Programação Lógica

PROLOG

Programação Lógica: Prolog

- Programação Lógica: um estilo diferente de programar, muito diferente das linguagens de programação convencionais como C, C++ ou Java
- Os adeptos da Programação Lógica diriam que "diferente" quer dizer mais claro, mais simples e … geralmente melhor!

Programação Lógica: Prolog

- A Linguagem de programação lógica mais utilizada é, de longe, o Prolog (Programming in Logic)
- O Prolog tem sido largamente utilizado no desenvolvimento de aplicações complexas, especialmente no campo da Inteligência Artificial

Programação Lógica: Prolog

- Apesar de ser uma linguagem de programação de aplicação genérica, o seu ponto-forte centra-se na computação simbólica e não na numérica

© IPG-ESTG EI 2020-21 Inteligência Artificial

Programação Lógica: Prolog

- O primeiro aspeto a chamar a atenção no Prolog é o facto de os programas parecerem muito mais simples do que programas equivalentes noutras linguagens

Programação Lógica: Prolog

- Nalgumas linguagens até a simples escrita do 1.º programa de teste mais comum (escrever 'Olá Mundo!') não é fácil.
- Em Prolog é muito simples:

```
write('Olá Mundo!').
```

Programação Lógica: Prolog

- As linguagens de programação convencionais são procedimentais: definem um conjunto de instruções que são executadas umas a seguir às outras, em sequência.
- Os programas Prolog são programas declarativos (apesar de, inevitavelmente, todos terem uma faceta procedimental).

Programação Lógica: Prolog

- Os programas Prolog baseiam-se na Lógica para construir conclusões válidas a partir de factos ou evidências que são disponibilizados.
- São constituídos por apenas 2 tipos de elementos:

  factos e regras

  que o sistema Prolog lê e armazena.

© IPG-ESTG EI 2020-21 Inteligência Artificial

Programação Lógica: Prolog

- O utilizador só tem de fazer perguntas (*queries*) às quais o sistema Prolog responde usando os factos e regras que armazenou.

Programação Lógica: Prolog

- Exemplo: um conjunto de perguntas e respostas sobre animais
- O programa:

```
cao(boby).
cao(pantufa).
cao(milord).
gato(soneca).
gato(tigre).
gato(macaco).
animal(X):- cao(X).
```

Programação Lógica: Prolog

- É constituído por 6 factos:

  boby, pantufa e milord são cães;

  soneca, tigre e macaco são gatos;

```
cao(boby).
cao(pantufa).
cao(milord).
gato(soneca).
gato(tigre).
gato(macaco).
animal(X):- cao(X).
```


Programação Lógica: Prolog

- E por 1 regra:

  qualquer coisa (seja X) é um animal se for um cão.

```
cao(boby).
cao(pantufa).
cao(milord).
gato(soneca).
gato(tigre).
gato(macaco).
animal(X):- cao(X).
```

Programação Lógica: Prolog

- Determinar se boby é um animal envolve uma forma simples de raciocínio lógico:

  Uma vez que

  Qualquer X é um animal se for um cão

  e que

  boby é um cão

  Dedução

  boby é um animal

Programação Lógica: Prolog

- Este tipo de raciocínio é fundamental para programar em Prolog
- Até uma simples pergunta como

  ?- cao(boby).

  pode ser vista como um pedido ao sistema Prolog para provar algo (neste caso que boby é um cão)

Programação Lógica: Prolog

- Ficam assim apresentados todos os elementos (3 no total) necessários para a programação lógica em Prolog: factos, regras e perguntas (*queries*).
- Não há outros: Tudo o resto é construído a partir deles.

Programação Lógica: Prolog

- Sistema Prolog

  http://www.swi-prolog.org

  SWI-Prolog's home

© IPG-ESTG EI 2020-21 Inteligência Artificial

Programação Lógica: Prolog

- ?- *system prompt*
- Indica que o sistema Prolog está à espera que o utilizador introduza uma sequência de um ou mais objetivos (perguntas), seguida de ponto final

- Exemplo:

?-write('Olá Mundo!'),nl,write('Bem-vindo ao Prolog'),nl.

Programação Lógica: Prolog

- Exemplo:

?-write('Olá Mundo!'),nl,write('Bem-vindo ao Prolog'),nl.

