Linguagem Prolog
Conceitos Básicos

Turma A

Prof. Marcelo Ladeira
CIC/UnB

Slides adaptados a partir de slides obtidos na Internet e de slides criados pelo Prof. Wagner Teixeira da Silva

Estrutura de um Programa Prolog Termos

- Usados para construir programas e estruturas de dados
- Tipos
  - constantes
    - inteiros
    - reais
    - átomos
      - não são variáveis
        - podem ser vistos como strings constantes
  - variáveis
  - termos compostos

Estrutura de um Programa Prolog
Constantes

- Atomos e números são definidos sobre os caracteres:
  A,B,..,Z
  a,b,...,z
  0,1,...,9
  caracteres especiais como + - * / < > = : . & _ ~
- Atomos
  - strings de letras, dígitos e o underscore, iniciando com MINÚSCULA
    - anna x_25 nil
  - string de caracteres especiais
    - * . = @ # $
  - Átomos especiais
    - ::== .:. [] !
  - strings de caracteres entre aspas simples
    - 'Tom' 'x_>:'
- Números
  - reais: 3.14 -0.57 1.23 1.23^4
  - inteiros: 23 5753 -42

Estrutura de um Programa Prolog
Variáveis

- **Iniciam com maiúscula ou underscore**
  - **seguido de qualquer número de letras, digitos ou "_"**
    - **X_25_chica X Barao Fred**
  - **variável anônima (pense como "não importa!")**
    - **um simples underscore**

      haschild(X) :- parent(X,_).
    - **comportamento igual ou diferente ?**

      likes(mary,_), likes(_,mary).

      likes(mary,X), likes(X,mary).

Por quê?

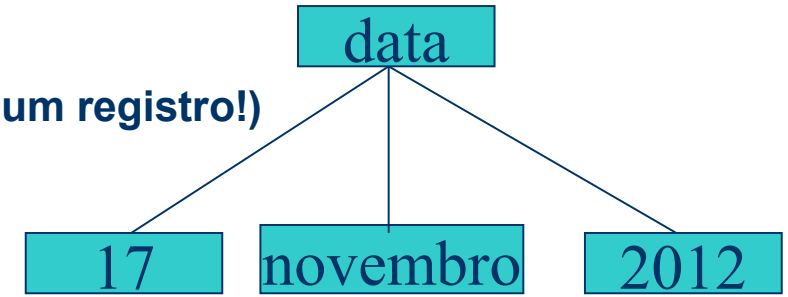
Estrutura de um Programa Prolog
Termo Composto

- Átomo seguido por uma seqüência de um ou mais termos, entre parenteses e separados por vírgula

  parent(pam,bob).

  deseja(estudante,passar(lp,2013,2)).

  - Denorex !
    - não é uma função
      - é estrutura de dados (tipo um registro!)



  - Exemplo
    - entradas via modo interpretador

      assert(data(17, novembro, 2012)).

      data(Dia, novembro, 2012).

      - Dia = 17

Estrutura de um Programa Prolog

- Cláusulas Prolog
  - Fatos, regras ou consultas
- Base de conhecimentos (BC)
  - fatos
  - regras
    - arquivo texto
      - carregado via “consult(arquivo-texto).”
- Prompt interpretador
  - consultas
    - BC é modificada via

      assert (insere fato ou regra)

      retract(elimina fato ou regra)

Estrutura de um Programa Prolog
Definição Formal

<fato> ::= <fa>. % fa significa fórmula atômica

<regra> ::= <fa_0> :- <fa_1>, ... , <fa_N>.

<consulta> ::= <fa_1>, ... , <fa_N>.

<fa> ::= <predicado>(<$termo_1$>, ... , <$termo_N$>) |
<op_préfixado> <$termo_1$> <$termo_2$> |
<$termo_1$> <op_infixado> <t_2> |
<$termo_1$> <$termo_2$> <op_pósfixado>

<predicado> ::= <átomo>

<termo> ::= <constante> | <variável> | <t_composto>

<constante> ::= <átomo> | <inteiro> | <número_real>

<t_composto> ::= <átomo>(<lista_termos>)

<lista_termos> ::= <termo>|<termo>,<lista_termos>

Construções Básicas

- **Fatos Universais**
  - Fatos com variáveis universalmente quantificadas.
  - Functor e átomo iniciam com letra minúscula.
- **Programa Prolog: Fatos e Regras**

```
mais(0,X,X).                    % fato
mais(A,B,C) :- C is A + B.      % regra
vezes(1,X,X).                   % fato
vezes(A,B,C) :- C is A * B.     % regra
```

- **Variável**
  - Está associada a um indivíduo não especificado, é uma incógnita de valor único. É local a uma sentença.

Usando o Prolog

- Para usar o Prolog você precisa saber:
  - Invocar o Prolog
  - Sair do Prolog:
    - ^Z
    - halt. (retorna ao chamador)
  - Usar um editor para editar um programa
  - Carregar um programa
    - load ou load_files(arquivo). % fonte ou objeto
    - consult(arquivo). % fonte
    - consult(user). % carrega via teclado. Fim ^Z
    - reconsult(arquivo). % recarrega arquivo

Usando o Prolog

- Capacidade especial (de algumas implementações)
  - Armazenar o atual BC em arquivo
    - save(arquivo). % salva a BC no arquivo, objeto
    - save_predicates(predicados, arquivo).
      % salva os predicados em arquivo, código
  - Recuperar o arquivo salvo
    - restore(arquivo). % recupera o código objeto
    - load ou load_files(arquivo). % objeto

Fato

- Relação verdadeira entre termos, via predicado.
  - Ex.: gosta(joao, cerveja).
    gosta(joao, maria).
- Fato é uma cláusula sem nenhuma condição.
- Sintaxe de predicado
  - <functor> (t_1,t_2,...,t_n).
    Predicado expressa uma relação entre termos.
- Aridade
  - Quantidade de termos no predicado. Expressa por <functor>/aridade.
    - Ex.: gosta/2 ou gosta/3
  - Predicados com mesmo functor podem ter diferentes aridades:
    gosta(joao, ler, livros).
    gosta(joao,maria).

Regra

- cabeça :- corpo.
  - o operador “:-” é denominado pescoço (neck)
- meta :- sm_1, sm_2,..., sm_k
  - meta só é verdadeira se suas submetas também o forem.
  - para provar que meta é verdadeira, deve-se provar antes que suas submetas também são.
  - meta e submetas são relações predicativas entre termos.
  - termos nomeiam objetos do discurso.

