Programação em Prolog

Profª Andréa Schwertner Charão
DELC/CT/UFSM

Sumário

- Processo de inferência (execução)
  - Unificação
  - Backtracking (retrocesso)
- Base de conhecimento
  - Estática
  - Dinâmica

Processo de inferência

- A resolução de um problema consiste em satisfazer uma consulta

- Para ilustrar o processo de inferência, consideremos a seguinte base

```
pato(donald).
pato(patolino).
ave(X) :- pato(X).
```

Exemplo de inferência

```
pato(donald).
pato(patolino).
ave(X) :- pato(X).
```

- Consulta/meta: "Donald é um pato?"

```
?- pato(donald).
```

- Busca de predicado pato
- Fato encontrado
- Resposta:

```
?- pato(donald).
true.
```

Exemplo de inferência

- Em SWI-Prolog:
  - Predicado **trace** ajuda a rastrear a execução
  - Predicado **notrace** volta ao modo normal

```
?- trace.
true.

[trace]  ?- pato(donald).
   Call: (6) pato(donald) ? creep
   Exit: (6) pato(donald) ? creep
true.

[trace]  ?- notrace.
true.
```

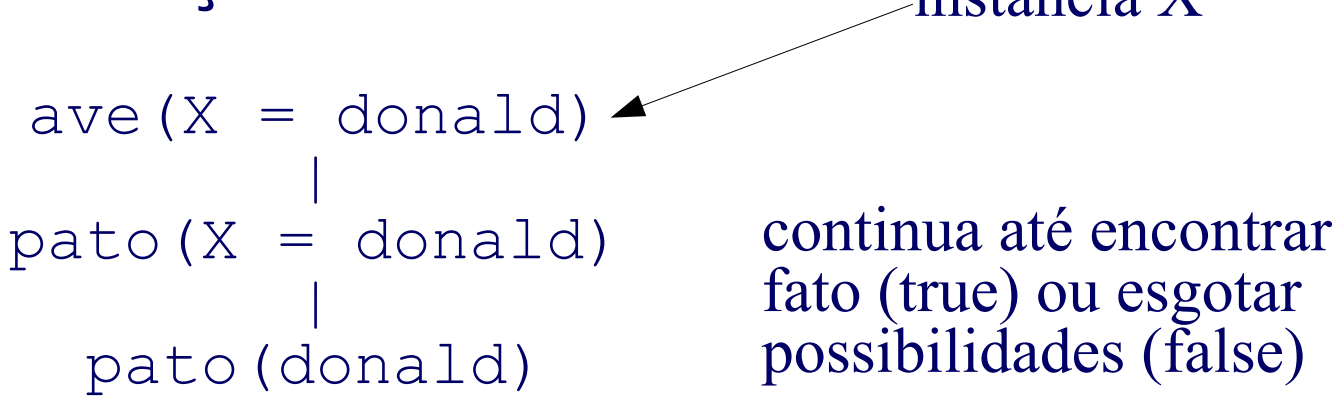
Exemplo de inferência

```
pato(donald).
pato(patolino).
ave(X) :- pato(X).
```

- Meta: "Donald é uma ave?"

```
?- ave(donald).
```

- Busca de predicado ave
- Fato não encontrado, regra encontrada
- **Unificação (matching)** com instanciação de variáveis



Exemplo de inferência

- Em SWI-Prolog:

```
[trace]  ?- ave(donald).
   Call: (6) ave(donald) ? creep
   Call: (7) pato(donald) ? creep
   Exit: (7) pato(donald) ? creep
   Exit: (6) ave(donald) ? creep
true.

[trace]  ?- ave(ze).
   Call: (6) ave(ze) ? creep
   Call: (7) pato(ze) ? creep
   Fail: (7) pato(ze) ? creep
   Fail: (6) ave(ze) ? creep
false.
```

Exemplo de inferência

- Consulta: "Quais são as aves na base?"

```
?- ave(X).
X = donald ;
X = patolino.
```

- **Unificação**: busca de um valor para X que torne o predicado verdadeiro
- **Backtracking** (retrocesso): refaz a unificação mais recente, tentando encontrar outros valores que tornem o predicado verdadeiro

Exemplo de inferência

- Em SWI-Prolog (Redo = backtracking):

```
[trace]  ?- ave(X).
   Call: (6) ave(_G366) ? creep
   Call: (7) pato(_G366) ? creep
   Exit: (7) pato(donald) ? creep
   Exit: (6) ave(donald) ? creep
X = donald ;
   Redo: (7) pato(_G366) ? creep
   Exit: (7) pato(patolino) ? creep
   Exit: (6) ave(patolino) ? creep
X = patolino.
```

Base de conhecimento

- Estática: fatos e regras definidos em arquivo (.pl) carregado no início do programa

- Dinâmica: fatos e regras podem ser inseridos ou removidos durante a execução
  - Predicado dynamic

```
:- dynamic pato/1, ave/1.
pato(donald).
pato(patolino).
ave(X) :- pato(X).
```

Base dinâmica

pato.pl

```
:- dynamic pato/1, ave/1.
pato(donald).
pato(patolino).
ave(X) :- pato(X).
```

```
?- assert(ave(zecarioca)).
true.

?- ave(X).
X = donald ;
X = patolino ;
X = zecarioca.
```