- Resultado:

```
Olá Mundo!
Bem-vindo ao Prolog
Yes
```

- Yes: indica que a sequência de objetivos foi bem sucedida

Programação Lógica: Prolog

- Do ponto de vista do sistema, o importante é saber se a sequência de objetivos indicada é ou não bem sucedida
- A restante informação apresentada no ecran é considerada muito menos importante e descrita como um mero efeito secundário

Programação Lógica: Prolog

- Uma sequência de um ou mais objetivos introduzidos pelo utilizador é designada uma pergunta (*querie*)

© IPG-ESTG EI 2020-21 Inteligência Artificial

Termos Prolog

- Termos: alguns exemplos
  - boby
  - cao(milord)
  - X
  - gato(X)

Termos Prolog

- Números:
  - 623
  - -51
  - +20
  - 6.43
  - -.25

Termos Prolog

- Átomos:

  Constantes que não têm valor numérico

  Podem ser escritos de 3 maneiras

1. Qualquer sequência de uma ou mais letras, algarismos e *underscore*, começando com letra minúscula:

   john

   today_is_Wednesday

   a32_BCD

Termos Prolog

- Átomos:

2. Qualquer sequência de caracteres entre plicas (‘ ’), incluindo espaços e letras maiúsculas:

   ‘Today is Wednesday’

   ’32abc’

   ‘today-is-Wednesday’

Termos Prolog

- Átomos:

3. Qualquer sequência de um ou mais caracteres especiais de uma lista que inclui os seguintes + - * / > < = & # @ :

+++

>=

>

+--

Termos Prolog

- Variáveis:
- Qualquer sequência de uma ou mais letras, algarismos e *underscore*, começando por uma letra maiúscula ou por um *underscore*:

X

Author

_123A

_ (variável anónima)

Termos Prolog

- Termos Compostos:
- functor(t_1,t_2,…,t_n), n>=1
  functor: átomo
  n.º de argumentos: aridade
- Exemplos:

  gosta(paulo,prolog)
  ler(X)
  cao(boby)

Termos Prolog

- Listas:
- Uma lista pode ser considerada um termo composto especial
- Podem ter um número de argumentos ilimitado, indicados entre [ ] e separados por vírgulas
- O elemento de uma lista pode ser de qualquer tipo, incluindo outra lista

Termos Prolog

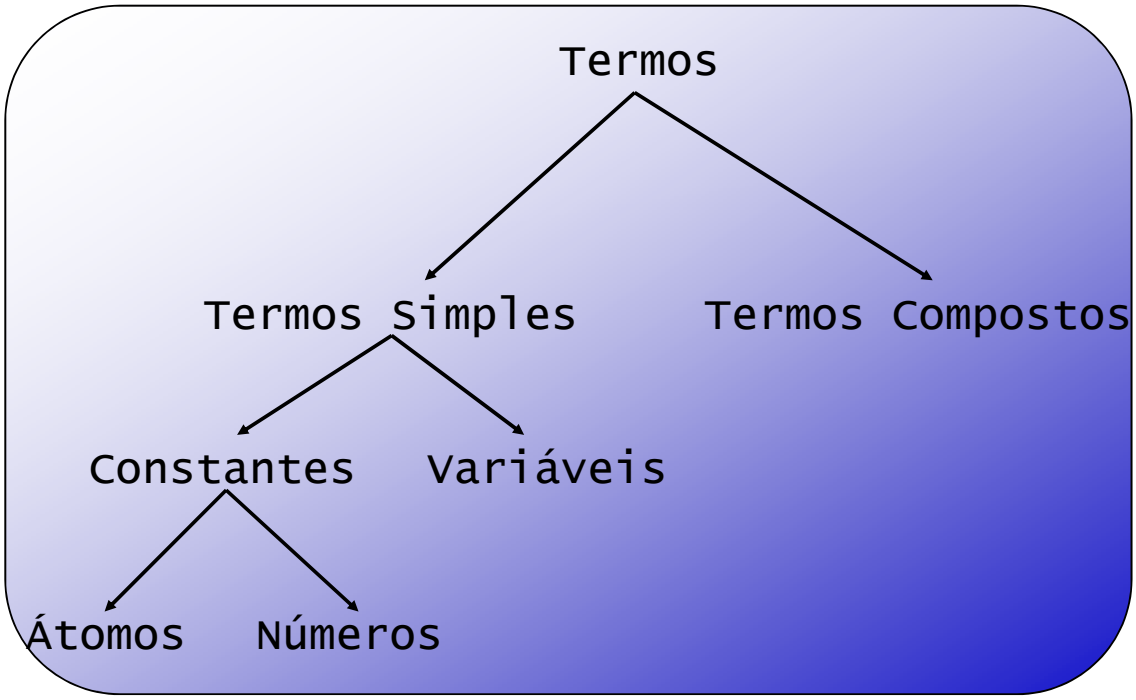
- Listas:
- Exemplos

  [dog, cat, fish, man]

  [[john,28], [mary,56,teacher]]

  [ ] – lista vazia

Termos Prolog - Resumo



Cláusulas e Predicados

- Cláusulas:
- Um programa Prolog consiste numa sucessão de cláusulas
- Uma cláusula pode ser definida em mais do que uma linha, ou podemos ter várias cláusulas definidas na mesma linha
- Uma cláusula termina com um ponto final seguido de pelo menos um espaço branco (espaço ou enter)
- Podemos ter 2 tipos de cláusulas: factos ou regras

Cláusulas e Predicados

- Os factos têm a seguinte forma:

  head.