Consulta
Meio de Recuperar Informações em Prolog

- **Exemplo:**

```
pai(joao,mane).
pai(joao, ze).
pai(joao, quim).
pai(mane,maria).
→  pai(ze, zefa).
pai(ze, ruth).
irmas(A,B) :- pai(P,A),
              pai(P,B),  A\==B.
avo(A,N) :- pai(P,N), pai(A,P).
tio(T,S) :- pai(P,S), irmas(P,T).
```

```
?- pai(P,zefa).
    P = ze
?- irmas(quim,A).
    A = mane ;
    A = ze ;
    false.
?- tio(T,zefa).
    T = mane ;
    T = quim ;
    false.
?- avo(A,N).
    A = joao , N = maria ;
    A = joao , N = zefa ;
    A = joao , N = ruth ;
    false.
```

Consulta
Meio de Recuperar Informações em Prolog

- Responder a uma consulta é determinar se ela é uma conseqüência do programa (axiomas da teoria).
- Consulta existencial

```
?- mais(3,X,8).          % existe um X + 3 = 8?
?- pai(P,joao).          % existe um P pai de joao?
```

- Consulta conjuntiva e variáveis cotizadas

```
? pai(joao, F), pai(F,N).    %Quem são os netos de joão?
? tio(T,P), pai(P,maria).    % Quem é o tio avó de Maria?
```

Predicados Bidirecionais

- **Exemplo:**

```
pai(joao,mane).
pai(joao, ze).
pai(joao, quim).
pai(mane,maria).
→       pai(ze, zefa).
pai(ze, ruth).
irmas(A,B) :- pai(P,A),
                pai(P,B),
   A\==B.
avo(A,N) :- pai(P,N), pai(A,P).
tio(T,S) :- pai(P,S),
   irmas(P,T).
```

```
?- pai(P,zefa).
      P = ze
?- pai(ze, F)
      F = zefa ;
      F=ruth;
      false.
?- tio(T,zefa).
      T = mane ;
      T = quim ;
      false.
?- avo(A,N).
      A = joao , N = maria ;
      A = joao , N = zefa ;
      A = joao , N = ruth ;
      false.
```

Predicados Bidirecionais

```
concat([],Ys,Ys).                                  % fato
concat([X|Xs],Ys,[X|Zs]) :- concat(Xs,Ys,Zs).      % regra

?- concat([1,2,3,4], [a,b,c,d], Rs).
      Rs = [1,2,3,4,a,b,c,d]
?- concat(Xs, Ys, [1,2,3,4,a,b,c,d]).
      Xs = [] , Ys = [1,2,3,4,a,b,c,d] ;
      Xs = [1] , Ys = [2,3,4,a,b,c,d] ;
      Xs = [1,2] , Ys = [3,4,a,b,c,d] ;
      Xs = [1,2,3] , Ys = [4,a,b,c,d] ;
      Xs = [1,2,3,4] , Ys = [a,b,c,d] ;
      Xs = [1,2,3,4,a] , Ys = [b,c,d] ;
      Xs = [1,2,3,4,a,b] , Ys = [c,d]
?- concat(Xs, [a,b,c,d], [1,2,3,4,a,b,c,d]).
      Xs = [1,2,3,4] ;
      false.
```

Relações

```
membro(X,[X|Xs]).                    % fato
membro(X,[Y|Xs]) :- membro(X,Xs).    % regra

?- membro(4, [1,2,3,4,5,6]).
      true.
?- membro(x, [1,2,3,4,5,6]).
      false.
?- membro(X, [1,2,3,4,5,6]).
      X = 1 ;
      X = 2 ;
      X = 3 ;
      X = 4 ;
      X = 5 ;
      X = 6 ;
      false.
```

Predicado Unidirecional

```
fat(0,1):-!.
fat(1,1):-!.
fat(N,F) :- N1 is N - 1, fat(N1,F1), F is N * F1.
```

?- fat(0,F).

F = 1

?- fat(5,F).

F = 120

?- fat(5,120).

true.

?- fat(5,100).

false.

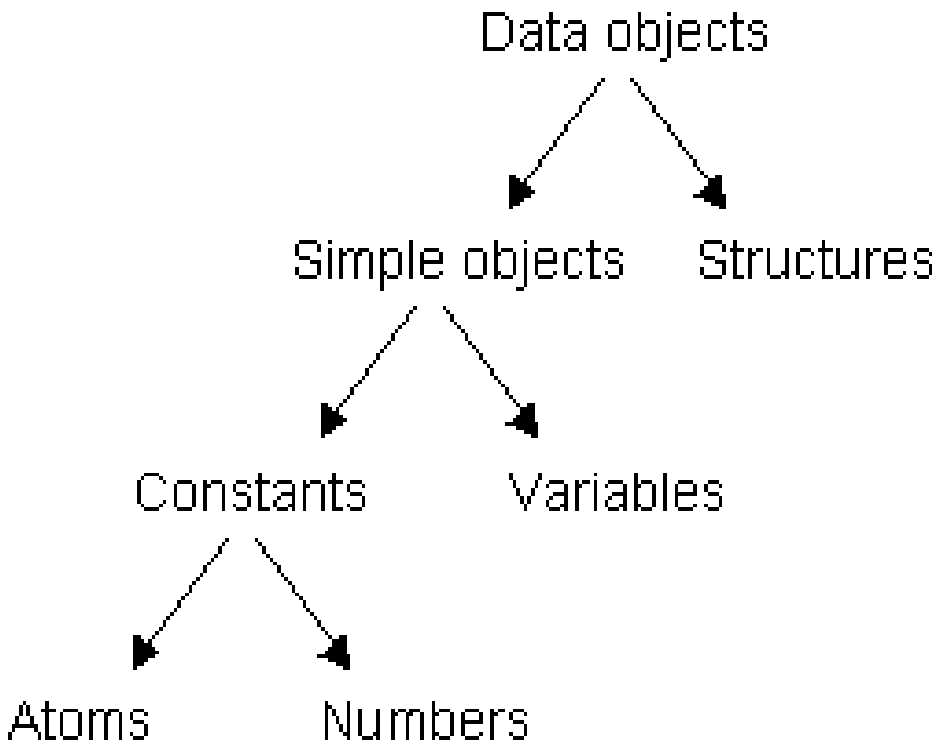
?- fat(N,120).

ERROR: is/2: Arguments are not sufficiently instantiated

Predicado Unidirecional

```
fib(N,F) :- fibx(N,1,1,F), !.
fibx(0,A,_,A).
fibx(N,A,B,F) :- N1 is N - 1, AB is A + B, fibx(N1,B,AB,F).

?- fib(0,F).
      F = 1
?- fib(1,1).
      true.
?- fib(10,F).
      F = 89
?- fib(20,F).
      F = 10946
?- fib(100,F).
      F =  573147844013817084101.
?- fib(N,1).
      N = 0
?- fib(N,89).
      ERROR: Arguments are not sufficiently instantiated
```

Termos (Data objects)



Termo

- Nomeia entidades do universo do discurso.
- <termo> ::= <simples>|<termo composto>
- <simples> ::= <constante>|<variável>
- <variável> ::= _ |<maiúscula>|<variável><letra>|
  <variável><dígito>| <variável>_
- variável
  Ex.: X _ _x Avo Gente G12 Pai_sangue _b
  - É uma incógnita. Designa uma entidade até o momento desconhecida.

Termo
Não-Variável

- <constante> ::= <átomo> | <número>
  - Há o predicado *atomic(Termo)* que retorna *true* quando o Termo é atômico, não estruturado.
- <número> ::= <inteiro> | <real>
  Ex.: 4, -10, 5.3, 0.123E-10, -1.32e102
  - pode ser reconhecido por *number(X)*.
  - <operadores-relacionais> ::= < | = | =< | >= | >
  - <aritmética> ::= <aritmética-genérica> | <aritmética bitwise>

Aritmética em Prolog

- **Exemplos**

  X is 1+2.

