

PLP

PROGRAMAÇÃO LÓGICA

Prof. Salvador

Programação lógica

- A programação lógica (ou declarativa) surgiu como um paradigma nos anos 70.
- O programador deve declarar objetivos de computação e não algoritmos.
- Os objetivos são expressos como um coleção de asserções, ou regras.

Programação lógica

- Um programa lógico expressa as especificações para soluções de problemas com o uso de expressões em lógica matemática.
- A lógica proposicional e predicativa proporciona os fundamentos formais para a programação lógica.

PROLOG

- A principal linguagem de programação em lógica é PROLOG.
- PROLOG = **PRO**gramming in **LOGic**
- A sintaxe da linguagem Prolog é baseada em cláusula de Horn.

Linguagem Prolog

- A principal utilização da linguagem Prolog reside no domínio da programação simbólica, não-numérica, sendo especialmente adequada à solução de problemas envolvendo objetos e relações entre objetos.

Linguagem Prolog

- 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;

Linguagem Prolog

- Prolog:
 - Suporta código recursivo e iterativo para a descrição de processos e problemas, dispensando os mecanismos tradicionais de controle, tais como while, repeat, etc;
 - Representa programas e dados através do mesmo formalismo.

Linguagem Prolog

- Programação lógica
 - Lógica para definição do que deve ser solucionado
 - Controle para obter as soluções
- Prolog se baseia no cálculo de predicados
- Sintaxe e semântica bem especificadas e resolução consistente
- A tarefa do programador Prolog é especificar o componente lógico do algoritmo
- O controle da execução é exercido por um sistema de inferência inerente à linguagem lógica

Linguagem Prolog

- Programar em Prolog envolve:
 - declarar alguns *fatos* a respeito de objetos e seus relacionamentos,
 - fazer *perguntas* sobre os objetos e seus relacionamentos e
 - definir algumas *regras* sobre os objetos e seus relacionamentos.

Linguagem Prolog

- **Fatos**

- Fatos são expressos em Prolog através de relações.
 - Ex.: João é pai de José.

Onde:

- João e José são objetos
- Pai é o relacionamento entre os objetos.
- Os nomes dos relacionamentos e objetos devem começar por letra minúscula.
- O relacionamento é escrito primeiro e os objetos entre parênteses e separados por vírgula.
- Terminar com ponto.

Assim, em PROLOG: **pai(joão,josé).**

Linguagem Prolog

- Observar que:

`gosta(joão, maria) ≠ gosta(maria, joão).`

- O primeiro fato significa que João gosta de Maria e o segundo que Maria gosta de João.
- Os nomes dos objetos e relacionamentos são arbitrários: `a(b,c).` = `gosta(joão,maria).`
- É recomendável usar nomes significativos.

Linguagem Prolog

Relação: **predicado**

gosta(joão,maria).

Objetos: **argumentos**

Um predicado pode ter um número qualquer de argumentos, denominado **aridade**.

Notação: **gosta/2** = o predicado gosta tem dois argumentos.

Linguagem Prolog

Perguntas

Quando temos uma base de dados (que é, ao mesmo tempo um programa), podemos fazer perguntas ao sistema PROLOG sobre os fatos ali contidos.

Um sistema PROLOG responde com **true** ou **false** às perguntas.

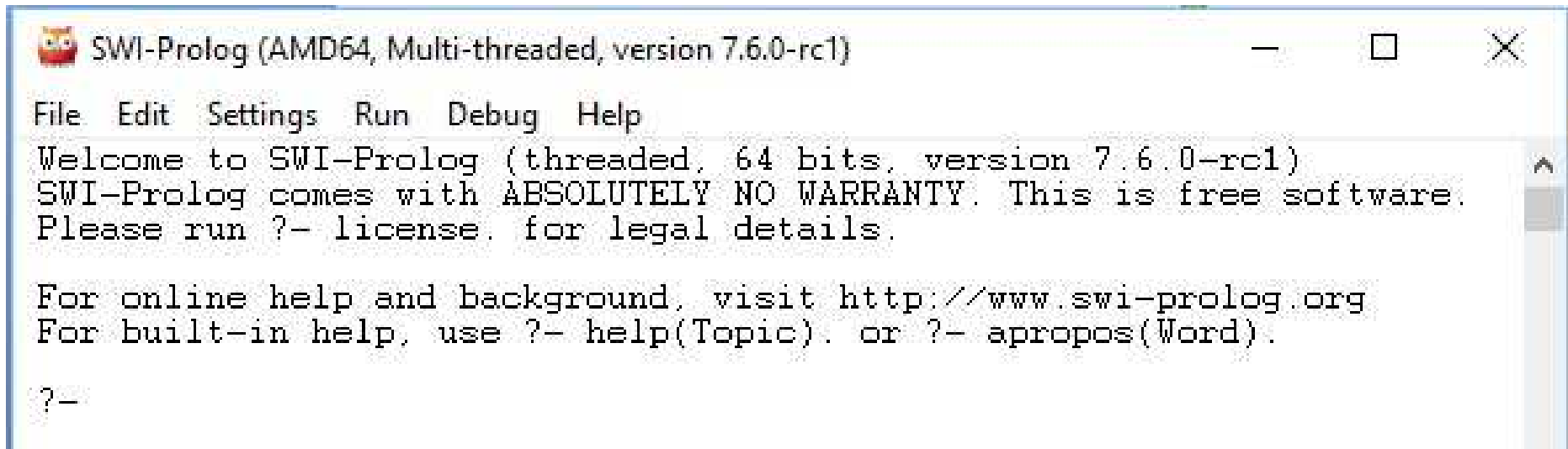
false significa que não foi possível provar.

Interpretador Prolog

- Um interpretador PROLOG bastante usado é o SWI-PROLOG.
- A página <http://www.swi-prolog.org/> disponibiliza tutoriais, manuais e diversas outras informações sobre o PROLOG.
- O interpretador pode ser encontrado em <http://www.swi-prolog.org/download/stable>

Interpretador PROLOG

- A tela inicial do programa é



```
SWI-Prolog (AMD64, Multi-threaded, version 7.6.0-rc1)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 64 bits, version 7.6.0-rc1)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?-
```

onde o prompt de comando está representado por ?-

Interpretador PROLOG

- Menu FILE
 - Consult – abre arquivo para execução.
 - Edit – abre arquivo para edição
 - New – cria arquivo.
 - Reload modified files – atualiza o arquivo que está aberto para execução e foi editado.

Interpretador PROLOG

- Menu HELP
 - **XPCE GUI Manual** – abrirá uma janela com sete menus.
 - **File > Demo programs** – dá acesso a alguns exemplos de programas em PROLOG.
 - Estão disponibilizados, também, os arquivos fontes.
 - Em **Browsers** está disponível um manual.