- head: a cabeça do facto
- Deve ser um átomo ou um termo composto
- Átomos e termos compostos: *call terms*

Cláusulas e Predicados

- As regras têm a seguinte forma:

  $head{:-}t_1,t_2,\ldots t_k.$ (k>=1)

- head: a cabeça do facto ou da regra (deve ser um *call term*)
- **:-** deve ser lido 'se' (*neck operator*)
- $t_1,t_2,\ldots t_k$: o corpo da cláusula (ou regra)

  constituído por um ou mais componentes, separados por vírgulas

  Os componentes são objetivos (*goals*) e as vírgulas devem ser lidas como 'e' (conjunção)

Cláusulas e Predicados

- A regra deve ser lida:
- 'head é verdadeiro se $t_1,t_2,\ldots,t_k$ são todos verdadeiros'
- Exemplos:

```
animal_grande(X):-animal(X),grande(X).
avo(X,Y):-pai(X,Z),pai(Z,Y).
go:-write('Ola mundo'),nl.
```

Cláusulas e Predicados

- Exemplo:

  Definição dos predicados progenitor, pai e mae

```
progenitor(maria,alberto).
progenitor(X,Y):-pai(X,Y).
progenitor(X,Y):-mae(X,Y).
pai(joao,pedro).
mae(ana,pedro).
```

- Este programa tem 5 cláusulas (3 factos e 2 regras)
- Nas 3 primeiras a cabeça (*head*) é um termo composto com functor progenitor e aridade 2 (2 argumentos)

Cláusulas e Predicados

- É possível que o programa inclua cláusulas em que progenitor também é a cabeça mas com aridade diferente:

```
progenitor(joao).
progenitor(X):-filho(X,Y).

/* X é progenitor se X tem um filho Y */
```

Cláusulas e Predicados

- É ainda possível que, no mesmo programa, progenitor seja usado como átomo (predicado sem argumentos), como no seguinte facto:

```
animal(progenitor).
```


Cláusulas e Predicados

- As cláusulas anteriores definem 2 predicados com o nome progenitor, um com aridade 2 e outro com aridade 1.
- Para os distinguirmos, podemos indicar

```
progenitor/2
progenitor/1
```

Cláusulas e Predicados

- No exemplo seguinte podemos identificar um predicado sem argumentos: go/0

```
go:-progenitor(joao,B),
      write('João tem um filho chamado '),
      write(B),nl.
```


Cláusulas e Predicados

- Interpretação de Regras:

- As regras têm uma interpretação declarativa e uma interpretação procedimental

```
apanha(X,Y):-cao(X),gato(Y),
      write(X),write(' apanha '),write(Y),nl.
```

Cláusulas e Predicados

- Interpretação Declarativa:

  "apanha(X,Y) é verdade se
  cao(X) é verdade e
  gato(Y) é verdade e
  write(X) é verdade e
  etc …"

```
apanha(X,Y):-cao(X),gato(Y),
      write(X),write(' apanha '),write(Y),nl.
```


Cláusulas e Predicados

- Interpretação Procedimental:

  "Para satisfazer apanha(X,Y),
  primeiro satisfazer cao(X),
  depois satisfazer gato(Y),
  depois satisfazer write(X),
  etc …"

```
apanha(X,Y):-cao(X),gato(Y),
      write(X),write(' apanha '),write(Y),nl.
```

Cláusulas e Predicados

- Os factos são de um modo geral interpretados declarativamente:

```
cao(boby).
```

é lido "boby é um cão".

Cláusulas e Predicados

- A ordem das cláusulas que definem um predicado e a ordem dos objetivos (*goals*) no corpo de cada regra são irrelevantes para a interpretação declarativa, mas de fundamental importância para a interpretação procedimental.

© IPG-ESTG EI 2020-21 Inteligência Artificial

Cláusulas e Predicados

- As cláusulas na base de dados do sistema Prolog são examinadas de cima para baixo.
- Se necessário, os objetivos no corpo de uma regra são examinados da esquerda para a direita.

Cláusulas e Predicados

- O programa do utilizador é constituído por factos e regras que definem novos predicados: predicados *user-defined*.
- Os predicados pré-definidos no Sistema Prolog designam-se *built-in predicates* (BIP). Exemplos:

  write/1, nl/0, repeat/0, member/2, append/3, consult/1, halt/0, …

Cláusulas e Predicados

- Recursividade

  É possível utilizar definições recursivas: definir predicados em função deles próprios.
- Recursividade direta:

  Predicado pred1 é definido em função dele próprio.
- Recursividade indireta:

  Predicado pred1 é definido por intermédio de pred2, que por sua vez é definido em função de pred3, … que é definido utilizando pred1.