  X = 3.

  X is ceiling(2.1).

  X = 3.

  X is ceiling(-2.1).

  X = -2.

Operadores Aritméticos

Nome/aridade	Explicação
abs / 1	valor absoluto (ISO)
+ / 2	adição (ISO)
acos / 1	arco cosseno
asin / 1	arco seno
atan / 1	arco tangente (ISO)
/\ / 2	and bit a bit (ISO)
\ / 1	bit complement (ISO)
<< / 2	shift bit a bit para a esquerda (ISO)
\/ / 2	or bit a bit (ISO)
>> / 2	shift bit a bit para a direita (ISO)
ceiling / 1	menor inteiro não menor que (ISO)
cos / 1	cosseno (ISO)
cosh / 1	cosseno hiperbólico

Operadores Aritméticos

Nome/aridade	Explicação
e / 0	número 2.71828. . .
exp / 1	e**exp (ISO)
** / 2	exponenciação (ISO)
float / 1	conversão para float (ISO)
/ / 2	divisão (ISO)
index / 3	localiza substring em string
// / 2	divisão inteira (ISO)
random / 1	gera número aleatório inteiro
floor / 1	maior inteiro não maior que
length / 1	comprimento da string

Operadores Aritméticos

Name/arity	Explanation
log / 1	logaritmo neperiano (ISO)
log10 / 1	logaritmo decimal
mod / 2	módulo de divisão inteira (ISO)
* / 2	multiplicação (ISO)
pi / 0	número 3.14159. . .
rem / 2	resto de divisão inteira (ISO)
round / 1	inteiro mais próximo (ISO)
sign / 1	retorna -1, 0 ou +1 (ISO)
- / 1	inverte o sinal (ISO)
sin / 1	seno (ISO)
sinh / 1	seno hiperbólico
sqrt / 1	raiz quadrada (ISO)

Operadores Aritméticos

Name/arity	Explanation
- / 2	subtração (ISO)
tan / 1	tangente
tanh / 1	tangente hiperbólica
truncate / 1	parte inteira de um real (ISO)

Operadores Aritméticos

- Operadores aritméticos que forçam Prolog a avaliar uma expressão como uma expressão aritmética
  - **Operator / arity** — **Explanation**
    - **< / 2** — **menor do que (ISO)**
    - **> / 2** — **maior do que (ISO)**
    - **=< / 2** — **menor ou igual a (ISO)**
    - **>= / 2** — **maior ou igual (ISO)**
    - **=\= / 2** — **diferente (ISO)**
    - **=:= / 2** — **igual (ISO)**

Avaliador de Expressões

X is E

X variável não ligada. E expressão aritmética.

E1 op E2

- **Onde op ∈ {<, =<, >=, >, =:=, =\=}**
- **E1 e E2 são expressões aritméticas avaliadas antes da comparação.**

```
?- X = 2, Y = 5, R is sqrt(X^2+Y).
     X = 2.
     Y = 5.
     R = 3
?- X = 2, Y = 5, Y - X =\= X.
     X = 2.
     Y = 5
?- X = 2, Y = 5, Y - X < Y, write(ok),nl.
     ok
     X = 2.
     Y = 5
```

Avaliador de Expressões Exemplo

```
?- raizes.
    quer achar as raizes de a*x*x+b*x+c ?(s/n)
        Informe coef a > 1.
        Informe coef b > 1.
        Informe coef c > -12.
            x1 = 3
            x2 = 4
    quer achar as raizes de a*x*x+b*x+c ? (s/n)
        Informe coef a > 1.
        Informe coef b > 1.
        Informe coef c > 12.
        Nao tem raizes reais
    quer achar as raizes de a*x*x+b*x+c ? (s/n)
    false.
```

Avaliador de Expressões Exemplo

```
raizes:- simNao('quer achar as raizes de a*x*x+b*x+c ?'),
        obtemcoef(A,B,C),
        D is B^2-4*A*C,
        ( D >= 0,  X1 is (-B + sqrt(D))/(2*A),
                   X2 is (-B - sqrt(D))/(2*A), nl,
                   write('x1 = '), write(X1),  nl,
                   write('x2 = '), write(X2);
          D < 0, nl, write('Nao tem raizes reais.') ),
        raizes.
simNao(Msg) :- nl, write(Msg), repeat,
       write(‘ (s/n): '),
       get_char(N), nl, member(N,[’S’,’s’,’N’,’n’]),
       !, member(N,[’S’,’s’]).
obtemcoef(A,B,C) :-
       obtem('Informe coef a > ', A),
       obtem('Informe coef b > ', B),
       obtem('Informe coef c > ', C).
obtem(Msg,X) :- nl, write(Msg), read(X).
```

Átomo

- Átomos são nomes textuais usados para identificar dados, predicados, operadores, módulos, arquivos, janelas, etc. Pode ser reconhecido pelo predicado atom(X).

  <átomo> ::= <átomo-alfa>|<átomo-simb>| <string>
  <átomo-apostrofado> |<átomo-especial>

  <átomo-alfa> ::= <letra-minúscula> | <átomo-alfa><letra>|
  <átomo-alfa><dígito> | <átomo-alfa> _

  Exemplos.: a, avo, amora, a2, a_, big_32, xMax, etc.

- <átomo-simb> ::= <caracter-simb>|
  <atomo simb><caracter-simb>
  <caracter-simb> ::= #|$|%|&|*|+|-|.|/|:|<|=|>|@|\|^|`| ~
  - Exemplos.: & &: ++ << >> <-- .. *-/*
    Obs.: desde que não seja operador primitivo

Átomo

<átomo-apostrofado> ::=
'qualquer seqüência de caracteres'

- Exemplo: 'Avo' '123' 'alo mundo' 'a'
- Um caracter em si é um átomo, exemplo: 'x'

<string> ::= "qualquer seqüência de caracteres "

Átomo

<átomo especial> ::= ! | []

- **Tais átomos têm funções especiais na linguagem Prolog.**
  - **! = cut**
  - **[] = representa lista vazia**

Termo Composto

- <termo-composto> ::= <lista> |
  <estrutura>

  - Pode ser identificado por predicado:
    compound(X)
  - Subtipos
    - Listas
    - Estruturas

Lista

- **É uma seqüência de termos entre colchetes , separados por vírgulas:**
  - **Exemplos:**

    [t1, t2, t3, ..., tn]

    [ ] representa a lista vazia

    [t1, t2, t3, ..., ti | Xs] representa uma lista com os primeiros i termos, e os demais estão representados pela cauda Xs (variável).

