Origem do PROLOG

- A Linguagem PROLOG foi criada nos anos 70 por Alain Colmareur, na Universidade de Marselha
- O nome da linguagem vem de PROgramming in LOGic, ou seja, segue o paradigma da Programação em Lógica
- É conjuntamente com a linguagem LISP, criada nos anos 50, uma das linguagens específicas para o desenvolvimento de Sistemas de Inteligência Artificial
- Enquanto que a linguagem LISP teve impacto nos EUA, o PROLOG alcançou notoriedade na Europa e no Japão
- O principal standard de PROLOG foi proposto em Edimburgo, por Clocksin & Mellish

Lógica e PROLOG

- A linguagem PROLOG usa o "Princípio da Resolução" para a "Prova de Teoremas", onde a solução de problemas é encontrada directamente por um conjunto de "Axiomas" ou indirectamente através da "Inferência Lógica"
- A Prova de Teoremas segue a "Pesquisa Primeiro em Profundidade", na tentativa de provar o teorema, a este mecanismo acrescenta-se o "retorno atrás quando ocorre falha" (backtracking)
- As variáveis podem residir num de dois estados: não instanciadas ou instanciadas, quando é encontrada uma solução podem ser exibidas as instanciações possíveis
- Quando algo não está explicitamente definido como um axioma é assumido como sendo falso (Assumpção do Mundo Fechado). Há várias extensões ao PROLOG que assumem uma lógica tri-valor (verdadeiro, falso ou desconhecido)
- Em PROLOG é possível criar/remover dinamicamente axiomas

Conceitos Básicos do PROLOG

Em PROLOG temos 3 conceitos básicos:

- **Factos**
  correspondem a axiomas
  é um tipo das *cláusulas* do PROLOG
  devem ser colocados no ficheiro da Base de Conhecimento
  p.ex.: `rio(douro).`
  `pai(pedro,ana).`

- **Regras**
  correspondem a implicações
  é outro tipo das *cláusulas* do PROLOG
  devem ser colocadas no ficheiro da Base de Conhecimento
  p. ex.: `neto(N,A):-filho(N,P),(descendente(P,A,_);descendente(P,_,A)).`

- **Questões**
  permitem interrogar a Base de Conhecimento, a partir da consola, o prompt é ?-
  as respostas correspondem a soluções possíveis
  p.ex.: `?-pai(P,ana).`

  `?-neto(rui,A).`

Factos

Um facto tem 1 functor (nome genérico do facto, começado por uma minúscula), geralmente têm um ou mais argumentos, englobados por parêntesis e separados por vírgulas, e termina com um ponto (o finalizador do PROLOG).

Poderão não existir argumentos.

Os argumentos são geralmente "átomos" (valores constantes, números ou strings começadas por minúsculas ou entre ''), podendo ser também variáveis (começadas por uma maiúscula).

Exemplos de Factos

casados(rui,isabel).

Functor: casados
Argumentos: 2 átomos – rui e isabel
Terminador: .

ligado.

Functor: ligado
Argumentos: nenhum
Terminador: .

potência(X,0,1).

Functor: potência
Argumentos: 3 – uma variável e 2 valores numéricos
Terminador: .

Uma Base de Conhecimento com factos

```
fica(porto,portugal).
fica(lisboa,portugal).
fica(coimbra,portugal).
fica(caminha,portugal).
fica(madrid,espanha).
fica(barcelona,espanha).
fica(zamora,espanha).
fica(orense,espanha).
fica(toledo,espanha).

passa(douro,porto).
passa(douro,zamora).
passa(tejo,lisboa).
passa(tejo,toledo).
passa(minho,caminha).
passa(minho,orense).
```

Questões sobre a Base de Conhecimento

fica(porto,portugal).	falha
fica(lisboa,portugal).	falha
fica(coimbra,portugal).	falha
fica(caminha,portugal).	falha
fica(madrid,espanha).	sucesso
fica(barcelona,espanha).	
fica(zamora,espanha).	
fica(orense,espanha).	
fica(toledo,espanha).	
passa(douro,porto).	falha
passa(douro,zamora).	falha
passa(tejo,lisboa).	falha
passa(tejo,toledo).	falha
passa(minho,caminha).	falha
passa(minho,orense).	falha

?-fica(madrid,espanha).
yes

?-passa(mondego,coimbra).
no

Usando variáveis nas questões sobre a Base de Conhecimento

fica(porto,portugal).	falha
fica(lisboa,portugal).	falha
fica(coimbra,portugal).	falha
fica(caminha,portugal).	falha
fica(madrid,espanha).	sucesso
fica(barcelona,espanha).	
fica(zamora,espanha).	
fica(orense,espanha).	
fica(toledo,espanha).	
passa(douro,porto).	falha
passa(douro,zamora).	falha
passa(tejo,lisboa).	sucesso
passa(tejo,toledo).	
passa(minho,caminha).	
passa(minho,orense).	