Cláusulas e Predicados

- Exemplo:

```
gosta(joao,X):-gosta(X,Y),cao(Y).
```

pode ser interpretado como

"João gosta de qualquer um
que goste pelo menos de um cão"

Cláusulas e Predicados

- Variáveis
- Variáveis em objetivos: pode ser interpretado como querendo significar "encontrar valores das variáveis que satisfaçam o objetivo".

```
?- animal_grande(A).
```

- Este objetivo pode ser lido como "encontrar um valor para A de modo que animal_grande(A) seja satisfeito".

Cláusulas e Predicados

- Atribuição de valores a variáveis
- Inicialmente, nenhuma variável numa cláusula tem valor atribuído.
- Quando o Sistema Prolog avalia o objetivo, a algumas variáveis pode ser atribuído um valor.
- Uma variável que tenha recebido um valor pode ficar novamente sem valor atribuído e poderá voltar a receber um valor diferente com o processo de retrocesso do Sistema Prolog (*backtracking*).

Cláusulas e Predicados

- Âmbito lexical de variáveis
- Numa cláusula como

```
progenitor(X,Y):-pai(X,Y).
```

as variáveis X e Y não têm qualquer relação com quaisquer outras variáveis com o mesmo nome utilizadas noutra situação.

- O âmbito lexical de uma variável é a cláusula em que ela aparece.

Cláusulas e Predicados

- Variáveis de quantificação universal

- Se uma variável aparece na cabeça de uma regra ou de um facto, significa que a regra ou facto se aplica para todos os possíveis valores da variável.

Cláusulas e Predicados

- Por exemplo, a regra:

```
animal_grande(X):-cao(X),grande(X).
```

pode ser lida como "para todos os valores de X, X é um animal grande se X é um cão e se X é grande".

- A variável X é dita ser quantificada universalmente.

Cláusulas e Predicados

- Variáveis de quantificação existencial
- Suponhamos as seguintes cláusulas:

```
pessoa(joana,borges,feminino,28,arquitecta).
pessoa(francisco,silva,masculino,62,medico).
pessoa(paulo,sousa,masculino,45,canalizador).
pessoa(martim,santos,masculino,23,quimico).
pessoa(maria,silva,feminino,24,programadora).
pessoa(martim,ribeiro,masculino,47,solicitador).
homem(A):-pessoa(A,B,masculino,C,D).
```

Cláusulas e Predicados

- As primeiras 6 cláusulas constituem a definição do predicado pessoa/5

```
pessoa(joana,borges,feminino,28,arquitecta).
pessoa(francisco,silva,masculino,62,medico).
pessoa(paulo,sousa,masculino,45,canalizador).
pessoa(martim,santos,masculino,23,quimico).
pessoa(maria,silva,feminino,24,programadora).
pessoa(martim,ribeiro,masculino,47,solicitador).
homem(A):-pessoa(A,B,masculino,C,D).
```


Cláusulas e Predicados

- A última cláusula é uma regra: "para todo o A, A é um homem se A é uma pessoa cujo género é masculino", para pelo menos um valor das variáveis B, C e D

```
pessoa(joana,borges,feminino,28,arquitecta).
pessoa(francisco,silva,masculino,62,medico).
pessoa(paulo,sousa,masculino,45,canalizador).
pessoa(martim,santos,masculino,23,quimico).
pessoa(maria,silva,feminino,24,programadora).
pessoa(martim,ribeiro,masculino,47,solicitador).
homem(A):-pessoa(A,B,masculino,C,D).
```

Cláusulas e Predicados

- Ou seja,

  Para todo o A, A é um homem se existir uma pessoa com nome A, apelido B, género masculino, idade C e ocupação D, para pelo menos um valor de B, C e D.

```
?- homem(paulo).
yes
```


Cláusulas e Predicados

- As variáveis B, C e D não aparecem na cabeça do predicado homem: são variáveis de quantificação existencial.

- A variável A é de quantificação universal.

Cláusulas e Predicados

- A variável anónima
- Para saber se existe uma cláusula que corresponda a alguém chamado paulo na base de dados, bastaria o seguinte:

```
?- pessoa(paulo,Apelido,Sexo,Idade,Ocupacao).
Apelido=sousa,
Sexo=masculino,
Idade=45,
Ocupacao=canalizador
```


Cláusulas e Predicados

- Em muitos casos, pode não ser importante saber o valor de todas as variáveis.
- Se basta saber se existe ou não alguém chamado paulo, seria mais simples fazer:

```
?- pessoa(paulo,_,_,_,_).
yes
```

Cláusulas e Predicados

- O caracter _ (*underscore*) denota uma variável especial: a variável anónima
- É utilizada quando o valor da variável não é necessário.