Linguagem Prolog

Perguntas (continuação)

Ex.:

`gosta(joão,peixe).`

`gosta(joão,maria).`

`gosta(maria,livro).`

`gosta(josé,livro).`

Perguntas:

`?- gosta(joão,carne).`

`false.`

`?- gosta(maria,livro).`

`true.`

`?- pai(joão,maria).`

`false.`

Obs.: Em um sistema PROLOG, o prompt de comando é indicado por `?-`

Linguagem Prolog

Variáveis

Permitem que se faça perguntas do tipo:

João gosta de que? ou João gosta de algo?

Variáveis iniciam por letra maiúscula ou _.

Ex.: X, Resultado, Nome_da_escola, _var001, _138.

A variável _ é chamada de variável anônima.

Uma variável pode estar:

- Instanciada: quando referencia um objeto.
- Não instanciada: quando ainda não referencia nenhum objeto.

Linguagem Prolog

- Variáveis:

Devem começar por letra maiúscula ou por _
(underline). Exemplos válidos:

X, Resultado, Nome_da_escola, _var001, _138
_ (variável anônima).

Linguagem Prolog

- Quando uma variável é usada em uma pergunta, Prolog **procura por todos os fatos** tentando encontrar um objeto com o qual a variável possa ser instanciada.
- Quando uma solução é encontrada, ela é mostrada
- Se o usuário estiver satisfeito com a resposta, basta digitar “.” ou *enter*.
- Se desejar mais respostas, usa-se ponto-e-vírgula.

- Exemplo:

?- gosta(joão,X).

X = peixe;

X = maria.

sistema aguarda ;

Resposta seguida de . = encerrou.

Linguagem Prolog

Conjunções

Permitem expressar relacionamentos mais complexos.

Ex.: “João gosta de Maria” e “Maria gosta de João”

Os dois fatos podem ser expressos em conjunto.

A conjunção **e** é representada por **,**

`gosta(joão,maria), gosta(maria,joão).`

Linguagem Prolog

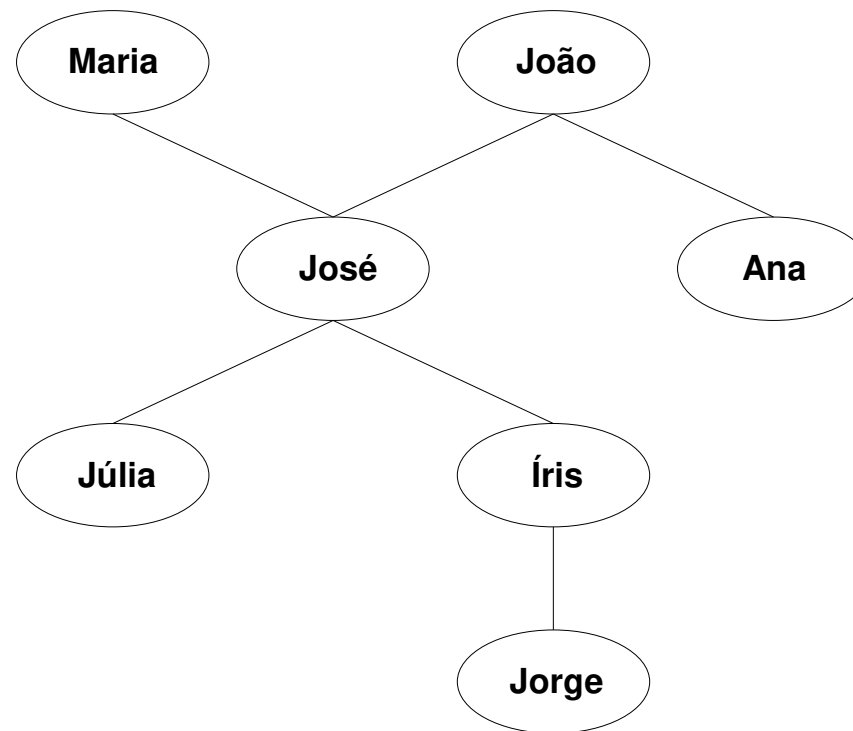
A conjunção ou é representada por ;

Ex. “João gosta de Maria” ou “Maria gosta de João”

`gosta(joão,maria); gosta(maria,joão).`

Linguagem Prolog

- Considera a figura abaixo, que representa uma árvore genealógica, estabeleça as relações progenitor.



Linguagem Prolog

- Relações do exemplo:

progenitor(maria, josé).

progenitor(joão, josé).

progenitor(joão, ana).

progenitor(josé, júlia).

progenitor(josé, íris).

progenitor(íris, jorge).

As relações constituem um programa, composto de seis cláusulas, cada uma das quais denota um fato acerca da relação progenitor.

Linguagem Prolog

- Pode-se formular questões menos simples, como "Quem são os avós de Jorge?". Como nosso programa não possui diretamente a relação avô, esta consulta precisa ser dividida em duas etapas, a saber:
 - (1) Quem é progenitor de Jorge? (Por exemplo, Y) e
 - (2) Quem é progenitor de Y? (Por exemplo, X)

Linguagem Prolog

- Assim, teríamos a pergunta:

progenitor(Y,jorge), progenitor(X,Y).

Outra situação:

gosta(joão,maria).

gosta(joão,josé).

gosta(joão,ingrid).

Como expressar o fato de que João gosta de todas as pessoas?

Uma forma de simplificar as duas situações acima é criar **regras**.

Linguagem Prolog

- **Regra**

- Uma regra, em Prolog, é a implementação de uma declaração condicional da Lógica Proposicional.

Uma declaração condicional é do tipo:

Se **A** então **B**,

onde A é o antecedente e B o conseqüente.

- Em Prolog, a regra é implementada na forma:
consequente se antecedente.

Linguagem Prolog

- Exemplo:

Se duas pessoas X e Y possuem os mesmos pais, então são irmãs.

Reescrevendo:

Duas pessoas X e Y são irmãs se possuem os mesmos pais.

Linguagem Prolog

- Relação irmã:

Para todo X e Y

X é irmã de Y se X e Y possuem um progenitor comum e

X é do sexo feminino.

Em Prolog:

irmã(X,Y) :-

progenitor(Z,X),

progenitor(Z,Y),

feminino(X).

Linguagem Prolog

- Perguntando ao sistema:

-?irmã(X,íris).

X = júlia;

X = íris

Respondendo que íris é irmã dela própria.

Todavia, essa é uma resposta lógica.

Para evitar esse tipo de resposta, é necessário ampliar a regra, especificando que X e Y devem ser diferentes.

Linguagem Prolog

A regra para a relação irmã fica:

irmã(X,Y) :-

 progenitor(Z,X),

 progenitor(Z,Y),

 feminino(X),

 diferente(X,Y).

Linguagem Prolog

- Recursão

Uma solução recursiva consiste em dividir o problema inicial em subproblemas menores e exatamente iguais ao problema inicial, até obter um subproblema simples, cuja solução é conhecida.

A partir da solução do problema simples, resolver o problema menos simples, sucessivamente, até chegar à solução final do problema inicial.

