GSI010 - Programação Lógica
Controle

Aula de hoje

- Backtracking
- Cut (!)
- Fail

Motor de inferência e *Backtracking*

Backtracking ou retrocesso é

Um algoritmo para a busca por soluções que no Prolog assume a forma de uma árvore.

Motor de inferência e *Backtracking*

Backtracking ou retrocesso é

Um algoritmo para a busca por soluções que no Prolog assume a forma de uma árvore.

Atuação do algoritmo

Incrementalmente encontra soluções candidatas e eventualmente abandona (**backtracking**) parte das soluções assim que verifica-se não serem alternativas viáveis.

Motor de inferência e *Backtracking*

Backtracking ou retrocesso é

Um algoritmo para a busca por soluções que no Prolog assume a forma de uma árvore.

Atuação do algoritmo

Incrementalmente encontra soluções candidatas e eventualmente abandona (**backtracking**) parte das soluções assim que verifica-se não serem alternativas viáveis.

Caminho de busca de soluções

Busca em profundidade em árvores: soluções e constrói árvore de cima para baixo, da esquerda para direita.

Backtracking

Backtracking

Quando o prolog falha, ou seja, chega em um `false`, o backtracking é acionado.

```
1 d(0).
2 d(1).
3 binario([A,B,C]) :- d(A), d(B), d(C).
```

```
1 ?- bin(N).
```

Backtracking - Exemplo

```
1 d(0).
2 d(1).
3 bin([A,B,C]) :- d(A), d(B), d(C).
```

```
1 ?- bin([1,0,1]).
```

1. Selecionar variável mais à esquerda para encontrar respostas.

Backtracking - Exemplo

```
1 d(0).
2 d(1).
3 bin([A,B,C]) :- d(A), d(B), d(C).
```

```
1 ?- bin([1,0,1]).
```

1. Selecionar variável mais à esquerda para encontrar respostas.
2. Procurar e unificar variável. Se houver mais de uma possibilidade de unificação, pegar a primeira da lista e removê-la.

Backtracking - Exemplo

```
1 d(0).
2 d(1).
3 bin([A,B,C]) :- d(A), d(B), d(C).
```

```
1 ?- bin([1,0,1]).
```

1. Selecionar variável mais à esquerda para encontrar respostas.
2. Procurar e unificar variável. Se houver mais de uma possibilidade de unificação, pegar a primeira da lista e removê-la.
3. Repetir até unificar todas as variáveis.

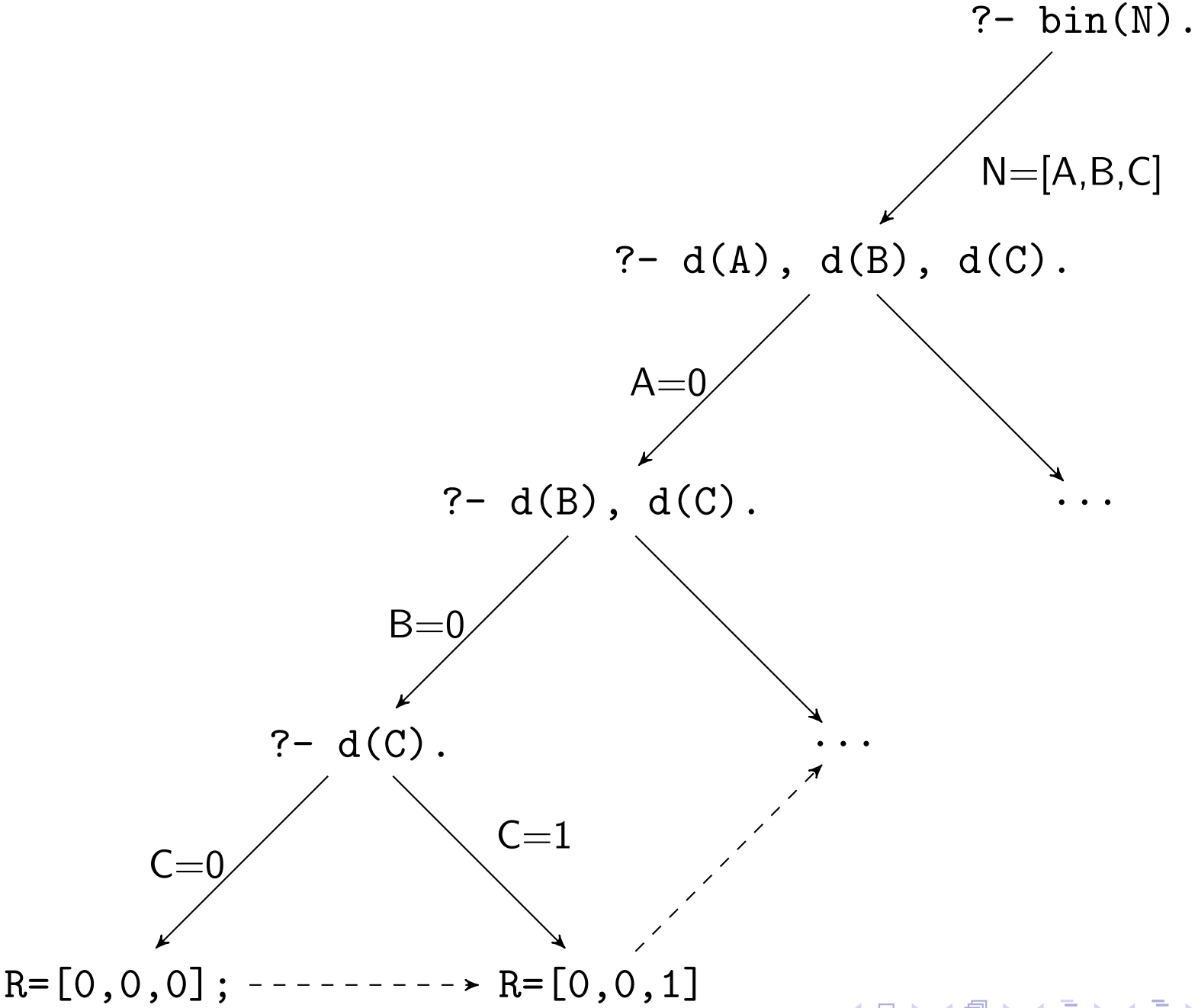
Backtracking - Exemplo

```
1 d(0).
2 d(1).
3 bin([A,B,C]) :- d(A), d(B), d(C).
```

```
1 ?- bin([1,0,1]).
```

