Representação do Conhecimento e Raciocínio
Programação Lógica
Prolog – parte 4

Inteligência Artificial - 2020/1

Prolog – parte 4

- Controle de retrocesso (corte)
- Predicados especiais para entrada e saída
- Predicados sem argumentos e predicados com mesmo nome

Prolog – Controle de retrocesso (corte)

- O retrocesso (backtracking) é um processo pelo qual todas as alternativas de solução para uma dada consulta são tentadas exaustivamente.
- No Prolog, o retrocesso é automático.
- É possível controlá-lo através de um predicado especial chamado **corte**, notado por **!** .
- Visto como uma cláusula, seu valor é sempre verdadeiro. Sua função é provocar um efeito colateral que interfere no processamento padrão de uma consulta.

Prolog – Controle de retrocesso (corte)

- Pode ser usado em qualquer posição no lado direito de uma regra.
- O corte é adequado às situações onde regras diferentes são aplicadas em casos mutuamente exclusivos.
- Faz com que o programa se torne mais rápido e ocupe menos memória.
- Quando colocado no final de uma cláusula que define um predicado, evita que as cláusulas abaixo dessa, relativas ao mesmo predicado, sejam usadas no backtracking.

Prolog – Controle de retrocesso (corte)

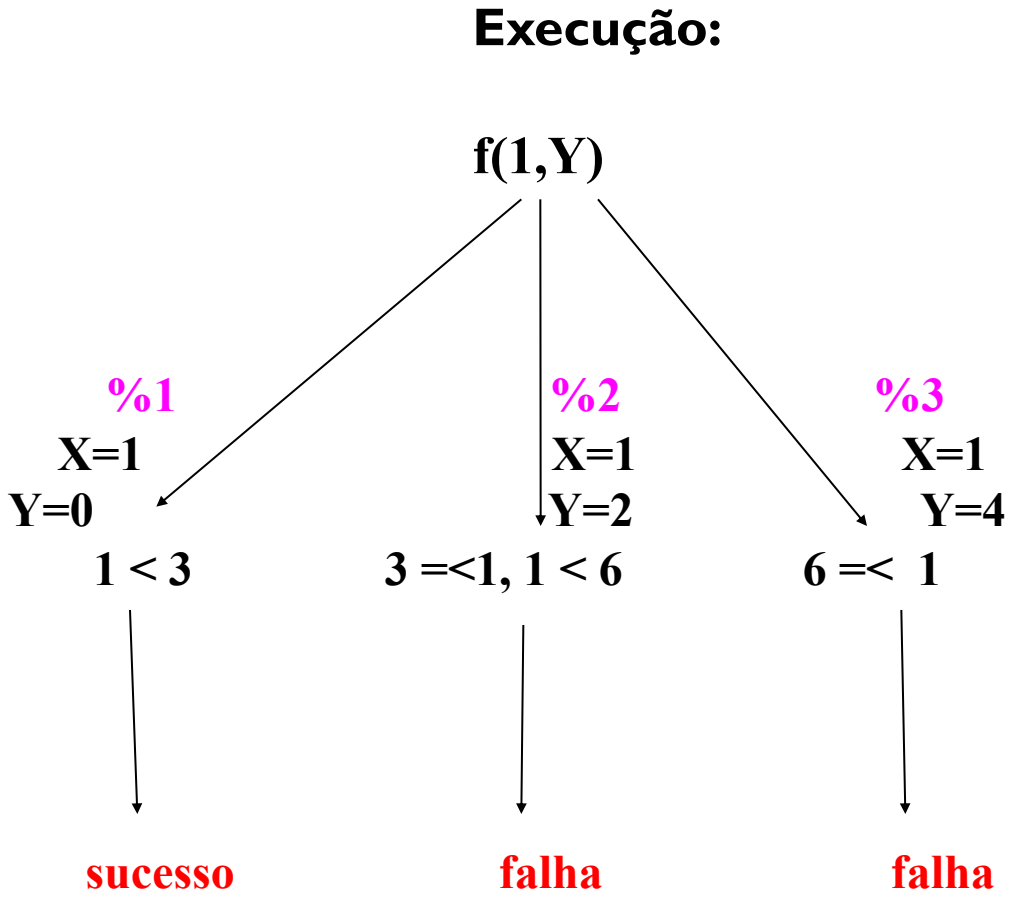
Exemplo:Construir um programa Prolog para implementar a função