Linguagem Prolog

- Uma regra para definir os antepassados diretos:

Para todo X e Z , X é antepassado de Z se X é progenitor de z .

Em Prolog:

```
antepassado(X,Z) :- progenitor(X,Z).
```

Linguagem Prolog

No entanto, para definir um segundo nível de antepassado:

```
antepassado(X,Z) :-  
    progenitor(X,Y), progenitor(Y,Z).
```

Mais um nível:

```
antepassado(X,Z) :-  
    progenitor(X,Y1),  
    progenitor(Y1,Y2),  
    progenitor(Y2,Z).
```

E assim, sucessivamente.

Inconveniente: só permite um número limitado de relações.

Linguagem Prolog

- Uma solução recursiva:
 - Não impõe limite
 - É mais elegante.

Reescrevendo:

Para todo X e Z

X é antepassado de Z se existe um Y tal que

X é progenitor de Y e

Y é antepassado de Z.

Em Prolog:

`antepassado(X,Z) :- progenitor (X,Y), antepassado(Y,Z).`

Linguagem Prolog

- Mas é preciso acrescentar a regra que dá os antepassados diretos. É essa regra que vai determinar o retorno da recursão.
 - Assim, temos:

```
antepassado(X,Z) :-  
    progenitor(X,Z).                % antepassado direto  
antepassado(X,Z) :-  
    progenitor (X,Y),  
    antepassado(Y,Z).                % antepassado indireto
```

Linguagem Prolog

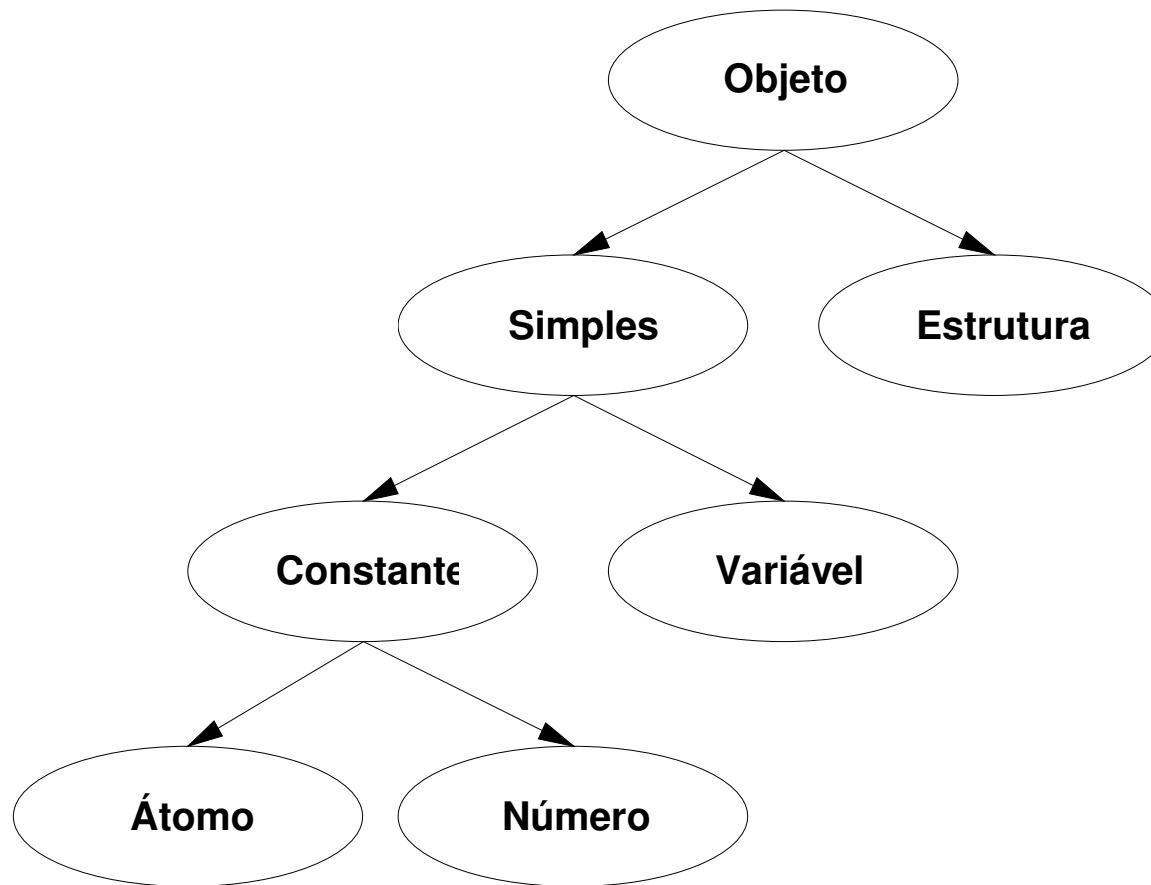
- Formule um programa em Prolog que contenha as seguintes informações:
 - Urso, peixe, peixinho, guaxinin, raposa, coelho, veado e lince são animais.
 - Grama e alga são plantas.
 - Urso come peixe, guaxinin, raposa e veado.
 - Peixe come peixinho
 - Peixinho come alga
 - Guaxinin come peixe
 - raposa come coelho
 - coelho come grama
 - veado come grama e
 - lince come veado.
- Acrescente as seguintes regras:
 - Se x come y e y é animal, então y é presa.
 - Formule uma regra para definir o predicado predador.
 - Formule uma regra para responder que animal é caçado.
 - Formule uma regra para responder o que come um animal “**na cadeia alimentar**”.