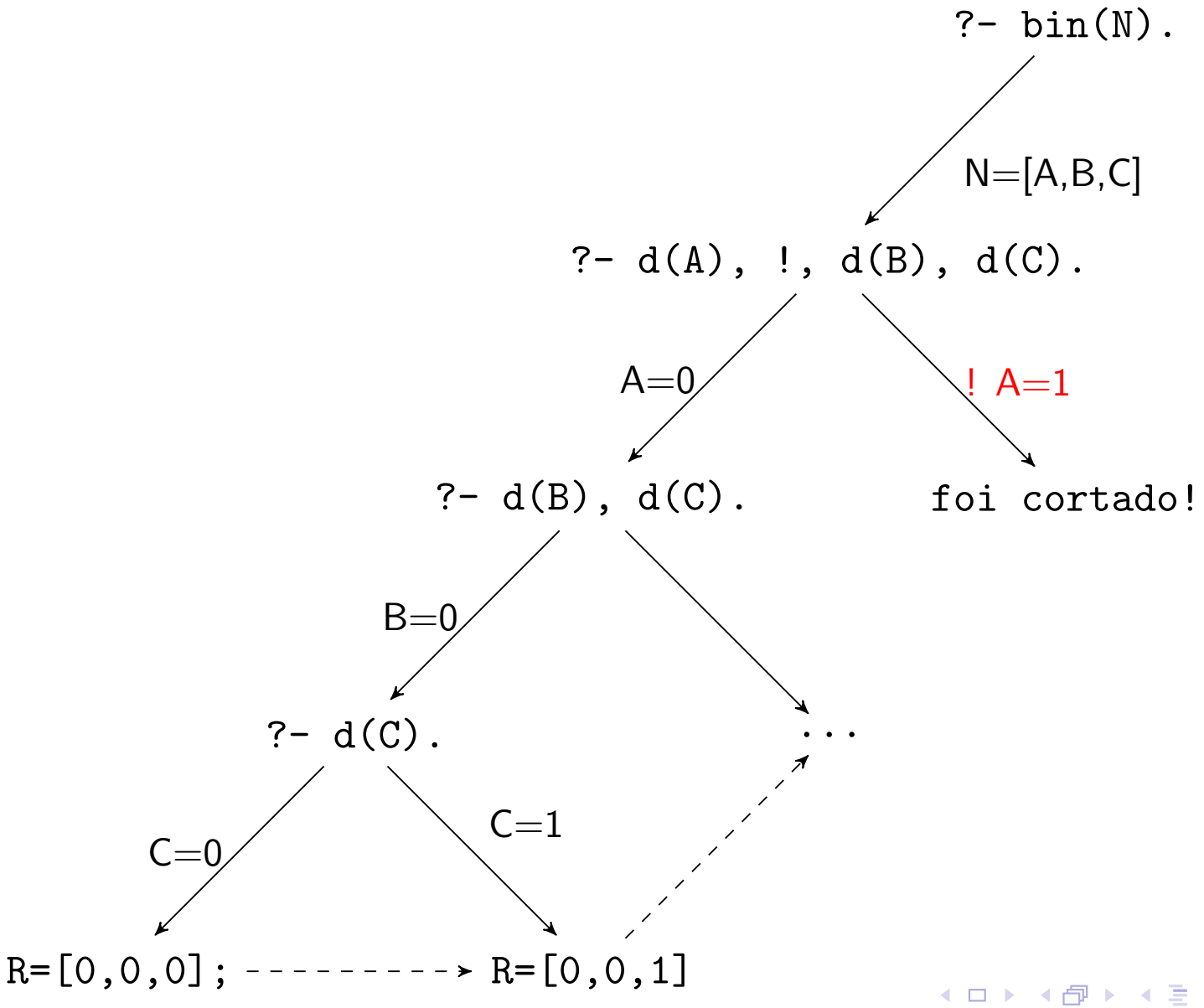
1. Selecionar variável mais à esquerda para encontrar respostas.
2. Procurar e unificar variável. Se houver mais de uma possibilidade de unificação, pegar a primeira da lista e removê-la.
3. Repetir até unificar todas as variáveis.
4. Se alguma variável não for possível unificar ou se o operador **;** for usado, inicia **backtracking**.

