Introdução a Prolog
Aula 1

Programação em lógica

Programação em lógica é um formalismo lógico-computacional fundamentado em três princípios básicos:

- uso de **linguagem formal** para representação de conhecimento
- uso de **regras de inferência** para manipulação de conhecimento
- uso de uma **estratégia de busca** para controle de inferências

Programação em lógica

Uma linguagem formal é precisa, suas sentenças
✓são objetos (fórmulas) com significado único
✓têm sintaxe e semântica bem definidas
mas também pode ser menos expressiva que a natural.

Regra de inferência é um padrão de manipulação sintática:
• permite criar novas fórmulas a partir de outras existentes
• em geral, simulam formas de raciocínio válidas
Ex.: *Se neva, faz frio. Está nevando. Logo, está frio*.

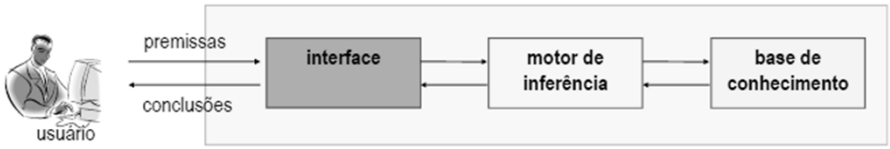
Estratégia de busca serve para decidir que parte do conhecimento armazenado deve ser explorada em busca da solução de um determinado problema.

Comparativo

PROGRAMAS CONVENCIONAIS	PROGRAMAS EM LÓGICA
PROCESSAMENTO NUMERICO	PROCESSAMENTO SIMBOLICO
SOLUCOES ALGORITMICAS	SOLUCOES HEURISTICAS
ESTRUTURAS DE CONTROLE E CONHECIMENTO INTEGRADAS	ESTRUTURAS DE CONTROLE E CONHECIMENTO SEPARADAS
DIFICIL MODIFICACAO	FACIL MODIFICACAO
SOMENTE RESPOSTAS TOTALMENTE CORRETAS	INCLUEM RESPOSTAS PARCIALMENTE CORRETAS
SOMENTE A MELHOR SOLUCAO POSSIVEL	INCLUEM TODAS AS SOLUCOES POSSIVEIS

Prolog

Prolog é o sistema de programação em lógica mais popular que existe!



- **Interface:** permite que o usuário entre com premissas codificadas em uma linguagem lógica e faça consultas para extrair conclusões destas premissas
- **Motor de inferência:** atualiza a base de conhecimento com premissas fornecidas pelo usuário e faz inferências para extrair informações implícitas
- **Base de conhecimento:** armazena as premissas fornecidas pelo usuário

Características do Prolog

- É uma linguagem orientada ao processamento simbólico
- Representa uma implementação da lógica como linguagem de programação
- Apresenta uma semântica declarativa inerente à lógica
- Permite a definição de programas reversíveis, isto é, programas que não distinguem entre os argumentos de entrada e os de saída
- Permite a obtenção de respostas alternativas

Características do Prolog

- Suporta código recursivo para a descrição de processos e problemas, dispensando os mecanismos tradicionais de controle, tais como while, repeat, etc.
- Permite associar o processo de especificação ao processo de codificação de programas
- Representa programas e dados através do mesmo formalismo
- Incorpora facilidades computacionais extralógicas e metalógicas

Prolog

- Recomenda-se o uso do **SWIProlog**, porém existem dezenas de sistemas Prolog disponíveis no mercado (sistemas comerciais ou acadêmicos): Amzi!, ALS, Arity, Eclipse, LPA, Quintus, SICStus, SWI, XBS.
- Praticamente todos estes sistemas rodam em Windows e Linux.
- O **SWI-Prolog** é desenvolvido pela Universidade de Amsterdam, é um software com distribuição livre para ensino e pesquisa. Ele usa poucos recursos da máquina, é rápido para compilação e possui documentação on-line (http://www.swi-prolog.org/) e implementa uma versão do Prolog ISO.