Linguagem Prolog

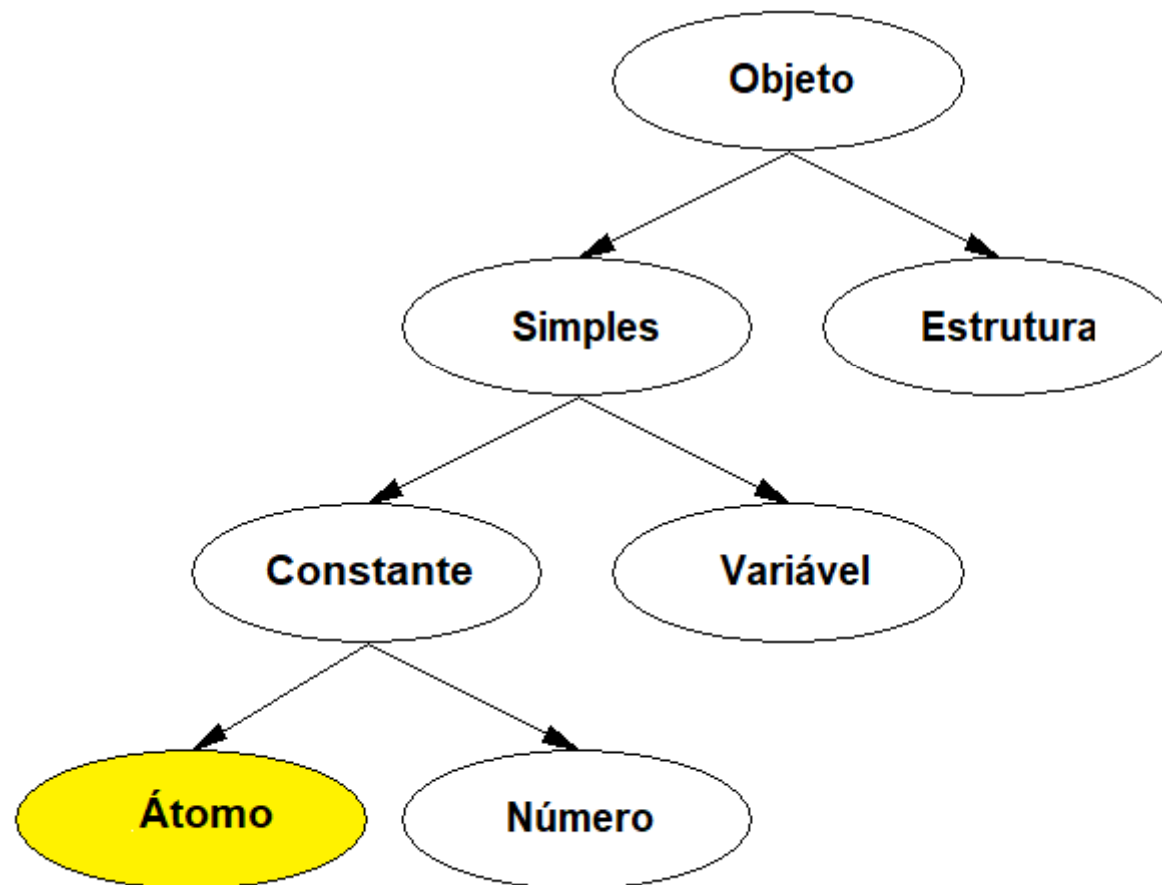
- Alfabeto da linguagem:
 - Pontuação () . ‘ “
 - Conectivos , (conjunção - e)
; (disjunção - ou)
:- (se)
 - Letras: a, b, c, ..., Z, A, B, ... , Z
 - Dígitos: 0, 1, ..., 9
 - Especiais: + - * / < > = : _ entre outros

Linguagem Prolog

- Classificação dos obietos

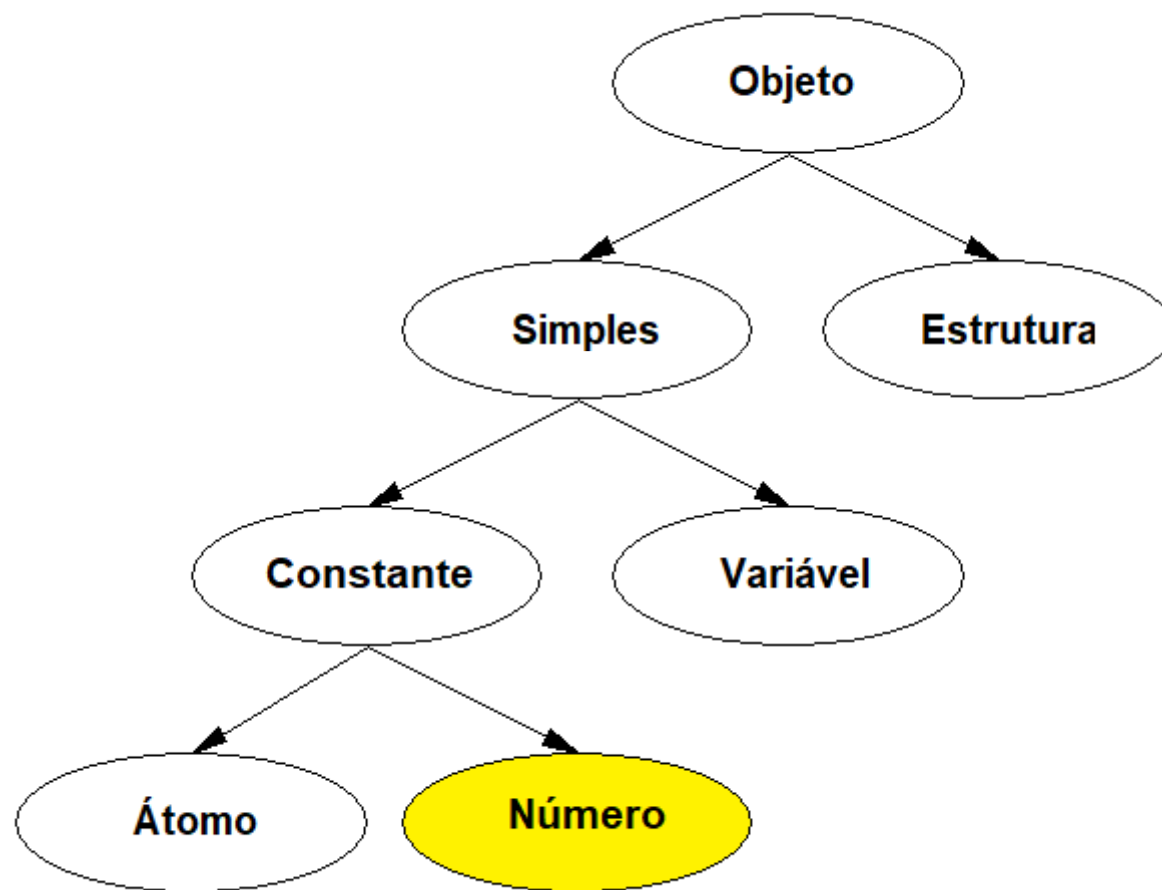


- **Átomo**



- Átomos
- São cadeias compostas por letras maiúsculas, minúsculas, dígitos numéricos e caracteres especiais.
- Podem ser construídos de três maneiras:
 - Cadeias de letras, dígitos e o caractere “_”
 socrates x_y nove casa a_12_tres
 - Cadeias de caracteres especiais
 <-----> ::= =/= =====> ++++++
 - Cadeias de caracteres quaisquer
 ‘representação do conhecimento’
 ‘Um nome ou frase qualquer’

- **Números**



- Números – podem ser inteiros ou reais.