    [X|Xs] representa uma lista com pelo menos 1 termo

    [X|Xs] = [1,2,3], com X= 1 e Xs = [2,3]

    [X,Y|Xs] = [1,2,3,4], com X=1, Y=2, e Xs =[3,4]

Lista

- **Predicados**

```
take(0,_,[]).
take(_,[],[]).
take(N,[X|Xs],[X|Ys]) :- N>0, N1 is N - 1,
                         take(N1,Xs,Ys).
```

```
?- take(3,[a,b,c,d,e], Rs).
   Rs = [a,b,c] ;
   false.
```

Lista

```
drop(0,Xs,Xs).
drop(_,[],[]).
drop(N,[X|Xs],Ys) :- N>0, N1 is N - 1,
                     drop(N1,Xs,Ys).
?- drop(3,[a,b,c,d,e], Rs).
    Rs = [d,e] ;
    false.
```

Estrutura

- É semelhante a um registro, com tipo, nome, e campos. Com sintaxe geral

  nome(t1,t2,...,tn),

  nome/n

- Exemplo:

  tem(bce, livro('Capitaes de Areia', 'Jorge Amado'))

  a(goania, brasilia, 200)

  pessoa(nome(‘Joca Estrada’), idade(42),
  filhos([‘Joao jr’, ‘Amauri Ferreir’,
  ‘Milene Ferreira’]))

Operadores

```
module(if,[se/1, entao/2, senao/2]).
:- op(800,xfx,entao).
:- op(750,fx, se).
:- op(810,xfx,senao).
se X :- X.
X entao Y :- X, Y.
X senao _ :- X,!.
_ senao Z :- Z.
      X = 12, se X=10.
            false.
      X = 12, X=12 entao Y=1.                    % implicação
            X=12,
            Y=1.
      X = 10, X==10 senao Y = 2.                 % ou
            X=10.
```

Operadores

?- X = 12, se X=10 entao Y=1.

false.

?- X = 12, se X=10 entao Y=1 senao Y = 2.

X=12,

Y=2.

?- X = 10, se X=10 entao Y=1 senao Y = 2.

X=10,

Y=1.

Operadores

Especifier	Class	Associatively
fx	prefix	non-associative
fy	prefix	right-associative
xfx	infix	non-associative
xfy	infix	right-associative
yfx	infix	left-associative
xf	postfix	non-associative
yf	postfix	left-associative

Operadores

Priority	Specifier	Operator(s)
1200	xfx	:-
1200	fx	:-
1100	xfy	;
1050	xfy	->
1000	xfy	,
900	fy	\+
700	xfx	=, \=
700	xfx	==, \==, @<, @=<, @>, @>=
700	xfx	is, =:=, =\=, <, =<, >, >=
500	yfx	+, -, /\, \/
400	yfx	*, /, //, rem, mod, <<, >>
200	xfx	**
200	xfy	^
200	fy	-, \

Operadores

Priority	Specifier	Operator(s)
1000	xfy	|
900	fy	not
700	xfx	is_string
600	yfx	&
200	fy	+

Unificando com Variável

- variável com variável

  X = Y

  - X unifica com Y e retorna true.

    ?- X = Y.

    X = Y

    - passam a representar a mesma variável.

  X \= Y

  - retorna sempre false.

    ?- X \= Y.

    false.

Unificando com Variável

- Variável em dados estruturados, há unificação se as estruturas forem isomórficas e se houver uma substituição sobre as variáveis que torne as estruturas idênticas.

```
?- X = arco(a,b,10).
    X = arco(a,b,10)
?- livro(autor(teixeira, joao), Titulo) =
        livro(Autor,titulo('Amor e Paz')).
    Titulo = titulo('Amor e Paz') ,
    Autor = autor(teixeira,joao)
```

Unificando com Variável

?- [X|Xs] = [a,b,c,d].

X = a , Xs = [b,c,d]

?- [X1,X2,c,d] = [a,b,c,d].

X1 = a , X2 = b

?- [[X|Xs],Y,c,d] = [[1,2,3],a,c,d].

X = 1 , Xs = [2,3] , Y = a

?- [[X,2,3],Y,b,c] = [[1|Xs],a,Z1,Z2].

X = 1 , Y = a , Xs = [2,3] , Z1 = b , Z2 = c

Unificando Constantes

T1 = T2

Unificam se tiverem valores idênticos e forem estruturalmente isomórficos.

```
?- 10 = 10.
    true.
?- arco(a,b,10) = arco(a,b,10).
    true.
?- 'wagner' = wagner.
    true.
?- "a b" = "a b".
    true.
```

Comparação de Termos Arbitrários Segundo Ordem Padrão

- Com os operadores
  ==/2 (idênticos), \==/2 ,
  @>=/2, @>/2 , @</2, @=</2
Tipo	Comparação
variáveis	endereços
números	valor numérico
átomos	valor ASCII

Ordem Padrão dos Termos

- **Com os operadores**
  ==/2 (idênticos), \== ,
  @>=/2, @>/2 , @</2, @=</2
  - **Segue a ordem**
    variáveis < flutuantes < inteiros
    < átomos < strings < listas < termos-compostos
  - **Se os termos são do mesmo tipo, então uma "comparação de tipo" é feita.**

Ordem Padrão dos Termos
Exemplo: qsort

```
separa(B,[X|Xs],[X|Ys],Zs) :- X @=< B,
                               separa(B,Xs,Ys,Zs).
separa(B,[X|Xs],Ys,[X|Zs]) :- X @>  B,
                               separa(B,Xs,Ys,Zs).
separa(_, [],[],[]).

qsort([X|Xs],Rs) :- separa(X,Xs,Ys, Zs),
                    qsort(Ys,Ry),
                    qsort(Zs,Rz),
                    append(Ry,[X|Rz], Rs).
qsort([],[]).
```

Ordem Padrão dos Termos
Exemplo: qsort

```
?- qsort([wagner, dada, "dada",
                  "manoel", 2.3, 4,3,7,
                   arco(a,b,5),arco(b,d,3)], Rs).
```

```
Rs = [2.3, 3, 4, 7, "dada", "manoel", dada,
                  wagner, arco(a,b,5), arco(b,d,3)] ;
```

Base de Conhecimentos

→ livro(autor('Fernando A.'), titulo('Orientacao a Objetos')).
livro(autor('Pedro Rezende'), titulo('Criptografia em Redes')).
livro(autor('Luiz Frota'), titulo('Seguranca em BD')).
livro(autor('Marcelo Ladeira'), titulo('Redes Bayesianas')).
livro(autor('Fernando A.'), titulo('Redes de Computadores')).
livro(autor('Maria Emilia'), titulo('Genoma Humano')).
livro(autor('Wagner Teixeira'), titulo('Funcoes de Crenca')).
livro(autor('Aluizio Arcela'), titulo('Computacao Sonica')).
livro(autor('Homero Picolo'), titulo('Computacao Grafica')).
livro(autor('Maria de Fatima'), titulo('IA na Educacao')).
livro(autor('Marco Aurelio'), titulo('IA na Educacao de Adultos')).
livro(autor('Li Weigang'), titulo('Redes Neurais e o Tempo')).
livro(autor('Francisco Cartaxo'), titulo('Especificacoes Formais')).
artigo(autor('Fernando A.'), titulo('quem usa Orientacao a Objetos')).
artigo(autor('Wagner Teixeira'), titulo('soma ortogonal')).
artigo(autor('Antonio Nuno'), titulo('filtro Grafico')).