Programando em Prolog com “Elegância”

Há um estilo de como programar **bem** em Prolog? **SIM** !! Contudo, tal elegância ou estilo de programação não é trivialmente expresso sob a forma de regras. Algumas dicas:

- Entenda **profundamente** o problema a ser escrito em Prolog. Um problema mal entendido, dificilmente será bem implementado (se é que for);
- Escreva da mesma maneira que o problema é montado mentalmente. Assim como se fala, se escreve em Prolog. “**Declare o problema, e não o resolva**”, esta é uma das máximas do Prolog;

Programando em Prolog com “Elegância”

Há um estilo de como programar **bem** em Prolog? **SIM** !! ... Algumas dicas:

- Evite o uso de operadores tradicionais como >, <=, is . . . etc., isso revela uma proximidade com a programação procedural. O foco do Prolog é “**Casamento de Padrões**”. Pense que dois objetos podem ser equivalentes ou diferentes, apenas isto. Logo, eles casam (*match*) ou não;
- Quando uma implementação começa ficar complicada, é porque algo assumido previamente está errado. Volte atrás, e refaça tudo sob um outro enfoque. Refazer um novo programa por inteiro normalmente é mais simples do que “*forçar*” um problema mal estruturado, convergir em uma solução aceitável;

Programando em Prolog com “Elegância”

Há um estilo de como programar **bem** em Prolog? **SIM** !! ...
Algumas dicas:

- Consulte os “*grandes mestres*” em Prolog. Aprender Prolog com programadores experientes é uma boa dica. Na lista de discussão do SWI-Prolog também é fácil encontrar muitos peritos em Prolog.

Objetos

- **Alfabeto:**

  Pontuação: ( ) . ' "

  Conectivos: , (conjunção) ; (disjunção) :- (implicação)

  Letras: a, b, c, ..., z, A, B, C, ..., Z

  Dígitos: 0, 1, 2, ..., 9

  Especiais: + - * / < > = : _ ... etc.

- **Constantes:** números, compreendendo os inteiros e os reais.

1	1812	0	-273
3.14159	0.000023	-273.16	

Números reais não são muito utilizados em programas Prolog típicos (linguagem orientada ao processamento simbólico, não-numérico).

Objetos

- **Constantes:** átomos

a. Cadeias de letras e/ou dígitos, podendo conter o caracter especial sublinhado (_), iniciando sempre com minúscula: x_Y_Z xyz

b. Cadeias de caracteres especiais:

<--------> ::= =/= ======> ... ++++

c. Cadeias de caracteres quaisquer, podendo inclusive incluir espaços em branco, desde que delimitados por apóstrofos ('):

'D. Pedro I‘

Objetos

- **Variáveis:** cadeias de letras, dígitos e do caracter sublinhado (_), devendo iniciar com este ou com maiúscula. Usadas em consultas ou regras.

  X Objeto2 _var35

- **Variável anônima:** representada pelo caracter _ e usada quando seu valor específico for irrelevante numa determinada consulta ou regra.

O escopo léxico de nomes de variáveis é uma única cláusula:

=> toda ocorrência de X25 dentro da mesma cláusula quer significar a mesma variável.

=> caso X25 ocorre em 2 cláusulas diferentes, então se está representando 2 variáveis diferentes.

Operadores

- **Aritméticos**

  Prolog oferece um predicado especial ***is***, bem como um conjunto operadores, através dos quais podemos efetuar operações aritméticas.

  ?- X is 2+3. X = 5

  Os operadores aritméticos são + (adição), - (subtração), * (multiplicação), mod (resto), / (divisão real), // (divisão inteira) e ^ (potenciação).

- **Relacionais**

  Para realizar comparações numéricas podemos usar os seguintes predicados primitivos: =:= (igual) , =\= (diferente), > (maior), >= (maior ou igual), < (menor) e =< (menor ou igual).

  Para realizar comparações simbólicas podemos usar os seguintes predicados primitivos: == (igual) e \== (diferente).

Características do Prolog

- **Fatos -** estabelecer um relacionamento existente entre objetos.

  masculino(jorge). => predicado(átomo)

  feminino(iris).

  gerou(iris, jorge).

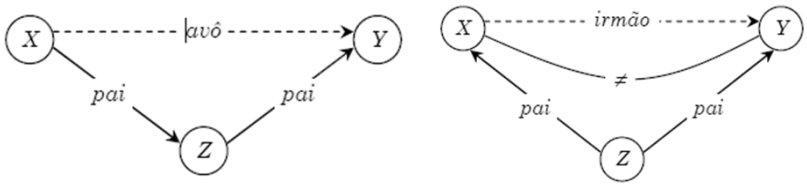
- **Construções Recursivas**

  antepassado(X, Z) :- gerou(X, Z).

  antepassado(X, Z) :- gerou(X, Y), antepassado(Y, Z).