Operadores Aritméticos Predicado pré-definido is	
Adição	+
Subtração	–
Multiplicação	*
Divisão	/
Divisão inteira	//
Resto div. inteira	mod
Potência	**
Atribuição	is

Operadores Relacionais	
$X > Y$	X maior que Y
$X < Y$	X menor que Y
$X \geq Y$	X maior ou igual a Y
$X \leq Y$	X menor ou igual a Y
$X := Y$	X igual a Y
$X = Y$	X unifica com Y
$X \neq Y$	X diferente de Y

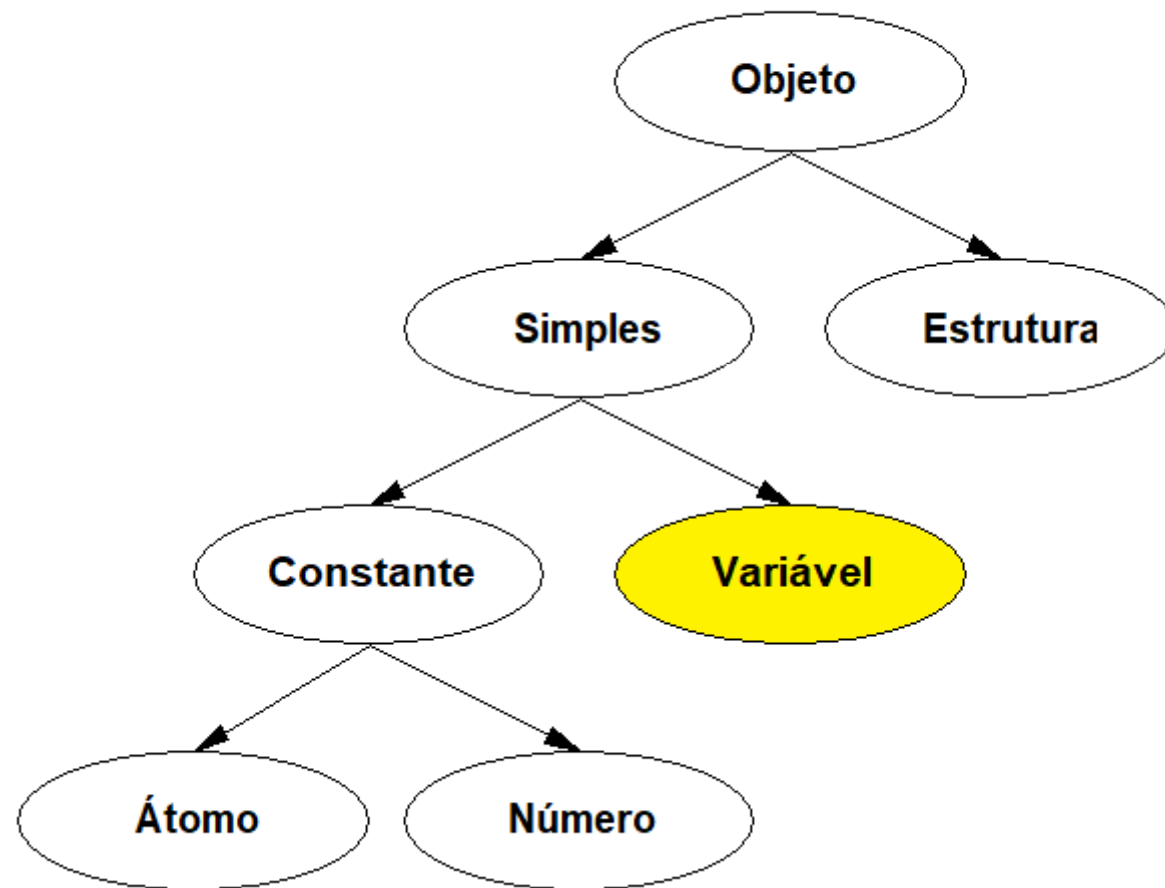
- Exemplos:

?- X is 3+2.

?- X is (5 mod 2).

?- 3 + 2 is X.

- Variáveis



- Variáveis

- Devem começar por letra maiúscula ou com o caractere _

N

Xis

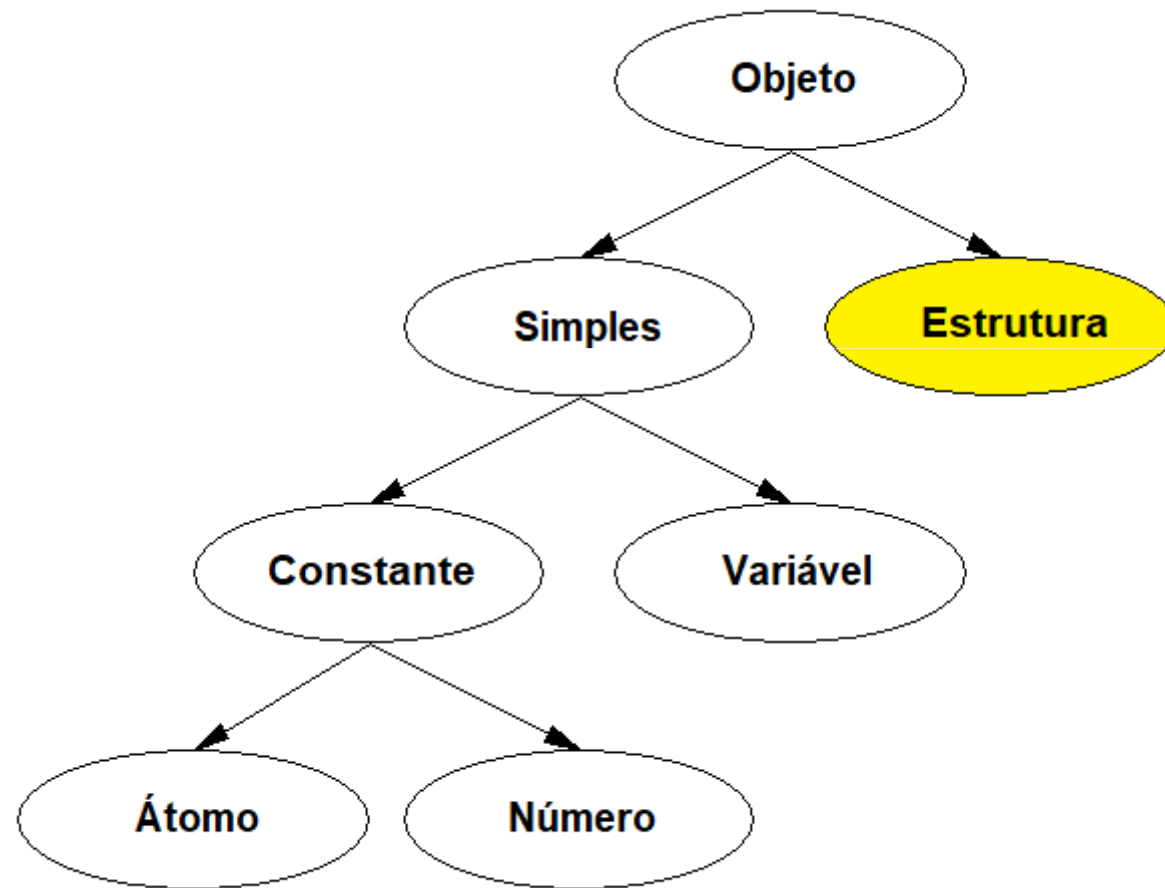
_var1

_345

A324Bis

_ (variável anônima)

- Estruturas



- Estruturas

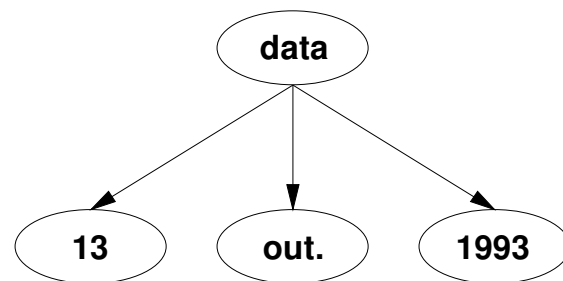
- Objetos que possuem vários componentes.