Cláusulas e Predicados

- Outros exemplos:

```
?- pessoa(paulo,Apelido,_,_,_).
Apelido=sousa

?- pessoa(martim,_,_,Idade,_).
Idade=23;
Idade=47

?- pessoa(martim,X,X,Idade,X).
no
```

Unificação

- Unificação é o processo que efetua a correspondência entre termos e variáveis
- Prolog unifica

mulher(X)

com

mulher(maria)

instanciando a variável **X** com o átomo **maria**.

Unificação

- Definição:
  - Dois termos podem ser unificados se são o mesmo termo ou se contêm variáveis que podem ser uniformemente instanciadas por termos de tal modo que os termos resultantes são iguais

Unificação

- Isto significa que:
  - **maria** e **maria** unificam
  - **42** e **42** unificam
  - **mulher(maria)** e **mulher(maria)** unificam
- Também significa que:
  - **vicente** e **maria** não unificam
  - **mulher(maria)** e **mulher(ana)** não unificam

Unificação

- E o que é que acontece com os termos:
  - **maria** e **X**

Unificação

- E o que é que acontece com os termos:
  - **maria** e **X**
  - **mulher(Z)** e **mulher(maria)**

Unificação

- E o que é que acontece com os termos:
  - **maria** e **X**
  - **mulher(Z)** e **mulher(maria)**
  - **gosta(maria,X)** e **gosta(X,vicente)**

© IPG-ESTG EI 2020-21 Inteligência Artificial

Instanciações

- Quando o Sistema Prolog unifica dois termos realiza todas as instanciações necessárias, de modo que os termos fiquem iguais

- Isto torna a unificação um poderoso mecanismo de programação

Definição (revisão) 1/3

1. Se T_1 e T_2 forem constantes, então T_1 e T_2 podem ser unificados se forem o mesmo átomo ou o mesmo número.

Definição (revisão) 2/3

1. Se T_1 e T_2 forem constantes, então T_1 e T_2 podem ser unificados se forem o mesmo átomo ou o mesmo número.
2. Se T_1 for uma variável e T_2 um termo de qualquer tipo, então T_1 e T_2 unificam, sendo T_1 instanciado com T_2 (e vice-versa).

Definição (revisão) 3/3

1. Se T_1 e T_2 forem constantes, então T_1 e T_2 podem ser unificados se forem o mesmo átomo ou o mesmo número.
2. Se T_1 for uma variável e T_2 um termo de qualquer tipo, então T_1 e T_2 unificam, sendo T_1 instanciado com T_2 (e vice-versa)
3. Se T_1 e T_2 forem termos compostos podem ser unificados se:
   a) tiverem os mesmos functor e aridade, e
   b) todos os argumentos correspondentes puderem ser unificados, e
   c) as instanciações de variáveis são compatíveis.

Unificação em Prolog: =/2

?- maria = maria.

yes

?-

Unificação em Prolog: =/2

?- maria = maria.

yes

?- maria = vicente.

no

?-

Unificação em Prolog: =/2

?- maria = X.

X=maria

yes

?-

Qual a resposta ?

?- X=maria, X=vicente.

Qual a resposta ?

?- X=maria, X=vicente.

no

?-

Porquê?

Qual a resposta ?

?- X=maria, X=vicente.

no

?-

Porquê?

Depois de avaliar o primeiro objetivo, o Sistema Prolog instanciou X com **maria**, de modo que depois já não a pode unificar com **vicente**. Assim, o segundo objetivo falha.

Termos Compostos: Exemplo

?- k(s(g),Y) = k(X,t(k)).

Termos Compostos: Exemplo

?- k(s(g),Y) = k(X,t(k)).

X=s(g)

Y=t(k)

yes

?-

Termos Compostos: Exemplo

?- k(s(g),t(k)) = k(X,t(Y)).

Termos Compostos: Exemplo

?- k(s(g),t(k)) = k(X,t(Y)).

X=s(g)

Y=k

yes

?-

Mais um exemplo

?- gosta(X,X) = gosta(joao,maria).

Listas

- Uma lista é uma sequência finita de elementos
- Exemplos de listas em Prolog:

[mia, vincent, jules, yolanda]
[mia, robber(honeybunny), X, 2, mia]
[ ]
[mia, [vincent, jules], [butch, friend(butch)]]
[[ ], dead(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

Listas

- Os elementos de uma lista representam-se entre parêntesis retos
- O comprimento (*length*) de uma lista é o número de elementos que a constituem
- Todos os tipos de termos Prolog podem ser elementos de uma lista
- A lista vazia, [ ], é uma lista especial

Listas: Cabeça e cauda

- Uma lista não vazia é constituída por duas partes
  - A cabeça (*head*)
  - A cauda (*tail*)
- A cabeça é o primeiro elemento da lista
- A cauda é toda a parte restante
  - A cauda é a lista que se obtém depois de se retirar o primeiro elemento (a cabeça)
  - A cauda de uma lista é sempre uma lista