Características do Prolog

- **Regras - grafos de relacionamentos:** regras podem ser formuladas mais facilmente se desenharmos antes um grafo de relacionamentos. Nesse tipo de grafo, objetos são representados por nós e relacionamentos são representados por arcos. Além disso, o arco que representa a relação que está sendo definida deve ser pontilhado.




```
gerou(maria, jose).          gerou(joao, jose).
gerou(joao, ana).            gerou(jose, julia).
gerou(jose, iris).           gerou(iris, jorge).
masculino(joao).             masculino(jose).
masculino(jorge).
feminino(maria).             feminino(ana).
feminino(julia).             feminino(iris).

filho(Y, X) :- gerou(X,Y), masculino(Y).
mae(X,Y) :- gerou(X,Y), feminino(X).
avo(X,Z) :- gerou(X,Y), gerou(Y,Z), masculino(X).

irma(X,Y) :- gerou(Z,X), gerou(Z,Y), feminino(X), X \== Y.
                                  % \== significa X≠Y, simbólico
antepassado(X,Z) :- gerou(X,Z).
antepassado(X,Z) :- gerou(X,Y), antepassado(Y,Z).
```

Características do Prolog

- **Consultas**

  ? gerou(joao, iris). => true/false

  ? mae(iris, Y). => traz filhos de Iris em Y

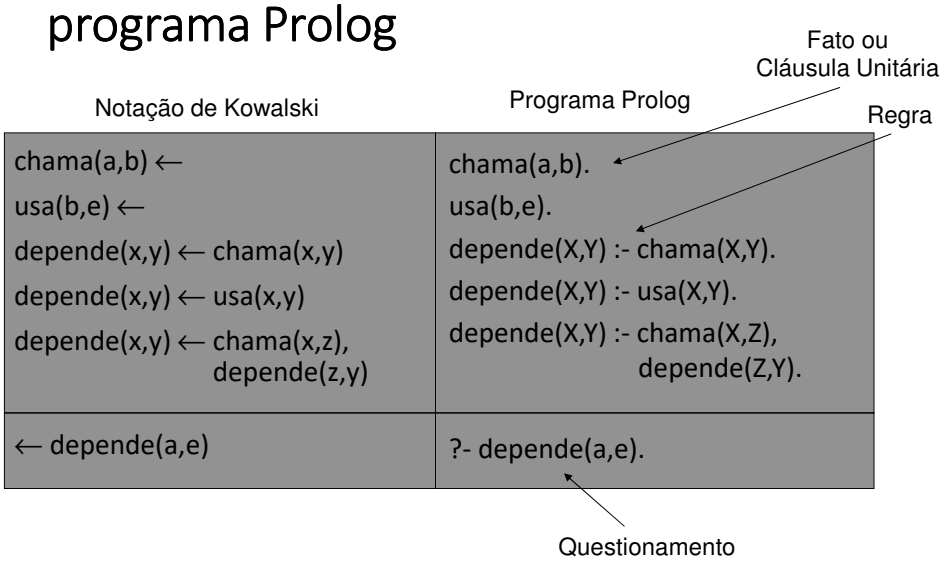
  ? mae(X, jose). => traz mãe do José em X

  ? irma(X, Y). => traz pares de irmãs em X,Y

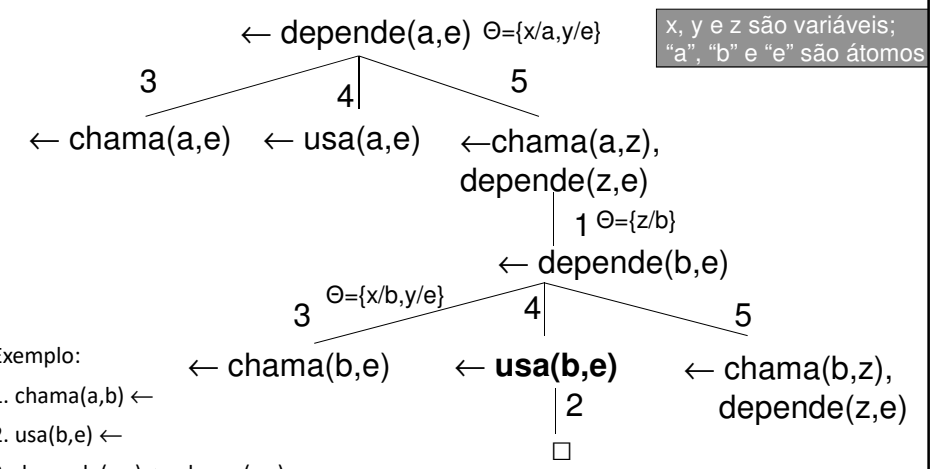
- **Novas Regras**

pai	filha	tio	tia
prima	primo	cunhado	sogra
sogro	descendente		

Da Notação de Kowalski para um programa Prolog



Matching e Substituição



Exemplo:

1. chama(a,b) ←
2. usa(b,e) ←
3. depende(x,y) ← chama(x,y)
4. depende(x,y) ← usa(x,y)
5. depende(x,y) ← chama(x,z), depende(z,y)
6. ← depende(a,e)

Matching e Substituição

O *matching* entre duas estruturas X e Y podem ocorrer em 4 situações

1. X é uma variável desinstanciada e Y está instanciada com um valor qualquer.

   Ex: ?- X = a.

2. X e Y estão instanciados com o mesmo valor.

   Ex.: ?- casa = casa.

Matching e Substituição

3. X e Y não estão instanciados. Neste caso X e Y passam a ser a mesma variável.

   Ex: ?- X = Y.

4. X e Y são estruturas com mesmo nome, número de parâmetros e uma das condições anteriores ocorre para todos os seus parâmetros.

   Ex.: ?- anda(joao, bicicleta(caloi)) = anda(joao, X).

Matching e Substituição

- Verifique o resultado dos *matchings*

  ?- pilota(A, londres) = pilota(londres,paris).

  ?- ponto(X,Y,Z) = ponto(X1,Y1,Z1).

  ?- letra(G) = palavra(letra).

  ?- num(alpha) = alpha.

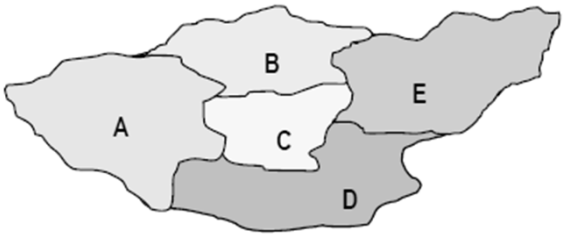
  ?- 'caixa' = caixa.

  ?- f(X,X) = f(a,b).

  ?- f(X,a(b,c)) = f(Z,a(Z,c))

Problemas: Exemplo 1 – colorir mapa.

Como colorir um mapa usando no máximo 4 cores, de modo que regiões adjacentes tenham cores distintas ?



Exemplo 1 – colorir mapa.
Solução:

Primeiramente, declara-se as cores que podem ser usadas na coloração por meio de *fatos.* Por exemplo, o fato **cor(azul)** estabelece que azul é uma das cores disponíveis.

Em seguida, declara-se que a tupla **(A,B,C,D,E)**, cujos componentes correspondem às regiões do mapa, é uma coloração válida se cada um de seus componentes é uma cor e se componentes representando regiões adjacentes no mapa têm valores distintos - isto é feito por meio de uma *regra*.

Exemplo 1 – colorir mapa.

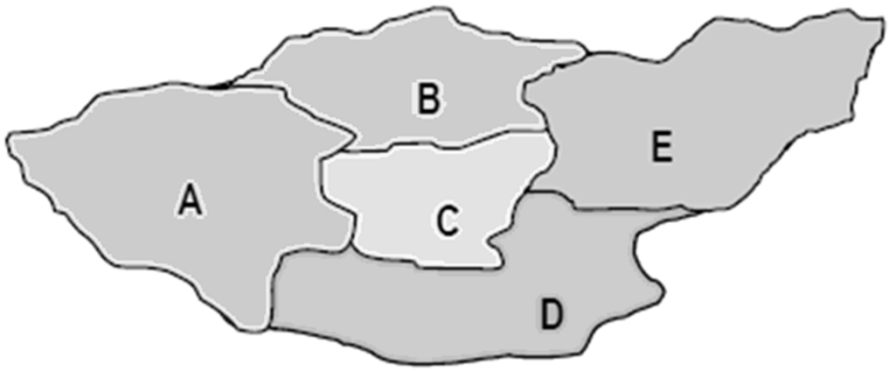
Implementação: % colorir.pl - colore um mapa usando no máximo quatro cores.

```
% cores disponíveis
cor(azul).
cor(verde).
cor(amarelo).
cor(vermelho).

% restrições para a solução
coloracao(A,B,C,D,E) :-
        cor(A), cor(B), cor(C), cor(D), cor(E),
        A\==B, A\==C, A\==D, B\==C, B\==E, C\==D, C\==E, D\==E.
```

Exemplo 1 – colorir mapa.