- Exemplo: uma data é composta de dia, mês e ano

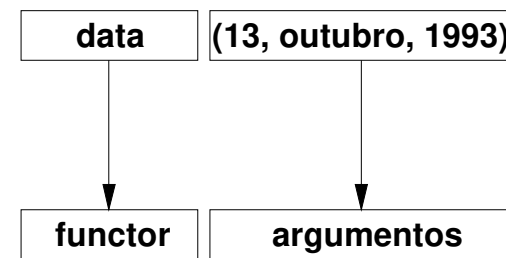
- São tratadas como objeto simples

- A combinação de diversos objetos em um é feita por um **functor** (símbolo funcional, nome de função)

- Exemplo: data(13, outubro, 1993)



(a)



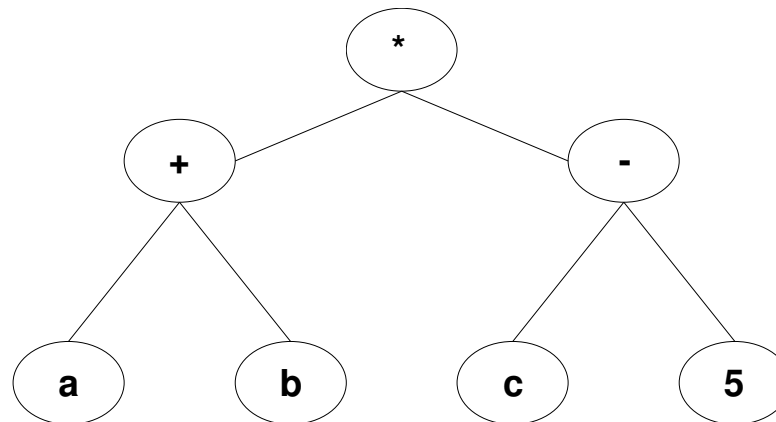
(b)

- Para representar um dia qualquer de março de 1993:
 - `data(Dia, março, 1996)`

Observar que Dia é uma variável, logo pode ser instanciada para qualquer objeto durante a execução.

- Todos os objetos estruturados podem ser representados em árvore

Exemplo: $(a + b) * (c - 5)$



Também pode ser escrita: $*(+(a,b),-(c,5))$

Ou seja, os símbolos $*$, $/$, $+$ e $-$ podem ser tomados como *functor*

A linguagem Prolog admite a forma prefixada e infixada.

- Unificação

É a operação mais importante entre dois termos Prolog.

Dados dois termos, diz-se que eles unificam se:

- São idênticos
- As variáveis de ambos os termos podem ser instanciados como objetos, de modo que os termos se tornem idênticos.

Exemplo de unificação:

Dados os termos

$\text{data}(D, M, 1994)$ e $\text{data}(X, \text{março}, A)$

Uma instanciação que torna os dois termos idênticos é:

D é instanciada com X;

M é instanciada com março;

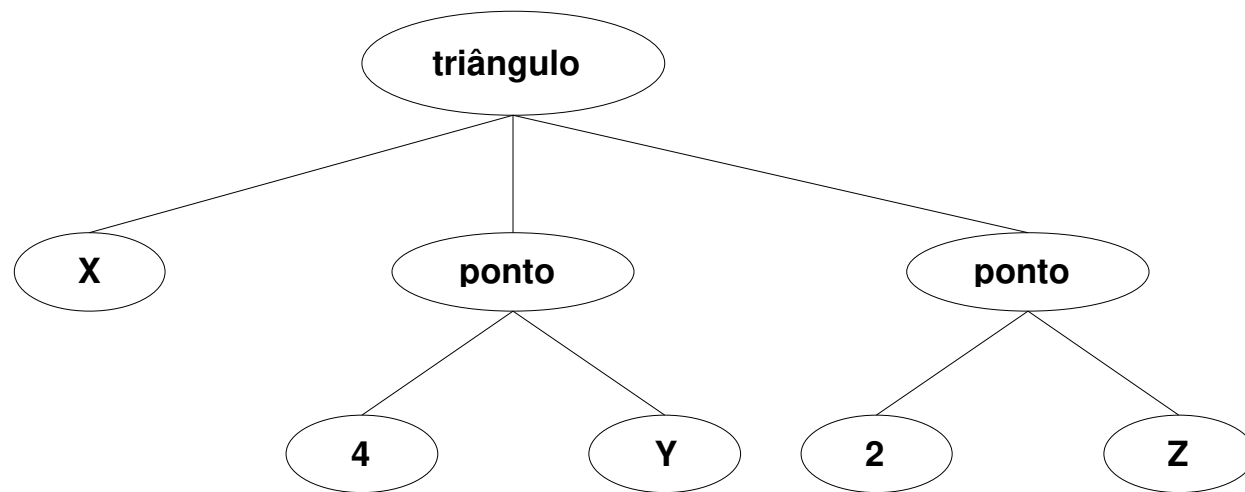
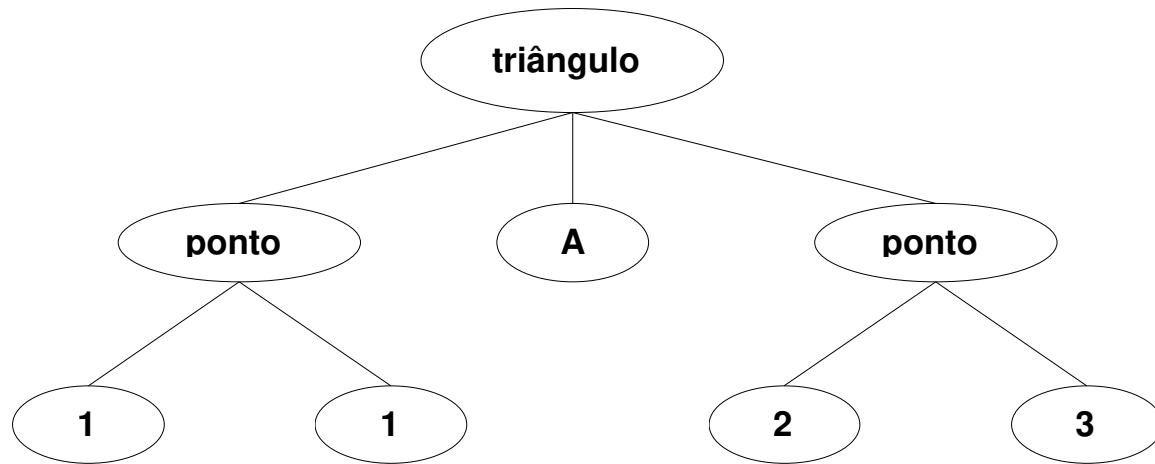
A é instanciada com 1994.