Algoritmo Backtracking



Corte: !

```
1 binario([A,B,C]) :- d(A), !, d(B), d(C).
```



Evitar retrocesso desnecessário

$f(x) = 0$ se $x < 5$
$f(x) = 1$ se $x \geq 5, x \leq 9$
$f(x) = 2$ se $x > 9$
Regras mutuamente excludentes.

```
1 f(X,0) :- write('mostrou 1\n'),X<5.
2 f(X,1) :- write('mostrou 2\n'),X>=5, X=<9.
3 f(X,2) :- write('mostrou 3\n'), X>9.
```

Evitar retrocesso desnecessário

$f(x) = 0$ se $x < 5$
$f(x) = 1$ se $x \geq 5, x \leq 9$
$f(x) = 2$ se $x > 9$
Regras mutuamente excludentes.

```
1 f(X,0) :- write('mostrou 1\n'),X<5.
2 f(X,1) :- write('mostrou 2\n'),X>=5, X=<9.
3 f(X,2) :- write('mostrou 3\n'), X>9.
```

```
1 f(X,0) :- write('mostrou 1\n'),X<5,!.
2 f(X,1) :- write('mostrou 2\n'),X>=5, X=<9,!.
3 f(X,2) :- write('mostrou 3\n'), X>9,!.
```

Evitar retrocesso desnecessário

```
1 max(X,Y,Y):- X =< Y.
2 max(X,Y,X):- X>Y.
```

Evitar retrocesso desnecessário

```
1 max(X,Y,Y):- X =< Y.
2 max(X,Y,X):- X>Y.
```

```
1 max(X,Y,Y) :- X =< Y,!.
2 max(X,Y,X) :- X>Y.
```

Evitar retrocesso errôneo

```
1 fatorial(1,1).
2 fatorial(X,F) :- X1 is X -1 , fatorial(X1, F1), F is X*
      F1.
```

Evitar retrocesso errôneo

```
1 fatorial(1,1).
2 fatorial(X,F) :- X1 is X -1 , fatorial(X1, F1), F is X*
      F1.
```

```
1 fatorial(1,1) :- !.
2 fatorial(X,F) :- X1 is X -1 , fatorial(X1, F1), F is X*
      F1.
```

Tipos de cortes

Cortes verdes

Todas as respostas/unificações são mantidas. Não existe modificação de resultados.

```
1 max(X,Y,Z) :- X =< Y,!, Y = Z.
2 max(X,Y,X).
```

Cortes vermelhos

Respostas podem ser modificadas. Redução das unificações possíveis.

Sensível a problemas.

- podem ser difíceis de ler/entender
- podem levar a erros sutis de programação

Predicado `fail/0`

Negação de falha

O predicado faz o prolog retroceder e executar o processo de `backtracking`.

Equivalente ao operador `not`:

```
1 neg(Objetivo):- Objetivo, !, fail.
2 neg(Objetivo).
```

```
1
2 executa(1):- nl, write('executou opcao 1'), nl.
3 executa(2):- nl, write('executou opcao 2'), nl.
4 executa(3):- nl, write('executou opcao 3'), nl.
5
6 menu:-
7 repeat,
8 write('1 - Opção A. '),nl,
9 write('2 - Opção B. '),nl,
10 write('3 - Opção C. '),nl,nl,
11 write('Introduza a opção: '),
12 read(Opcao),
13 executa(Opcao), fail.
```

Exercício

Use corte verde para melhorar estes predicados.

```
1 class(Number, positive) :- Number > 0.
2 class(0, zero).
3 class(Number, negative) :- Number < 0.
```

Exercício

Usando `cut`, escreva um predicado separar/3, que separa uma lista de inteiros em duas listas, uma contendo os números positivos (e o zero) e outra contendo os números negativos. Por exemplo:

```
1 ?- separar([3,4,-5,-1,0,4,-9],P,N)
2     P = [3,4,0,4], N = [-5,-1,-9].
```

Referências

- Luis, A. M. Palazzo, Introdução à programação prolog, Educat, 1997
- Slides profs. Elaine Faria, Hiran Nonato e Gabriel Coutinho - UFU
- Slides da Profa. Solange - ICMC - USP