Implementação: SOLUÇÃO ALCANÇADA



Problemas: Exemplo 2 – geração de binários.

Gerar todos os números binários compostos por três dígitos.

Solução:

*Declarar que dígitos podem ser usados na composição de um número binário.

*Definir restrições sobre componentes de uma estrutura representando um número binário de três dígitos.

Implementação: % binario.pl -

```
% dígitos binários
digito(0).
digito(1).
% restrições para a solução
binario(N) :- N = (A,B,C), digito(A), digito(B), digito(C).
```

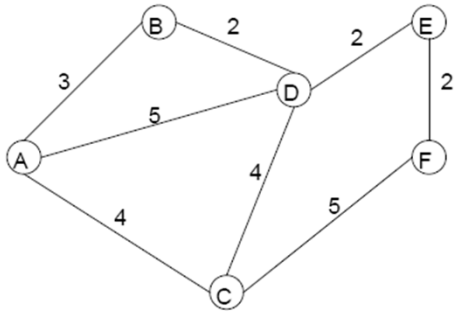
Para ver todas as respostas: ?- forall(binario(N), writeln(N)).

Exercícios

1- Implemente os seguintes programas Prolog

- soma(X,Y,Z) (Z é o resultado da soma de X e Y)
- maior(X,Y,Z) (Z é o maior entre X e Y)
- menor(X,Y,Z) (Z é o menor entre X e Y)
- par(X) (é verdade se X é par e falso se X é ímpar)
- impar(X) (é verdade se X é ímpar e falso se X é par)

Assuma que os arcos em um grafo expressem custos, como no exemplo abaixo:



e sejam descritos através de assertivas da forma

```
arco(R, S, T)
```

significando que há um arco de custo T entre os nodos R e S. Por exemplo, arco(A, B, 3) descreve um arco de custo 3 entre os nodos A e B. Assuma também que o relacionamento mais(X, Y, Z) vale quando X+Y=Z. Defina o relacionamento custo(U, V, L) de forma a expressar que existe um caminho de custo L entre os nodos U e V.

Exercício RC 7. Representação de Conhecimento – Alunos e Professores

Considere a seguinte base de factos exemplo:

```
aluno(joao, paradigmas).
aluno(maria, paradigmas).
aluno(joel, lab2).
aluno(joel, estruturas).
frequenta(joao, feup).
frequenta(maria, feup).
frequenta(joel, ist).
professor(carlos, paradigmas).
professor(ana_paula, estruturas).
professor(pedro, lab2).
funcionario(pedro, ist).
funcionario(ana_paula, feup).
funcionario(carlos, feup).
```

Escreva as seguintes regras em prolog:

a) Quem são os alunos do professor X?
b) Quem são as pessoas da universidade X? (alunos ou docentes)
c) Quem é colega de quem? Se aluno: é colega se for colega de disciplina ou colega de curso ou colega de universidade. Se professor: se for professor da mesma universidade.

Exercício RC 8. Representação de Conhecimento – Carros e Valores

Considere a seguinte base de factos exemplo:

```
comprou(joao, honda).
ano(honda, 1997).
comprou(joao, uno).
ano(uno, 1998).
valor(honda, 20000).
valor(uno, 7000).
```

a) Crie uma regra pode_vender onde o primeiro argumento é a pessoa, o segundo o carro e o terceiro é o ano actual (não especificar "homem" ou "carro" nas regras), onde a pessoa só pode vender o carro se o carro for comprado por ela nos últimos 10 anos e se seu valor for menor do que 10000 Euros.

Atualize as datas dos carros para:

ano(honda,2014)

ano(uno,2015)

Exercício – restaurante.

Sabendo-se que:
*há duas opções de **entrada** (salada ou pão)
*três opções de **prato** principal (peixe, carne ou massa)
*duas opções de **sobremesa** (sorvete, pudim)
Quais são todas as possíveis refeições completas que podem ser formadas?

```
entrada(salada).
entrada(pão).
prato(peixe).
prato(carne).
prato(massa).
sobremesa(sorvete).
sobremesa(pudim).
refeicao(R) :- ... % complete esta regra !
```