Não unificam: $\text{data}(D, M, 1994)$ e $\text{data}(D, M, 94)$

$\text{data}(X, Y, Z)$ e $\text{ponto}(X, Y, Z)$

- Regras que determinam se dois termos S e T unificam:
 - Se S e T são constantes, unificam se representam o mesmo objeto.
 - Se S é variável e T qualquer coisa, unificam com S instanciada com T . Se T é variável, é instanciada com S .
 - Se S e T são estruturas, unificam somente se:
 - S e T têm o mesmo *functor* principal
 - Todos os seus componentes correspondentes unificam

- Exemplo:



- Exercícios:

Quais dos seguintes objetos estão sintaticamente corretos e a que tipo de objeto pertencem?

- a. Daniela
- b. daniela
- c. 'Daniela'
- d. _daniela
- e. 'Daniela vai a Paris'
- f. vai(daniela, paris)
- g. 8118
- h. 2(X, Y)
- i. +(sul, oeste)
- j. três(Cavalos(Baios))

- Quais das próximas operações de unificação serão bem sucedidas e quais irão falhar?

Para as que forem bem sucedidas, quais são as instanciações de variáveis resultantes?

- a. $\text{ponto}(A, B) = \text{ponto}(1, 2)$
- b. $\text{ponto}(A, B) = \text{ponto}(X, Y, Z)$
- c. $\text{mais}(2, 2) = 4$
- d. $+(2, D) = +(E, 2)$
- e. $t(p(-1, 0), P2, P3) = t(P1, p(1, 0), p(0, Y))$

- Considere a expressão:

$$2+a * b-c$$

Ao avaliar essa expressão, o sistema considerará a precedência dos operadores, sendo que $*$ tem maior prioridade que $+$.

Equivale a escrevê-la da forma:

$$-(+(2,*(a,b)),c)$$

Para dar outra prioridade de avaliação (executar a soma $a + b$ primeiro, por exemplo), é necessário explicitar através do uso de parênteses.

$$*(+(2,a),-(b,c))$$

Embora não exista o comando de atribuição, como nas linguagens imperativas, é possível instanciar uma variável com o valor de uma expressão, usando o operador especial ***is***.

Exemplo: $A \text{ is } 2 + 2$.

Operadores dão legibilidade a um programa.
Em Prolog é possível especificar operadores.

A definição de um operador é feita por uma cláusula especial denominada *diretiva*.

Exemplo:

`:- op(1200, xfx, ':-').`

Essa é a definição do operador 'se'.

1200 – é a prioridade (a maior prioridade é um).

xfx significa que o operador f (no caso ':-') será colocado entre dois argumentos.

Há uma diferença entre x e y (quando for o caso).

x significa um argumento com prioridade menor que o operador.

y significa um argumento com prioridade menor ou igual ao operador.

Exemplo:

A expressão $a - b - c$ é entendida como $(a-b)-c$. Isso acontece porque o operador $-$ é definido como yfx.

Conjunto padrão de operadores da linguagem

`:- op(1200, xfx, ["', ':-']).`
`:- op(1200, fx, ["', ':-', '?!-']).`
`:- op(1100, xfy, ';').`
`:- op(1000, xfy, ',').`
`:- op(700, xfx, [is, =, <, >, =<, >=, ==, =\=, \==, =:=]).`
`:- op(500, yfx, [+,-]).`
`:- op(500, fx, [+,-,not]).`
`:- op(400, yfx, [*,/,div]).`
`:- op(300, xfx, mod).`
`:- op(200, xfy, ^).`

- Definindo novos operadores
 - Necessidade: processamento de uma classe particular de problemas, como expressões booleanas.
 - Definição dos conectivos:
 - $\text{:- op (800, xfx, <===>)}.$
 - $\text{:- op (750, xfy, \^{})}.$
 - $\text{:- op (700, xfy, \vee)}.$
 - $\text{:- op (650, fxy, \sim)}.$

Com essa definição, é possível expressar a lei de De Morgan em um programa:

$$\sim(A \wedge B) \iff \sim A \vee \sim B.$$

- Usando os operadores

A consulta ?-X = 1 + 2.

Obtém a resposta X = 1 + 2

Por ou lado: ?X is 1 + 2.

Resposta X = 3

Comparação: ?-100*50 > 1000
yes

Pesquisa na base de dados as pessoas que nasceram
em um determinado período: de 1990 a 2000

?- nasceu(Nome,Ano), Ano >= 1990, Ano <= 2000.

Exemplo de um fatorial recursivo em C:

```
int fatorial (int n)
{
  if(n==0)
    return 1;
  else
    return fatorial(n-1) * n;
}
```

Em PROLOG:

```
fat(0, 1).
fat(N, R) :-
  N1 is N - 1,
  fat(N1, A),
  R is N * A.
```

Exercício

- 1 Dadas as distâncias dos planetas do sistema solar ao Sol, solicita-se um programa que, dados dois planetas, retorna a distância entre eles: $\text{dist}(P1, P2, D)$.

Planeta	Distância
Mercúrio	36
Vênus	67
Terra	93
Marte	141
Júpiter	484
Saturno	886
Urano	1790
Netuno	2900

Exercício

- 2 O máximo divisor comum de dois números pode ser obtido por divisões sucessivas e por subtrações sucessivas.

Fazer um programa que recebe dois números e retorna o MDC desses dois números.

Solução 1: usando o método de subtrações sucessivas.

Exemplo: $\text{MDC}(324, 27)$

Dados dois inteiros positivos, X e Y , seu máximo divisor comum D pode ser encontrado segundo três casos distintos:

- (1) Se X e Y são iguais, então D é igual a X ;
- (2) Se $X < Y$ então D é igual ao mdc entre X e a diferença $X - Y$;
- (3) Se $X > Y$, então cai-se no mesmo caso (2), com X substituído por Y e vice-versa.

- As três cláusulas Prolog que expressam os três casos acima são:

mdc(X , X , X).

mdc(X , Y , D) :- $X < Y$,

$Y1$ is $Y - X$,

mdc(X , $Y1$, D).

mdc(X , Y , D) :- $X > Y$,

mdc(Y , X , D).

Listas

Uma notação usada para representar listas é empregando colchetes.

Notação: $[H|T]$

H – cabeça da lista

T – corpo da lista.

O corpo de uma lista é outra lista.

$[X|Y]$ ou $[X|[Y|Z]]$ unificam com $[a,b,c,d]$

?- $[X | Y] = [a, b, c]$.