Cabeça e cauda: Exemplo 1

- [mia, vincent, jules, yolanda]

  Cabeça:
  Cauda:

Cabeça e cauda: Exemplo 1

- [mia, vincent, jules, yolanda]

  Cabeça: mia
  Cauda:

Cabeça e cauda: Exemplo 1

- [mia, vincent, jules, yolanda]

  Cabeça: mia
  Cauda: [vincent, jules, yolanda]

Cabeça e cauda: Exemplo 2

- [[ ], dead(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

  Cabeça:
  Cauda:

Cabeça e cauda: Exemplo 2

- [[ ], dead(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

  Cabeça: [ ]
  Cauda:

Cabeça e cauda: Exemplo 2

- [[ ], dead(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

  Cabeça: [ ]
  Cauda: [dead(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

© IPG-ESTG EI 2020-21 Inteligência Artificial

Cabeça e cauda: Exemplo 3

- [dead(z)]

  Cabeça:
  Cauda:

Cabeça e cauda: Exemplo 3

- [dead(z)]

  Cabeça: dead(z)
  Cauda:

Cabeça e cauda: Exemplo 3

- [dead(z)]

  Cabeça: dead(z)
  Cauda: [ ]

Cabeça e cauda da lista vazia

- A lista vazia não tem cabeça nem cauda
- Para o Sistema Prolog, [ ] é uma lista simples especial, sem qualquer estrutura interna
- A lista vazia tem um papel importante para os predicados recursivos no processamento de listas em Prolog

O operador |

- O Prolog tem um operador especial (*built-in*), |, que pode ser utilizado para decompor uma lista nas respetivas cabeça e cauda
- O operador | é uma facilidade-chave na definição de predicados Prolog para manipulação de listas

O operador |

```
?- [Head|Tail] = [mia, vincent, jules, yolanda].

Head = mia
Tail = [vincent,jules,yolanda]
yes

?-
```

O operador |

```
?- [X|Y] = [mia, vincent, jules, yolanda].

X = mia
Y = [vincent,jules,yolanda]
yes

?-
```


O operador |

```
?- [X|Y] = [ ].

no

?-
```

O operador |

```
?- [X,Y|Tail] = [[ ], dead(z), [2, [b,c]], [], Z, [2, [b,c]]] .

X = [ ]
Y = dead(z)
Tail = [[2, [b,c]], [ ], Z, [2, [b,c]]]
yes

?-
```


A Variável Anónima

- Suponhamos que estamos interessados no 2.º e no 4.º elementos de uma lista

```
?- [X1,X2,X3,X4|Tail] = [mia, vincent, marsellus, jody, yolanda].
X1 = mia
X2 = vincent
X3 = marsellus
X4 = jody
Tail = [yolanda]
yes

?-
```

A Variável Anónima

- Há uma maneira mais simples de obter apenas a informação desejada:

```
?- [ _,X2, _,X4|_ ] = [mia, vincent, marsellus, jody, yolanda].
X2 = vincent
X4 = jody
yes

?-
```

- O *underscore* é a variável anónima

A Variável Anónima

- Utilizada quando é necessário recorrer a uma variável, não interessando aquilo que o Sistema Prolog lhe instancia
- Cada ocorrência da variável anónima é independente, i.e. pode ser sujeita a algo diferente

© IPG-ESTG EI 2020-21 Inteligência Artificial

Elemento de uma Lista

- Uma das coisas mais comum que podemos querer saber sobre uma lista é verificar se algo é ou não um seu elemento
- Assim, consideremos um predicado que, dados um termo **X** e uma lista **L**, nos indica se **X** pertence ou não a **L**
- Este predicado costuma designar-se por member/2

member/2

```
member(X,[X|T]).
member(X,[H|T]):- member(X,T).
```

```
?-
```

member/2

```
member(X,[X|T]).
member(X,[H|T]):- member(X,T).
```

```
?- member(yolanda,[yolanda,trudy,vincent,jules]).
```


member/2

```
member(X,[X|T]).
member(X,[H|T]):- member(X,T).
```

```
?- member(yolanda,[yolanda,trudy,vincent,jules]).
yes
?-
```

member/2

```
member(X,[X|T]).
member(X,[H|T]):- member(X,T).
```

```
?- member(vincent,[yolanda,trudy,vincent,jules]).
```


member/2

```
member(X,[X|T]).
member(X,[H|T]):- member(X,T).
```

```
?- member(vincent,[yolanda,trudy,vincent,jules]).
yes
?-
```

member/2

```
member(X,[X|T]).
member(X,[H|T]):- member(X,T).
```

```
?- member(zed,[yolanda,trudy,vincent,jules]).
```