$$f(x) = \begin{cases} 0 \text{ se } x < 3 \\ 2 \text{ se } x >= 3 \text{ e } x < 6 \\ 4 \text{ se } x >= 6 \end{cases}$$

```
f(X,0) :- X < 3.                %1
f(X,2) :- 3 =< X, X < 6 .       %2
f(X,4) :- 6 =< X.               %3
```

Prolog – Controle de retrocesso (corte)

Consulta:
?- f(1,Y).
Y = 0 ;
false.



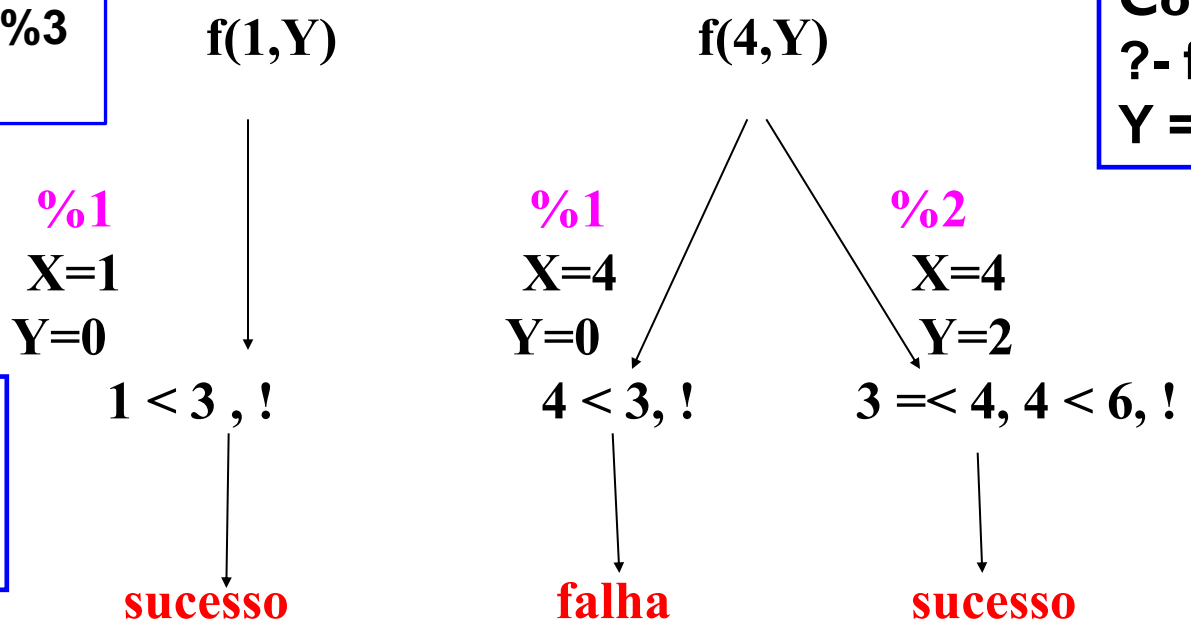
Prolog – Controle de retrocesso (corte)

- Na busca, as regras 2 e 3 são tentadas inutilmente, resultando em falha.
- Sabemos, no momento da programação, que as regras representam casos mutuamente exclusivos.
- O uso do corte torna-se conveniente, para evitar esforço de busca desnecessário e tornar a execução da consulta mais eficiente.

```
f(X,0) :- X < 3, ! .                    %1
f(X,2) :- 3 =< X, X < 6, ! .            %2
f(X,4) :- 6 =< X.                       %3
```

Prolog – Controle de retrocesso (corte)

```
f(X,0) :- X < 3, ! .          %1
f(X,2) :- 3 =< X, X < 6, ! .  %2
f(X,4) :- 6 =< X.             %3
```



Consulta:
?- f(1,Y).
Y = 0.