Consulta sobre a Base de Livros

```
?- livro(X,Y).
   X = autor('Fernando A.') ,
   Y = titulo('Orientacao a Objetos') ;
   X = autor('Pedro Rezende') ,
   Y = titulo('Criptografia em Redes') ;
   X = autor('Luiz Frota') ,
   Y = titulo('Seguranca em BD') ;
   X = autor('Marcelo Ladeira') ,
   Y = titulo('Redes Bayesianas')
?- livro(autor('Fernando A.'),Y).
   Y = titulo('Orientacao a Objetos') ;
   Y = titulo('Redes de Computadores');
   false.
```

- **Há um ponteiro apontando para o corrente fato na base. Quando uma nova solicitação é feita , o ponteiro é atualizado para refletir o novo fato que atende a solicitação. Quando o ponteiro chega ao fim da base, a solicitação corrente falha.**

Consulta sobre uma base

- **Backtracking (retrocesso)**

```
?- artigo(autor(X),_).
    X = 'Fernando A.' ;
    X = ‘Wagner Teixeira' ;
    X = 'Antonio Nuno'
?-  artigo(autor(X),_), write(X), nl, fail.
    Fernando A.
    Wagner Teixeira
    Antonio Nuno
    false.
```

Incluindo Cláusulas na BC

- asserta(clausula).
  Inclui uma cláusula na BC, no início das cláusulas com mesmo nome.

?- dynamic(pai/2).

?- asserta(pai(joao, maria)).

true.

?- listing(pai/2).

:- dynamic(pai/2).

pai(joao, maria).

true.

Incluindo Cláusulas na BC

- **assertz(cláusula) - inclui no final das cláusulas de mesmo nome.**

```
?- assertz( (irmao(A,B) :- pai(P,A), pai(P,B), A \== B) ).
   true.
?- listing(irmao/2).
   :- dynamic irmao/2.
   irmao(A, C) :-
         pai(B, A),
         pai(B, C),
         A\==C.
   true.
```

Observa que a regra precisa estar entre parêntesis por conta da prioridade do operador :-/2 ser maior que 1000.

Univ

- Operador “=../2” transforma termo em lista
  - Apenas um dos seus operandos pode ser variável
  - Exemplos

```
?- struct(hello, X) =.. L.
      L = [struct, hello, X]
?- Term =.. [baz, foo(1)]
      Term = baz(foo(1))
?- Pai = [pai, joao, ze].
      Pai = [pai,joao,ze]
?- P =.. [pai,joao,X], call(P).      % relembre pai(joao,maria)!
      P = pai(joao,maria) , X = maria
```


call/1 chama o argumento como uma meta a ser avaliada.

Unificação

- É o processo que torna dois termos idênticos, ainda que para isso se faça substituição de variáveis em termos por outros termos.
  - o termo t_1 unifica com o termo t se existe uma substituição θ_1 que torna t_1 identico a t (representada por $t \equiv t_1\theta_1$)
    - seja t, t_1 e t_2 termos, t é uma instância comum de t_1 e t_2 se $\exists\ \theta_1$ e θ_2 substituições tais que $t \equiv t_1\theta_1$ e $t \equiv t_2\theta_2$.
    - Exemplo

      t1= concat([1,2,3],[4,5],Rs),
      t2= concat([X|XS],Ys,[X|Zs]),
      t = concat([1,2,3],[4,5],[1|Zs]).
      onde
      θ_1= (Rs/[1|Zs]),
      θ_2= (X/1, Xs/[2,3], Ys/[4,5])

Unificação

- **Unificador**
  um unificador de dois termos t e t_1 é uma substituição θ que torna $t \equiv t_1/\theta$ (t e t_1 ficam idênticos).
- **Regras da unificação**
  - **a) variáveis unificam com variáveis**
    - **X = Y.**
  - **b) variáveis unificam com termos**
    - **X = gosta(joao,ler).**
  - **c) termos unificam com termos, se eles casam**
    - **pai(P,F) = pai(joao, ze) com θ=(P/joao, F/ze).**
    - **2 = 2, joao = 'joao', etc.**

Continua ...

Unificação

- **Regras da unificação**

  d) Um termo não-atômico unifica com outro se houver correspondência estrutural.

  livro(autor(Sn,N),titulo(‘Clarissa’)) =livro(autor(‘Verissimo’,’Erico’), T).

  com θ = (Sn/’Verissimo’, N/’Erico’, T/titulo(‘Clarissa’))

- **Execução**

  ?- livro(autor(Sn,N),titulo('Clarissa')) =
  livro(autor('Verissimo','Erico'), T).

  Sn = 'Verissimo' ,

  N = 'Erico' ,

  T = titulo('Clarissa')

Prova por Refutação

- **Motivação**

$$
\begin{aligned}
BC \Rightarrow P \quad &\Leftrightarrow \quad \neg\neg(BC \Rightarrow P) \\
&\Leftrightarrow \quad \neg\neg(\neg BC \vee P) \\
&\Leftrightarrow \quad \neg(BC \wedge \neg P) \\
&\Leftrightarrow \quad \neg(BC \wedge \neg P) \vee \mathit{False} \\
&\Leftrightarrow \quad (BC \wedge \neg P) \Rightarrow \mathit{False}
\end{aligned}
$$

Resolução

- **É uma regra de inferência**

$$a)\ \frac{A \vee B, \neg B \vee C}{A \vee C}$$

$b)$ Seja os literais p_i, q, r e s_i,

onde o unificador $\theta = (pj/\neg q)$, então

$$\frac{\begin{array}{c} r :- p_1, \ldots, p_j, \ldots p_m \\ q :- s_1, \ldots, s_n \end{array}}{r :- p_1, \ldots, p_{j-1}, p_{j+1}, \ldots, p_m}$$

Resolução

- Duas cláusulas de Horn são resolvidas em uma nova cláusula se uma delas contiver um predicado negado que corresponda a um predicado não-negado na outra cláusula.
  - Tautologia

- A nova cláusula elimina o termo de correspondência e fica disponível para uso em resposta a pergunta.
  - As variáveis são substituídas por constantes associadas de maneira consistente

Exemplo de Resolução por

```
come (urso, peixe).
come (urso, raposa).          % predicado binário
come (cavalo, mato).
animal (urso).
animal (peixe).               % predicado unário
animal (raposa).
presa(X) :- come(Y,X), animal(X).    % regra
```

Regra de Inferência: Resolução

- **Observe que a regra (Cláusula de Horn)**

  presa(X) :- come(Y,X), animal(X)

  Corresponde a wff

  ∀x∀y(come(Y,X) ^ animal(X) -> presa(X))

  Corresponde a cláusula

  ¬(come(Y,X) ^ animal(X)) v presa(X)

  ¬come(Y,X) v ¬animal(X) v presa(X)

Regra de Inferência: Resolução

- ?- presa(X).
  - O Prolog procura, na BC, por uma regra com o predicado presa(X) como o conseqüente
  - Busca outras cláusulas que possam ser resolvidas com a regra
  - Faz as substituições das variáveis na cláusula regra

1. ¬come(Y,X) v ¬animal(X) v presa(X)
2. come(urso,peixe)
3. ¬animal(peixe) v presa(peixe) {resolvente de 1 e 2}
4. animal (peixe)
5. presa (peixe) {resolvente de 3 e 4}

Controle do Programa

- Controla como o programa pode ser executado.
  - Ordem das cláusulas no programa
  - Ordem das submetas dentro de cada cláusula
  - Pesquisa em profundidade (tenta satisfazer uma submeta antes de seguir para a próxima)
  - Retrocede, tentando uma outra alternativa de regra ou instância de valores para o mesmo predicado.
  - O prolog dispões de técnicas como
    - repeat - fail, loops, recursão e cut.