?-fica(X,espanha).
X=madrid *<cr>*
yes

?- passa(tejo,C).
C=lisboa *<cr>*
yes

?-fica(X,Y).
X=porto Y=portugal *<cr>*
yes

Questões sobre a Base de Conhecimento pedindo alternativas com ;

```
fica(porto,portugal).        falha
fica(lisboa,portugal).       falha
fica(coimbra,portugal).      falha
fica(caminha,portugal).      falha
fica(madrid,espanha).        sucesso
fica(barcelona,espanha).     sucesso
fica(zamora,espanha).        sucesso
fica(orense,espanha).        sucesso
fica(toledo,espanha).        sucesso

passa(douro,porto).
passa(douro,zamora).
passa(tejo,lisboa).
passa(tejo,toledo).
passa(minho,caminha).
passa(minho,orense).
```

```
?-fica(X,espanha).
X=madrid ;
X=barcelona ;
X=zamora ;
X=orense ;
X=toledo
yes
```


Questões sobre a Base de Conhecimento pondo as alternativas numa lista

```
fica(porto,portugal).        falha
fica(lisboa,portugal).       falha
fica(coimbra,portugal).      falha
fica(caminha,portugal).      falha
fica(madrid,espanha).        sucesso
fica(barcelona,espanha).     sucesso
fica(zamora,espanha).        sucesso
fica(orense,espanha).        sucesso
fica(toledo,espanha).        sucesso
```

```
?-findall(X,fica(X,espanha),L).
L=[madrid,barcelona,zamora,orense,espanha]
yes
```

Questões sobre a Base de Conhecimento com conjunção (, na questão)

```
fica(porto,portugal).          sucesso
fica(lisboa,portugal).
fica(coimbra,portugal).
fica(caminha,portugal).
fica(madrid,espanha).
fica(barcelona,espanha).
fica(zamora,espanha).
fica(orense,espanha).
fica(toledo,espanha).

passa(douro,porto).            sucesso
passa(douro,zamora).
passa(tejo,lisboa).
passa(tejo,toledo).
passa(minho,caminha).
passa(minho,orense).
```

```
?-fica(X,portugal),passa(R,X).
X=porto R=douro <cr>
yes
```

E se fossem pedidas alternativas com o ; ?

```
?-fica(X,portugal),passa(R,X).
X=porto R=douro ;
X=lisboa R=tejo ;
X=caminha R=minho
```


Questões sobre a Base de Conhecimento com disjunção (; na questão)

```
fica(porto,portugal).          sucesso
fica(lisboa,portugal).
fica(coimbra,portugal).
fica(caminha,portugal).
fica(madrid,espanha).
fica(barcelona,espanha).
fica(zamora,espanha).
fica(orense,espanha).
fica(toledo,espanha).

passa(douro,porto).
passa(douro,zamora).
passa(tejo,lisboa).
passa(tejo,toledo).
passa(minho,caminha).
passa(minho,orense).
```

```
?-fica(X,portugal);fica(X,espanha).
X=porto <cr>
yes
```

E se fossem pedidas alternativas com o ; ?

```
?-fica(X,portugal);fica(X,espanha).
X=porto ;
X=lisboa ;
X=coimbra ;
X=caminha ;
X=madrid ;
X=barcelona ;
X=zamora ;
X=orense ;
X=toledo
yes
```

Questões sobre a Base de Conhecimento com negação (not)

fica(porto,portugal).	sucesso
fica(lisboa,portugal).	
fica(coimbra,portugal).	
fica(caminha,portugal).	
fica(madrid,espanha).	
fica(barcelona,espanha).	
fica(zamora,espanha).	
fica(orense,espanha).	
fica(toledo,espanha).	
passa(douro,porto).	falha
passa(douro,zamora).	falha
passa(tejo,lisboa).	falha
passa(tejo,toledo).	falha
passa(minho,caminha).	falha
passa(minho,orense).	falha

```
?-not fica(porto,portugal).
no

?-not passa(mondego,coimbra).
yes
```


Regras

Uma regra tem 1 termo (functor e argumentos) do lado esquerdo que corresponde a aquilo que se pretende provar (conclusão), seguido do ":-".

À direita do ":-" podem aparecer outros termos afectados por operadores (, ; not) e por primitivas de controlo. No fundo o lado direito corresponde às condições da regra.

Tal como nos factos, podemos ter regras com o mesmo nome e mesmo nº de argumentos, ou até com o mesmo nome e nº de argumentos diferentes.

Exemplos de Regras

filho(X,Y):-homem(X),(descendente(X,Y,_);descendente(X,_,Y)).

Assume-se que descendente(A,B,C) indica que A é filho do pai B e da mãe C

A regra deve ser lida do seguinte modo:

SE X é homem E (X é descendente do seu pai Y OU
X é descendente da sua mãe Y)
ENTÃO X é filho de Y

O _ corresponde a uma variável da qual não necessitamos o valor

potência(_,0,1):-!.
potência(X,N,P):- N1 is N-1,potência(X,N1,P1),P is X*P1.

Esta é uma definição recursiva da potência inteira e não negativa de um nº

A recursividade é muito comum nas regras do PROLOG

O ! (cut) é uma primitiva de controlo que será explicada posteriormente