Consulta:
?- f(4,Y).
Y = 2.

Este tipo de corte é chamado de corte verde : se for retirado, o programa tem exatamente o mesmo significado.
Altera-se apenas a eficiência da execução.

Prolog – Controle de retrocesso (corte)

- O corte pode também ser usado para tornar o programa mais compacto, sem ter que escrever explicitamente as condições de aplicação de cada regra.

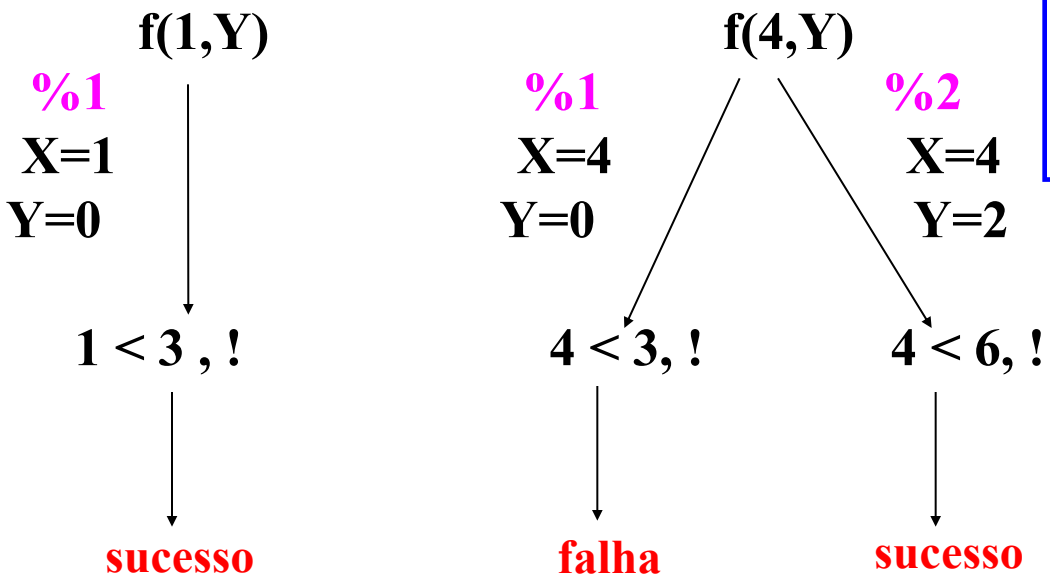
```
f(X,0) :- X < 3, ! .        %1
f(X,2) :- X < 6, ! .        %2
f(X,4).                     %3
```

- Este tipo de corte é chamado de corte vermelho : quando retirado, o programa tem significado diferente, geralmente produzindo resultados errados.

Prolog – Controle de retrocesso (corte)

```
f(X,0) :- X < 3, ! .      %1
f(X,2) :- X < 6, ! .      %2
f(X,4).                   %3
```

Execuções:



Consulta:
?- f(1,Y).
Y = 0.

Consulta:
?- f(4,Y).
Y = 2.

Na consulta f(4,Y), quando a cláusula 1 é usada, 4<3 falha e o corte não é executado, assim existe backtracking e a cláusula 2 é usada.
Retirando-se os cortes, o programa produz resultados errados.

Prolog – Controle de retrocesso (corte)

- Com o uso do corte, vários programas já estudados podem ser modificados para ficarem mais eficientes.

- Para usar o corte, deve ser analisada a operação que se espera realizar com o predicado.

Prolog – Exemplos de uso do corte

Eliminar todas as ocorrências de um elemento de uma lista

Versão SEM corte:

```
del_todas(Elem,[ ],[ ]).
del_todas(Elem, [Elem|Y], Z) :-   del_todas(Elem,Y,Z).
del_todas(Elem,[Elem1|Y], [Elem1|Z]) :- Elem \== Elem1,
                                        del_todas(Elem,Y,Z).
```

Versão COM corte:

```
del_todas_2(Elem,[ ],[ ]) :- !.
del_todas_2(Elem, [Elem|Y], Z) :- del_todas_2(Elem,Y,Z), !.
del_todas_2(Elem,[Elem1|Y], [Elem1|Z]) :-
                                        del_todas_2(Elem,Y,Z).
```