member/2

```
member(X,[X|T]).
member(X,[H|T]):- member(X,T).
```

```
?- member(zed,[yolanda,trudy,vincent,jules]).
no
?-
```

member/2

```
member(X,[X|T]).
member(X,[H|T]):- member(X,T).
```

```
?- member(X,[yolanda,trudy,vincent,jules]).
```


member/2

```
member(X,[X|T]).
member(X,[H|T]):- member(X,T).
```

```
?- member(X,[yolanda,trudy,vincent,jules]).
X = yolanda;
X = trudy;
X = vincent;
X = jules;
no
```

member/2

```
member(X,[X|T]).
member(X,[H|T]):- member(X,T).
```

```
?- member(X,[ ]).
no
```


member/2 (outra definição)

```
member(X,[X|_]).
member(X,[_|T]):- member(X,T).
```

Recursividade no tratamento de Listas

- O predicado member/2 foi definido utilizando recursividade
  - é realizada determinada operação à cabeça da lista e, depois,
  - é realizada, de forma recursiva, a mesma operação à cauda
- Esta técnica é muito comum em Prolog, pelo que será importante dominá-la
- Vejamos outro exemplo …

Exemplo: a2b/2

- O predicado a2b/2 recebe duas listas como argumentos e terá sucesso:
  - se o 1.º argumento for uma lista de a's e,
  - se o 2.º argumento for uma lista de b's com a mesma dimensão da 1.ª lista

```
?- a2b([a,a,a,a],[b,b,b,b]).
yes
?- a2b([a,a,a,a],[b,b,b]).
no
?- a2b([a,c,a,a],[b,b,b,t]).
no
```

Definição de a2b/2: passo 1

```
a2b([],[]).
```

- Geralmente, a melhor maneira de resolver problemas destes é começar por considerar o caso mais simples
- Aqui significa: a lista vazia

Definição de a2b/2: passo 2

```
a2b([],[]).
a2b([a|L1],[b|L2]):- a2b(L1,L2).
```

- Agora devemos raciocinar recursivamente!
- Quando deverá a2b/2 decidir que duas listas não vazias são uma lista de a's e uma lista de b's com a mesma dimensão?

Teste de a2b/2

```
a2b([],[]).
a2b([a|L1],[b|L2]):- a2b(L1,L2).
```

```
?- a2b([a,a,a],[b,b,b]).
yes
?-
```


Teste de a2b/2

```
a2b([],[]).
a2b([a|L1],[b|L2]):- a2b(L1,L2).
```

```
?- a2b([a,a,a,a],[b,b,b]).
no
?-
```

Teste de a2b/2

```
a2b([],[]).
a2b([a|L1],[b|L2]):- a2b(L1,L2).
```

```
?- a2b([a,t,a,a],[b,b,b,c]).
no
?-
```


Outros testes de a2b/2

```
a2b([],[]).
a2b([a|L1],[b|L2]):- a2b(L1,L2).
```

```
?- a2b([a,a,a,a,a], X).
X = [b,b,b,b,b]
yes
?-
```

Outros testes de a2b/2

```
a2b([],[]).
a2b([a|L1],[b|L2]):- a2b(L1,L2).
```

```
?- a2b(X,[b,b,b,b,b,b,b]).
X = [a,a,a,a,a,a,a]
yes
?-
```


Aritmética em Prolog

- O Prolog fornece um conjunto de operações aritméticas básicas
- Números inteiros e números reais

Aritmética	Prolog
2 + 3 = 5	?- 5 is 2+3.
3 x 4 = 12	?- 12 is 3*4.
5 – 3 = 2	?- 2 is 5-3.
3 – 5 = -2	?- -2 is 3-5.
4 : 2 = 2	?- 2 is 4/2.
1 - resto da divisão inteira de 7 por 2	?- 1 is mod(7,2).
3 - quociente da divisão inteira de 7 por 2	?- 3 is 7//2

Aritmética: Exemplos

```
?- 10 is 5+5.
yes

?- 4 is 2+3.
no

?- X is 3 * 4.
X=12
yes

?- R is mod(7,2).
R=1
yes
```


Aritmética: Definição de predicados

```
addThreeAndDouble(X, Y):-
    Y is (X+3) * 2.
```

Aritmética: Definição de predicados

```
addThreeAndDouble(X, Y):-
    Y is (X+3) * 2.
```

```
?- addThreeAndDouble(1,X).
X=8
yes

?- addThreeAndDouble(2,X).
X=10
yes
```


Operadores Aritméticos

- Importante: os operadores +, -, / e * não calculam qualquer resultado aritmético em Prolog
- Expressões como 3+2, 4-7, 5/5 são apenas termos Prolog
  - Functor: +, -, /, *
  - Aridade: 2
  - Argumentos: inteiros