$X=a$ $Y=[b, c]$

?- $[X | [Y | Z]] = [a, b, c, d]$.

$X=a$ $Y=b$ $Z=[c, d]$

São equivalentes:

$[a, b, c] == [a, b | [c]] == [a, b, c | []] == [a | [b | [c]]] == [a | [b, c]]$

- Operações sobre listas

Construção de listas:

Dados X e Y, a lista terá cabeça X e corpo Y.

Fato:

$\text{lista}(X, Y, [X \mid Y]).$

Fazendo a consulta:

$?\text{-lista}(a, b, Z).$

Qual a resposta?

$Z = [a \mid b]$

Outras consultas:

?-lista(a, [b, c], Z).

Z=[a, b, c]

?-lista(a, [], Z).

Z=[a]

?-lista(a, X, [a, b, c]).

X=[b, c]

?-lista([a, b, c], [d, e], Z).

Z=[[a, b, c], d, e]

- Ocorrência de um elemento em uma lista
Uma relação para expressar a ocorrência de um elemento em uma lista pode ser:

membro(X,L)

ou seja, X é membro de L se:

1 – X é a cabeça de L, ou

2 – X é membro do corpo de L.

Em Prolog:

membro(X, [X | C]).
membro(X, [Y | C]) :-
 membro(X, C).

Na primeira cláusula não interessa o corpo e na segunda a cabeça da lista. Assim, podemos ter:

membro(X, [X | _]).
membro(X, [_ | C]) :-
 membro(X, C).

- Tamanho da lista

Em Prolog:

tamanho([],0).

tamanho([_ |R],N) :-

tamanho(R,N1), N is N1 + 1.

?-tamanho([a,b,c,d],T).

T = 4

- Concatenação de listas

A concatenação de duas listas L1, L2, resultando numa terceira L3 é definida pela relação:

$\text{conc}(L1, L2, L3).$

Dois casos a considerar:

(1) L1 é vazia: a lista resultante será igual à primeira.

$\text{conc}([], L, L)$

(2) L1 é não vazia: pode ser denotada por $[X | L1]$. Esta concatenada com L2 resultará em uma lista L3 com a cabeça de L1, o corpo de L1 e toda L2.

Em Prolog:

$\text{conc}([X | L1], L2, [X | L3]) \text{ :- conc}(L1, L2, L3).$

conc([], L, L).

conc([X | L1], L2, [X | L3]) :- conc(L1, L2, L3).

Realizando consultas:

?-conc([a, b, c], [1, 2, 3], L).

L=[a, b, c, 1, 2, 3]

?-conc([a, [b, c], d], [a, [], b], L).

L=[a, [b, c], d, a, [], b]

?conc(L1, L2, [a, b, c]).

L1=[]

L2=[a, b, c];

L1=[a]

L2=[b, c];

L1=[a, b]

L2=[c];

L1=[a, b, c]

L2=[];

Numa lista dos meses do ano, procurar os meses antes e depois de maio:

?-M=[jan, fev, mar, abr, mai, jun, jul, ago, set, out, nov, dez],
conc(Antes, [mai | Depois], M).

Antes=[jan, fev, mar, abr]

Depois=[jun, jul, ago, set, out, nov, dez]

- Remoção de elementos de uma lista
Remover X de uma lista L resultando na lista L1, pode ser programada:
 remove(X, L, L1).

Novamente, dois casos a considerar:

- (1) Se X é cabeça de L, então L1 é o seu corpo.
- (2) Se X está no corpo de L, então L1 é obtida removendo X desse corpo.

Em Prolog:

```
remove(X, [X | Z], Z).  
remove(X, [Y | Z], [Y | W]) :- remove(X, Z, W).
```

```
remove(X, [X | C], C).  
remove(X, [Y | C], [Y | D]) :- remove(X, C, D).
```

Consultas:

```
?- remove(b,[a,b,c],N).  
N = [a,c]
```

```
?- remove(a,[a,b,a,c],N).  
N = [b,a,c];  
N = [a,b,c];
```

A operação de inserir um elemento X em uma lista L, resultando na lista L1, pode ser definida a partir da operação remove:

```
insere(X, L, L1) :- remove(X, L1, L).
```

Sintaxe da função abs:

abs(-3,X).

X = 3.

Converter os valores de uma lista em seus valores absolutos.

Ex.: converte([-1,3,-5,-6],[1,3,5,6]).

converte([], []).

converte([X|L1], [Y|L2]) :-

converte(L1,L2),

abs(X,Y).

- Inversão de listas

Um processo mais intuitivo, embora menos eficiente, é denominado “inversão ingênua” e consiste em:

- Tomar o primeiro elemento da lista;
- inverter o restante;
- concatenar a lista formada pelo primeiro elemento ao inverso do restante.

Em Prolog:

```
inverter([ ], [ ]).
```

```
inverter([X | Y], Z) :-  
    inverter(Y, Y1),  
    conc(Y1, [X], Z).
```

- Sublistas

Uma lista S é sublista de uma lista L, se S ocorre em L.

sublista([c,d], [a,b,c,d,e]).

A relação sublista pode ser baseada na relação membro, sendo esta mais genérica:

S é sublista de L se:

- (1) L pode ser decomposta em duas listas, L1 e L2, e
- (2) L2 pode ser decomposta em S e L3.

Em Prolog:

sublista(S, L) :-

conc(L1, L2, L),

conc(S, L3, L2).

```
sublista(S, L) :-  
    conc(L1, L2, L),  
    conc(S, L3, L2).
```

```
?-sublista(S,[a,b,c]).
```

```
S=[];
```

```
S=[a];
```

```
S=[a,b];
```

```
S=[a,b,c];
```

```
S=[];
```

```
S=[b];
```

```
S=[b,c];
```

```
...
```

- Interação com o usuário

write - escreve na tela.

sintaxe: write(Variável)

write('string entre asas').

read – recebe valor do usuário e atribui a uma variável.

sintaxe: read(Variável).

nl – new line → leva o cursor para a linha seguinte.

Exemplo: escreve :- write('Digite algo'),

read(X),nl,

write('Voce digitou '), write(X).

- Considere a base de dados da árvore genealógica das primeira aulas.
- Fazer uma programa (regra) mae_de, para listar as mães e seus filhos, no formato:

Nome_da_mãe **é mãe de** nome_do_filho(a).

- Backtracking
 - Não precisa ser implementado. Inerente ao PROLOG.
 - Pode causar ineficiência em algum caso.
- O controle do backtracking pode tornar o programa mais rápido e ocupar menos memória.
- O operador ! é usado para controlar o backtracking.
- Utilizado quando as regras são mutuamente exclusivas.

- Exemplo:

extenso(1) :- write('Um'), !.

extenso(2) :- write('Dois') , !.

extenso(3) :- write(' Três') , !.

extenso(_) :- write('Fora de intervalo').