Ordem das Cláusulas no Programa

- Dependendo da ordem das cláusulas no programa, ele pode entrar em loop infinito, ou apresentar um melhor ou pior desempenho.
- Loop infinito.

```
fat(N, F) :- N1 is N-1, fat(N1, F1), F is N * F1.
fat(0,1).
```

  - Assim, a ordem influi para tornar o programa decidível !

Ordem das Cláusulas no Programa

- **Desempenho**
  a) ++([],Ys,Ys).
  ++([X|Xs],Ys,[X|Zs]) :- ++(Xs,Ys,Zs).

  b) ++([X|Xs],Ys,[X|Zs]) :- ++(Xs,Ys,Zs).
  ++([],Ys,Ys).
  - **A opção b apresenta melhor desempenho, sem risco de loop, pois o padrão [X|Xs] impede o casamento com lista vazia.**
    - **quando X e Xs forem constantes.**
    - **no caso de serem variáveis o desempenho é pior.**

(++)/3 é uma outra forma de representação do predicado concat/2.

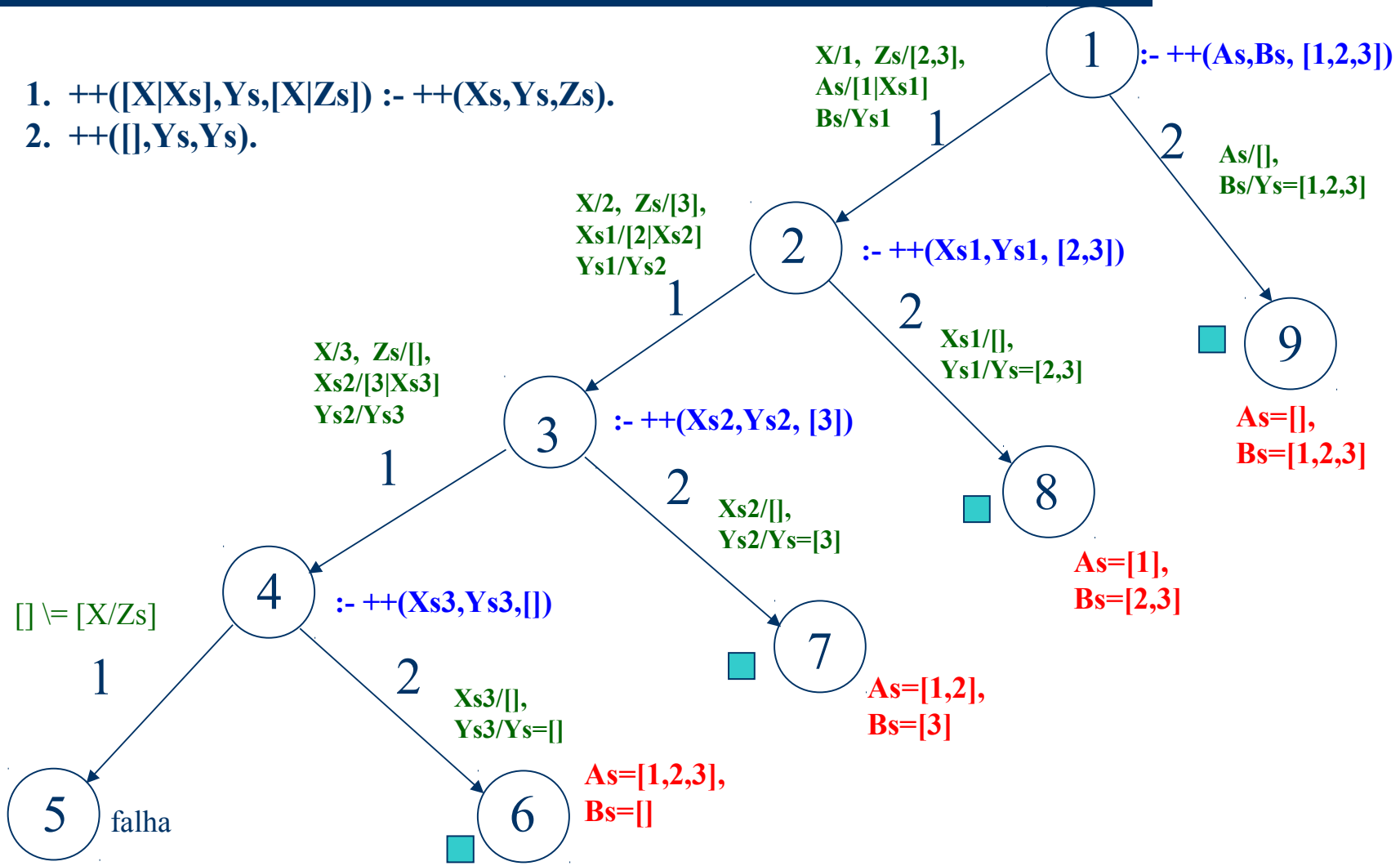
Ordem das Cláusulas no Programa

```
append([],Ys,Ys).
append([X|Xs], Ys, [X|Zs]) :-
    append(Xs,Ys,Zs).

?- append(As, Bs, [1,2,3]).
     As = [] ,
     Bs = [1,2,3] ;
     As = [1] ,
     Bs = [2,3] ;
     As = [1,2] ,
     Bs = [3] ;
     As = [1,2,3] ,
     Bs = [] ;
false.
```

```
++([X|Xs], Ys, [X|Zs]) :- ++
(Xs,Ys,Zs).
++([],Ys,Ys).

?- ++(As, Bs, [1,2,3]).
     As = [1,2,3] ,
     Bs = [] ;
     As = [1,2] ,
     Bs = [3] ;
     As = [1] ,
     Bs = [2,3] ;
     As = [] ,
     Bs = [1,2,3]
```

Árvore de Refutação

1. ++([X|Xs],Ys,[X|Zs]) :- ++(Xs,Ys,Zs).
2. ++([],Ys,Ys).



Ordem das submetas na cláusula

- tio(T,S) :- irmao(P,T), pai(P,S).
  - T pode ter vários irmãos, mas S só tem um pai. Assim,

    tio(T,S) :- pai(P,S), irmao(P,T).

    É mais eficiente, pois não provoca retrocesso, ao tentar buscar outro P irmão de T que possa ser pai de S.

