Linguagem Prolog
Conceitos Básicos

TURMA A
PROF. MARCELO LADEIRA CIC/UNB

ADAPTADOS A PARTIR DE SLIDES OBTIDOS NA INTERNET – PARTE 3 DE 4

Controle do Programa

Controla como o programa pode ser executado:

- ordem das cláusulas no programa;
- ordem das submetas dentro de cada cláusula;
- busca 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, 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.

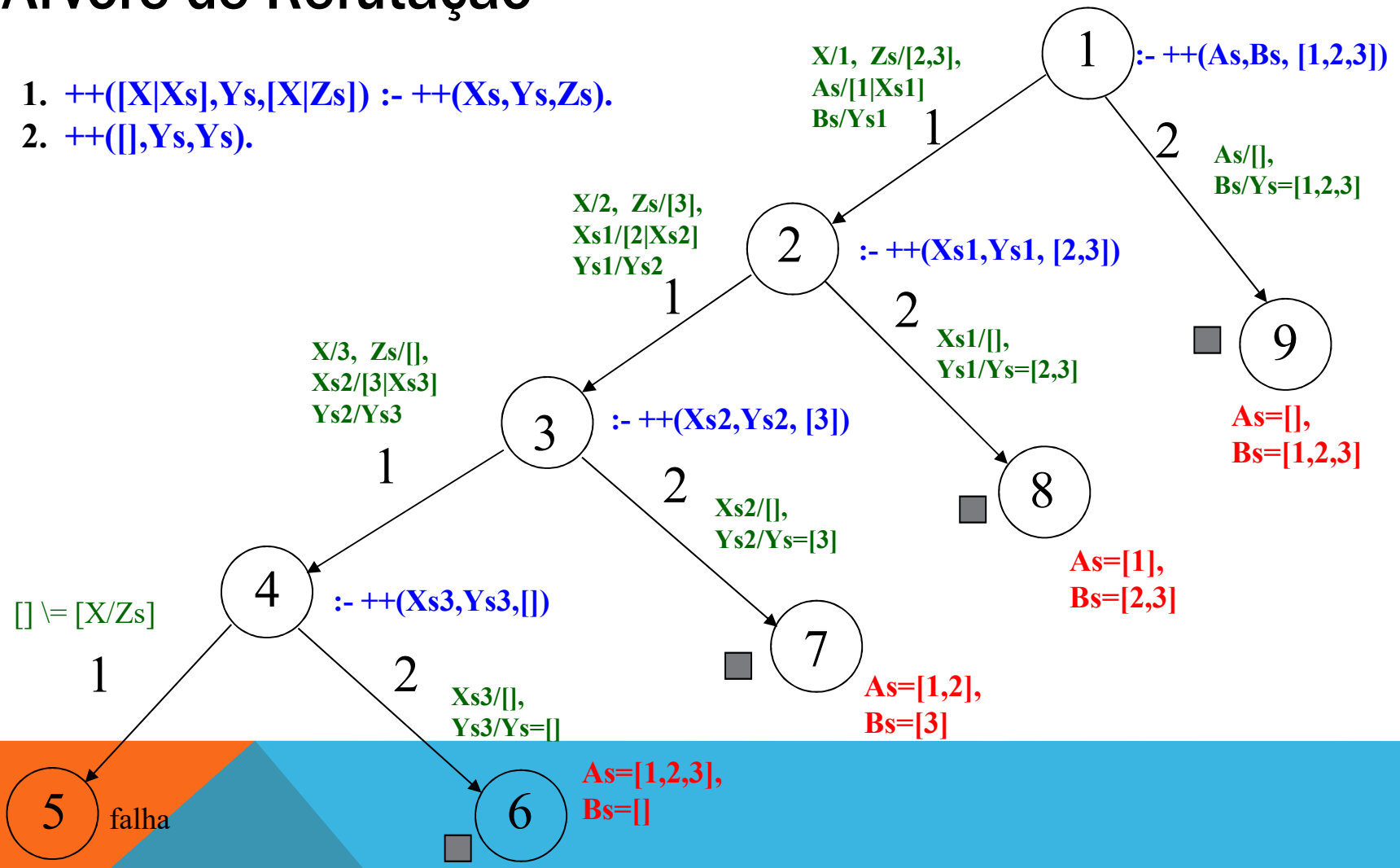
Ordem das Cláusulas no Programa

```
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 falhou.
  - 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.

```
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 na BC

```
?- 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.
```