo. O ficheiro deverá estar em UTF-8 e em lado algum deverão ser usados caracteres acentuados ou cedilhas.

Será levada a cabo uma **avaliação automática** com desafios diferentes dos usados nos testes; será igualmente avaliada a **qualidade do código** (uso adequado do Prolog, comentários, paragrafação, ausência de *warnings*, nomes de predicados e de variáveis cuidadosamente escolhidos, etc.).

Os alunos devem ainda ter em conta que:

- Projetos que não respeitem os formatos terão 0 na avaliação automática.
- Caso se detectem cópias os alunos terão 0 no projeto e não poderão fazer LP este semestre.
- Poderá haver uma discussão oral do projeto e/ou uma demonstração do seu funcionamento (a ser decidido caso a caso).

Bom trabalho e uma dica: *em Roma sê Romano*, ou seja, se vai trabalhar em Prolog, aproveite o que o Prolog tem para oferecer.

## Apêndice A

### Pistas básicas

A pista mais básica corresponde à indicação explícita do local onde a peça deve ocorrer no tabuleiro. Por exemplo, a pista dada na Figura 7, indica que o triângulo vermelho deve ocorrer na posição (`center, left`).

O predicado `coloca/4` é usado para representar esta pista, em Prolog. Por exemplo, a pista da Figura 7 é representada por `coloca(peca(triangulo, vermelho), center, left, Tabuleiro)`.

---

<sup>1</sup> Atenção que a chamada ao Prolog pode mudar de acordo com o sistema operativo.

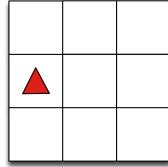


Figura 7: Ocorrência do triângulo vermelho em (center, left).

## Pistas de nível intermédio

As pistas intermédias implementadas no projeto de 2011/2012 são as da Figura 8.

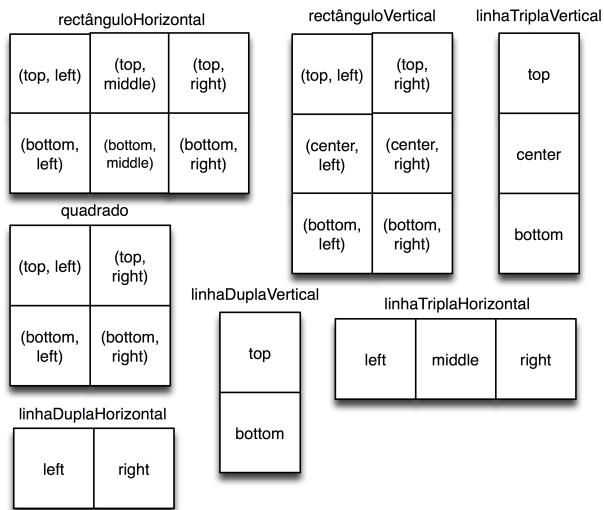


Figura 8: Figuras associadas às pistas intermédias.

Assim, por exemplo, a pista da Figura 9 corresponde a dizer que o triângulo vermelho tem de ser colocado numa posição central, seja na coluna da esquerda (**left**), na do meio (**middle**) ou na da direita (**right**).

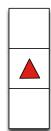


Figura 9: Pistas relativas à localização do triângulo vermelho.

Para representar cada uma destas pistas consideraram-se os predicados que se seguem:

1. `rectanguloVertical(Peca, Linha, Coluna, Tabuleiro)`
2. `rectanguloHorizontal(Peca, Linha, Coluna, Tabuleiro)`
3. `linhaTriplaVertical(Peca, Linha, Tabuleiro)`
4. `linhaTriplaHorizontal(Peca, Coluna, Tabuleiro)`
5. `quadrado(Peca, Linha, Coluna, Tabuleiro)`
6. `linhaDuplaVertical(Peca, Linha, Tabuleiro)`
7. `linhaDuplaHorizontal(Peca, Coluna, Tabuleiro)`

## Pistas de nível avançado

As pistas ditas de nível avançado representam restrições indicando onde NÃO deve ser colocada uma peça. Estas pistas podem ser dadas por um tabuleiro 3x3, bem como pelos tabuleiros parciais correspondentes às pistas positivas (Figura 8). Assim, por exemplo, a Figura 10 representa duas pistas: a primeira indica que o triângulo vermelho não pode estar nas posições (top, left), (center, left) e (center, middle); a segunda indica que o triângulo vermelho não pode estar em nenhuma posição do centro.

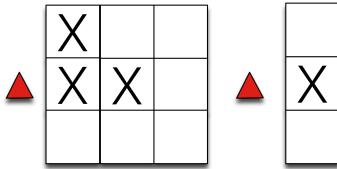


Figura 10: Pistas de nível avançado sobre a localização do triângulo vermelho.

Em termos de predicados, consideraram-se os seguintes (assuma que a variável `Peca` representa a peça em jogo, `ListaNeg` a lista com as posições onde a peça não deve ser colocada (uma espécie de lista negativa) e `Tabuleiro` o tabuleiro em uso):

1. `matrizNeg(Peca, ListaNeg, Tabuleiro)`
2. `rectanguloVerticalNeg(Peca, ListaNeg, Tabuleiro)`
3. `rectanguloHorizontalNeg(Peca, ListaNeg, Tabuleiro)`
4. `linhaTriplaVerticalNeg(Peca, ListaNeg, Tabuleiro)`
5. `linhaTriplaHorizontalNeg(Peca, ListaNeg, Tabuleiro)`
6. `quadradoNeg(Peca, ListaNeg, Tabuleiro)`
7. `linhaDuplaVerticalNeg(Peca, ListaNeg, Tabuleiro)`
8. `linhaDuplaHorizontalNeg(Peca, ListaNeg, Tabuleiro)`

Deve ser claro que `ListaNeg` contém pares apenas nos casos da `matrizNeg`, `rectanguloVerticalNeg`, `rectanguloHorizontalNeg` e `quadradoNeg`, dado que nestas situações é necessário indicar as linhas e as colunas para definir as posições onde não ocorrem as peças. Por exemplo, as pistas da Figura 10 seriam indicadas recorrendo, respectivamente, aos predicados:

```
matrizNeg(peca(triangulo, vermelho), [(top, left), (center, left), (center, middle)], Tabuleiro)
linhaTriplaVerticalNeg(peca(triangulo, vermelho), [center], Tabuleiro).
```