Retrocesso (backtracking)

- **Meta**
  - **Sucede, se todas as submetas no corpo sucedem.**
  - **Falha, se ao menos uma submeta no corpo falha.**
  - **Quando há falha, o Prolog tenta satisfazer a meta com outras alternativas, buscando um casamento a partir de onde a meta ficou.**
    - **Esse processo de tentar satisfazer todas as submetas, antes de dá-la por falha é o backtracking**

Retrocesso

```
simNao(Msg) :- nl, write(Msg), repeat,
        write(' (s/n) '), get_char(N), nl,
        member(N,"SsNn"), !,
        member(N,"Ss").
?- simNao('Quer jogar primeiro? ').
    Quer jogar primeiro? (s/n)
          t
    (s/n)
          n
    false.
```


Note que o repeat repete o que se segue a ele, até o final da cláusula!

Operadores e Predicados de controle

- , (virgula) - conjunção.
  - Todas as submetas ligadas precisam ser verdadeiras para a meta da cláusula ser verdadeira.

```
irmas(A,B) :- pai(P,A), pai(P,B), A\==B.
```

- ; (ponto e vírgula) - Disjunção.
  - Ao menos um dos disjuntos deve suceder para que a meta da cláusula suceda.

```
irmas(A,B) :- (   pai(P,A), pai(P,B);
                  mae(M,A), mae(M,B) ), A\==B.
not(P) :- P, !, fail; true.          % sucede se P falha.
```

Operadores e Predicados de controle

- Não é padrão para todo prolog

```
ifthen (P,Q) :- P, !, Q.              % falha sob retrocesso
ifthen(P,Q) :- !.
ifthenelse(P,Q,R) :- P, !, Q.    % falha sob retrocesso
ifthenelse(P,Q,R) :- not(P), !, R.
case([C1  ->  P1,
   C2  ->  P2,
   ....
   Cn -> Pn|P]).
```

Operadores e Predicados de controle

- ! - cut. Sempre verdadeiro. Sob retrocesso, o predicado que o contém falha.
- Predicado com cut

```
simNao(Msg) :- nl, write(Msg), repeat, write('s/n ?'),
   get_char(N), nl, member(N,"SsNn"), !,
   member(N,"Ss").

?- simNao('Continua? ').
     Continua? (s/n)
          r
     (s/n)
          n
     false.
```

Operadores e Predicados de controle

```
P -> Q                          % Se P então Q.
    X = 2 -> Y is X+3, Y < 7 -> Z is Y+4.
    X = 2 ,
    Y = 5 ,
    Z = 9
?- X = 2 -> Y is X+3, Y > 7 -> Z is Y+4.
    false.
```

repeat - fail

- O predicado repeat força um programa a gerar soluções alternativas via retrocesso.
  - Exemplos:

```
% cidades e vizinhos.
cidades :- repeat, nl, write(' cidade (ultima = fim) >'),
                read(X), assertz(cidade(X)), X= fim, !.
vizinhos :- repeat, cidade(X),
    ( X = fim, !;
      nl, write('Informe os vizinhos  de '),
      write(X), read(Ys), assertz(vizinhos(X,Ys))), fail.
```

Resultado da execução

```
?- cidades.
        cidade
                go.
        cidade
                bsb.
        cidade
                bh.
        cidade
                fim.
        true.
?- vizinhos.
        Informe os vizinhos  de go|
                [bsb,cuiaba, bh].
        Informe os vizinhos  de bsb
                [go, bh].
        Informe os vizinhos  de bh
                [bsb, go].
        true.
```

Situação no db

```
?- listing([vizinhos/2, cidade/1]).
    /* vizinhos/2 */
    vizinhos( go, [bsb,cuiaba,bh] ).
    vizinhos( bsb, [go,bh] ).
    vizinhos( bh, [bsb,go] ).

    /* cidade/1 */
    cidade( go ).
    cidade( bsb ).
    cidade( bh ).
    cidade( fim ).
    true.
```

Cut - fail, cut e espaço de busca

- O cut é usado em geral para manusear exceções e restringir o espaço de busca.

Cut - fail

```
elegivel(X) :- analfabeto(X), !, fail.
elegivel(X) :- cidadao(X),idade(X,Y), Y >= 18.
cidadao(joao).
cidadao(susana).
cidadao(roberta).
cidadao(ricardo).
cidadao(marcelo).
idade(joao, 36).
idade(susana, 15).
idade(roberta, 31).
idade(ricardo, 22).
idade(marcelo, 17).
analfabeto(joao).
```

```
?- elegivel(joao).
     false.     % busca mínima, cut ok
?- elegivel(ricardo).
     true.      % busca mínima, cut ok
?- elegivel(X).
     false. % como? Nesse caso devo  tirar o "!" ?
?- elegivel(X).      % tirando o cut
     X = joao ; % como? Ele é analfabeto!
     X = roberta ;
     X = ricardo ;
     false.
```

Cut - fail

```
elegivel(X) :- cidadao(X),
               analfabeto(X), fail.
elegivel(X) :- cidadao(X),
               not( analfabeto(X)),
                idade(X,Y), Y >= 18.

cidadao(joao).
cidadao(susana).
cidadao(roberta).
cidadao(ricardo).
cidadao(marcelo).
idade(joao, 36).
idade(susana, 15).
idade(roberta, 31).
idade(ricardo, 22).
idade(marcelo, 17).
analfabeto(joao).
```

```
?- elegivel(X).
     X = roberta ;
     X = ricardo ;
     false.
?- elegivel(joao).
     false.
?- elegivel(roberta).
     true.
?- elegivel(marcelo).
     false.
?- elegivel(susana).
     false.
```

Obs.: nem sempre o cut é a melhor solução.