Prolog – Exemplos de uso do corte

Eliminar todas as ocorrências de um elemento de uma lista

```
?- del_todas(a,[a,b,c,a,4,a,c,b],L).
   L = [b, c, 4, c, b] ;
   false.

?- del_todas_2(a,[a,b,c,a,4,a,c,b],L).
   L = [b, c, 4, c, b].
```

Prolog – Exemplos de uso do corte

- Contar o número de ocorrências de um dado elemento no primeiro nível de uma lista:-

```
conta_ocorr(Elem,[ ],0) :- !.

conta_ocorr(Elem,[Elem|Y],N) :-
                                conta_ocorr(Elem,Y,N1),
                                N is N1 + 1, !.

conta_ocorr(Elem,[Elem1|Y], N) :-
                                 conta_ocorr(Elem,Y,N).
```

Prolog – Exemplos de uso do corte

?- conta_ocorr(x,[e,34,x,[e,d,f],par(a,b),x,567,x],S).

S = 3.

?- conta_ocorr(x,[e,34,x,[e,x,f], x, kfkfkf,5069],S).

S = 2.

?- conta_ocorr(a,[e,l,e,m,e,n,t,o],S).

S = 0.

Prolog – Entrada e Saída

- Prolog possui predicados especiais que executam, como efeito colateral, a entrada e a saída de termos para o programa

- Predicado read

- Sintaxe: read(Termo)

  onde Termo (variável ou átomo) pode ou não estar instanciado

- Lê um termo do dispositivo de entrada corrente e unifica com Termo. O termo dado deve ser seguido de . (ponto).

Prolog – Exemplos de uso do corte

Predicado read - Exemplos

```
?- read(X), Y is X + 1.
|: 3.
X = 3 ,
Y = 4.

?- read(X), read(Y), Z is X+Y.
|: 3.
|: 8.
X = 3,
Y = 8,
Z = 11.
```

Prolog – Exemplos de uso do corte

- Predicado write
- Sintaxe: write(Termo)

  onde Termo pode ou não estar instanciado
- Escreve o termo no dispositivo de saída corrente
- Predicado nl
- muda para próxima linha no dispositivo de saída

Prolog – Exemplos de uso do corte

Predicado write - Exemplos

```
?- write(palavra).
palavra
true.

?- write([a,b,c]).
[a,b,c]
true.
```

```
?- write(primeira), write('   '),
   write(segunda).
primeira   segunda
true.

?- write(primeira), nl, write(segunda).
primeira
segunda
true.
```

Prolog - Predicados sem argumentos e predicados com mesmo nome

- Um predicado é identificado pelo seu nome e pela aridade(número de argumentos).
- Predicados com o mesmo nome e com número de argumentos diferentes são considerados diferentes.
- Os predicados sem argumentos são normalmente usados para identificar procedimentos que usam read e write ou para iniciar programas com muitos predicados.

Prolog - Predicados sem argumentos e predicados com mesmo nome

Exemplo: Soma dos elementos de uma lista numérica

```
soma :- write('Digite uma lista de numeros'),
        read(Lista),
        soma(Lista,Resultado),
        write('A soma dos elementos da lista e = '),
        write(Resultado),
        nl.

soma([ ],0).
soma([Elem| Cauda], S) ;- soma (Cauda,S1),
                          S is S1 + Elem.
```

Prolog - Predicados sem argumentos e predicados com mesmo nome

Exemplo: Soma dos elementos de uma lista numérica

```
soma :- write('Digite uma lista de numeros'),
        read(Lista),
        soma(Lista,Resultado),
        write('A soma dos elementos da lista e = '),
        write(Resultado),
        nl.

soma([ ],0).
soma([Elem| Cauda], S) ;- soma (Cauda,S1),
                          S is S1 + Elem.
```

Consulta

```
| ?- soma.
Digite uma lista de numeros|: [4,5,6,4.4,0.3,-7].
A soma dos elementos da lista e = 12.7
true.
```

- Fim do Tópico Programação Lógica