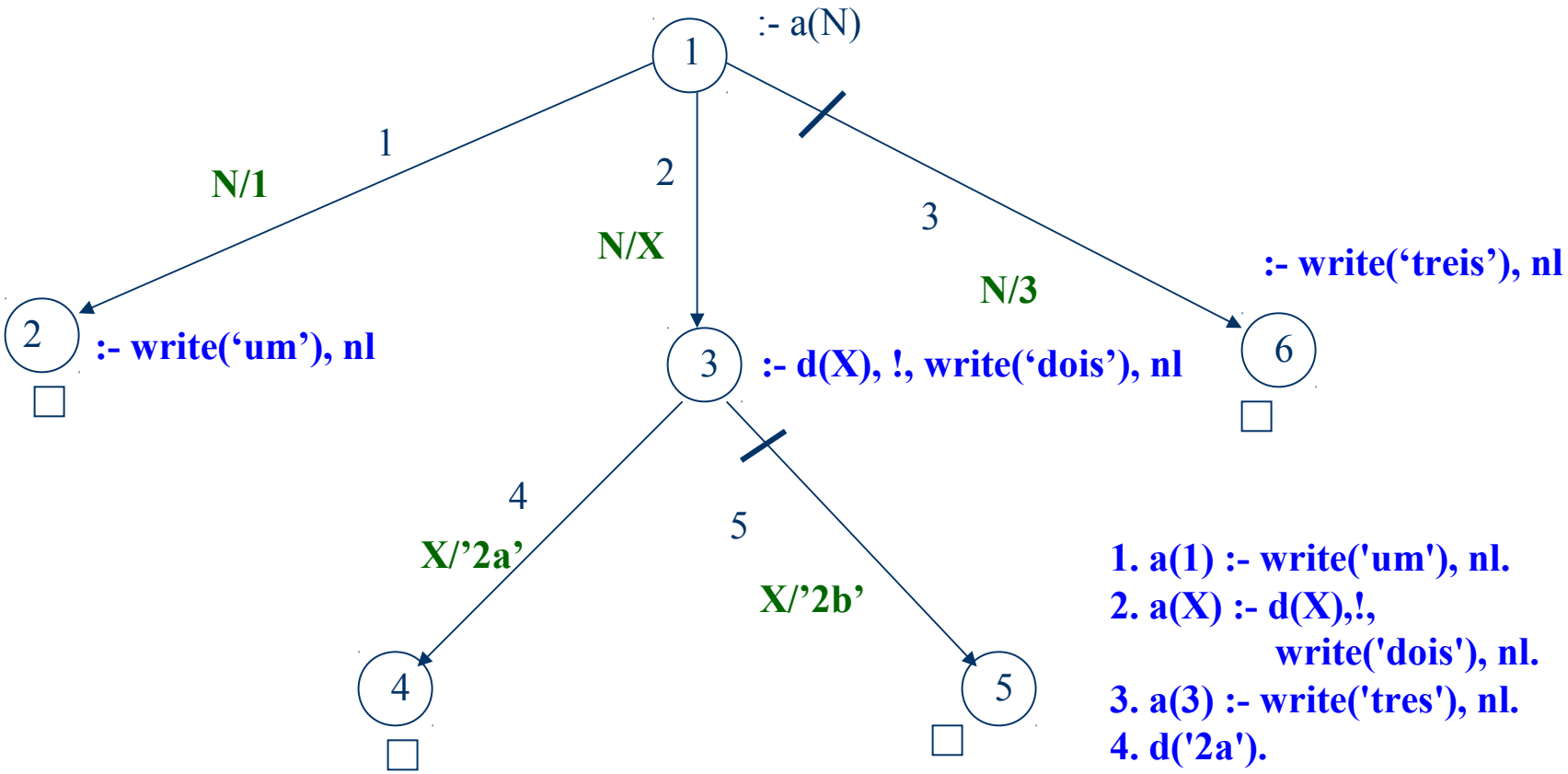
Cut - fail

```
a(1) :- write('um'), nl.
a(X) :- d(X),
        write('dois'), nl.
a(3) :- write('tres'), nl.
d('2a').
d('2b').
?- a(N).
      um
      N = 1 ;
      dois
      N = '2a' ;
      dois
      N = '2b' ;
      tres
      N = 3
?-
```

```
a(1) :- write('um'), nl.
a(X) :- d(X),!, write('dois'), nl.
a(3) :- write('tres'), nl.
d('2a').
d('2b').

?- a(N).
      um
      N = 1 ;
      dois
      N = '2a'
?-
```

Obs.: faça a árvore de refutação para ter uma melhor compreensão do cut

cut e espaço de busca




```
1. a(1) :- write('um'), nl.
2. a(X) :- d(X),!,
           write('dois'), nl.
3. a(3) :- write('tres'), nl.
4. d('2a').
5. d('2b').
```

Cut e espaço de busca

- O espaço de busca de uma consulta é o conjunto de todas as respostas possíveis.
- Cut
  O corte é usado para reduzir o espaço de busca, instruindo o interpretador a não retroceder sobre os predicados que precedem o cut.
  - Desvantagens
    - sacrifica a clareza do programa
    - causa disruptura brusca na execução do programa.
  - Efeito sobre consultas compostas
    |?- a(X), b(Y), !, c(X,Y,Z)
    obtenha um único valor para X e Y e avalie c(X,Y,Z).
  - Efeito sobre cláusulas de um mesmo predicado
    - se uma das cláusulas tem um corte no seu corpo, e o prolog o alcança, então ele não faz retrocesso sobre outras cláusulas do conjunto. (Ex.: ++, e a(N) com árvore de refutação). Estude o caso de ++ com corte, sobre a árvore de refutação.

Parando depois da 1a resposta

```
cidadao(joao).
cidadao(susana).
cidadao(roberta).
cidadao(ricardo).
cidadao(marcelo).
idade(joao, 36).
idade(susana, 15).
idade(roberta, 31).
idade(ricardo, 22).
idade(marcelo, 17).
analfabeto(joao).
```

?- idade(joao, Y), !.

Y = 36

Obs.: dado que cada pessoa só tem uma idade, o corte é desejável

?- idade(X,Y),!, Y > 18.

X = joao ,

Y = 36

obs.: Neste último caso, quer-se mais de uma resposta, para diferentes valores de X, contudo, o cut impediu.

Corte na submeta

```
temRG(X) :- homem(X),
            idade(X,Y),!,Y>18.
mulher(susana).
mulher(roberta).
homem(joao).
homem(ricardo).
homem(marcelo).
idade(joao, 36).
idade(susana, 15).
idade(roberta, 31).
idade(ricardo, 22).
idade(marcelo, 17).
```

```
?- temRG(X).
     X = joao
?-
```

Obs.: reduz-se o espaço de busca, mas não atende o que se pretende.

Isolando o efeito do corte

```
temRG(X) :- homem(X),
            idade(X,Y), Y>18.
mulher(susana).
mulher(roberta).
homem(joao).
homem(ricardo).
homem(marcelo).
idade(joao, 36) :- !.
idade(susana, 15) :- !.
idade(roberta, 31) :- !.
idade(ricardo, 22) :- !.
idade(marcelo, 17) :- !.
```

```
?- temRG(X).
     X = joao ;
     X = ricardo ;
     false.
     obs.: agora OK, mas..

?- idade(X,Y).
     X = joao ,
     Y = 36
?-
```

Obs.: o corte no predicado idade/2 não permite múltiplas respostas.

Isolando o efeito do corte

```
temRG(X) :- homem(X),
            souma(idade(X,Y)), Y>18.
mulher(susana).
mulher(roberta).
homem(joao).
homem(ricardo).
homem(marcelo).
idade(joao, 36).
idade(susana, 15).
idade(roberta, 31).
idade(ricardo, 22).
idade(marcelo, 17).
souma(P) :- P, !.
```

```
?- temRG(X).
     X = joao ;
     X = ricardo ;
     false.
?- idade(X,Y).
     X = joao ,
     Y = 36 ;
     X = susana ,
     Y = 15 ;
     X = roberta ,
     Y = 31 ;
     X = ricardo ,
     Y = 22 ;
     X = marcelo ,
     Y = 17
?-
```

Isolando o efeito do corte

- O predicado souma(P) enseja um uso adequado do cut, quando a pretensão é eliminar buscas desnecessárias, ou restringir a uma só resposta.
  - Não precisa usar cut diretamente na consulta
  - Não precisa usar cut sobre cláusulas na base.
  - souma(P) falha sob retrocesso, mas não impende que variáveis em predicados anteriores, no mesmo corpo, sejam revaloradas.
  - É assim uma solução mais limpa para o cut.