Operadores Aritméticos

```
?- X = 3 + 2.
```


Operadores Aritméticos

```
?- X = 3 + 2.
X = 3+2
yes

?-
```

Operadores Aritméticos

```
?- X = 3 + 2.
X = 3+2
yes

?- 3 + 2 = X.
```


Operadores Aritméticos

```
?- X = 3 + 2.
X = 3+2
yes

?- 3 + 2 = X.
X = 3+2
yes

?-
```

© IPG-ESTG EI 2020-21 Inteligência Artificial

O Predicado is/2

- Para forçar o Sistema Prolog a efetuar o cálculo de expressões aritméticas, temos de usar o predicado

  is

  tal como vimos em exemplos anteriores
- Representa uma indicação para o Sistema Prolog efetuar cálculos
- Uma vez que não é um predicado Prolog comum, é necessário ter em conta algumas restrições

O Predicado is/2

```
?- X is 3 + 2.
```

© IPG-ESTG EI 2020-21 Inteligência Artificial

O Predicado is/2

```
?- X is 3 + 2.
X = 5
yes

?-
```


O Predicado is/2

```
?- X is 3 + 2.
X = 5
yes

?- 3 + 2 is X.
```

© IPG-ESTG EI 2020-21 Inteligência Artificial

O Predicado is/2

```
?- X is 3 + 2.
X = 5
yes

?- 3 + 2 is X.
ERROR: is/2: Arguments are not sufficiently instantiated

?-
```


O Predicado is/2

```
?- X is 3 + 2.
X = 5
yes

?- 3 + 2 is X.
ERROR: is/2: Arguments are not sufficiently instantiated

?- Result is 2+2+2+2+2.
```

© IPG-ESTG EI 2020-21 Inteligência Artificial

O Predicado is/2

```
?- X is 3 + 2.
X = 5
yes

?- 3 + 2 is X.
ERROR: is/2: Arguments are not sufficiently instantiated

?- Result is 2+2+2+2+2.
Result = 10
yes

?-
```


Restrições na utilização de is/2

- Podem ser utilizadas variáveis no lado direito do predicado **is**
- Mas quando o Sistema Prolog efetua a avaliação, as variáveis já devem ter sido instanciadas por um termo Prolog sem variáveis
- Esse termo Prolog deve ser uma expressão aritmética

Notação

- Duas notas finais sobre expressões aritméticas
  - 3+2, 4/2, 4-5 são apenas termos Prolog numa notação mais amigável do utilizador: **3+2** é, na realidade, **+(3,2)**
  - O predicado **is** também é um predicado Prolog de aridade 2

Notação

- Duas notas finais sobre expressões aritméticas
  - 3+2, 4/2, 4-5 são apenas termos Prolog numa notação mais amigável do utilizador: **3+2** é, na realidade, **+(3,2)**
  - O predicado **is** também é um predicado Prolog de aridade 2

```
?- is(X,+(3,2)).
X = 5
yes
```

© IPG-ESTG EI 2020-21 Inteligência Artificial

Listas e Aritmética

- Qual o tamanho de uma lista ?
  - A lista vazia tem dimensão: zero;
  - Uma lista não-vazia tem dimensão: um mais dimensão da sua cauda.

Dimensão de uma Lista

```
length([],0).
length([_|L],N):-
    length(L,X),
    N is X + 1.
```

```
?-
```

Dimensão de uma Lista

```
length([],0).
length([_|L],N):-
    length(L,X),
    N is X + 1.
```

```
?- length([a,b,c,d,e,[a,x],t],X).
```


Dimensão de uma Lista

```
length([],0).
length([_|L],N):-
    length(L,X),
    N is X + 1.
```

```
?- length([a,b,c,d,e,[a,x],t],X).
X=7
yes
?-
```

Inverter uma Lista em Prolog

```
?- reverse([a,b,c,d],L).
L=[d,c,b,a]
yes
?- reverse(L,[a,b,c,d]).
L=[d,c,b,a]
yes
?-
```


Acrescentar elementos a uma Lista

```
append([], L, L).
append([H|L1], L2, [H|L3]):-
    append(L1, L2, L3).
```

```
?- append([1,2,3,4],[5,6,7,8,9],L).
L=[1,2,3,4,5,6,7,8,9]
yes
?- append([],[a,b,c,d],L).
L=[a,b,c,d]
yes
?-
```

Operadores Relacionais

- Alguns predicados Prolog efetuam eles próprios cálculos aritméticos
- São exemplo os operadores relacionais

Operadores Relacionais

Aritmética	Prolog
$x < y$	X < Y
$x \leq y$	X =< Y
$x = y$	X =:= Y
$x \neq y$	X =\= Y
$x \geq y$	X >= Y
$x > y$	X > Y

© IPG-ESTG EI 2020-21 Inteligência Artificial

Operadores Relacionais

- Têm o significado óbvio
- Forçam a avaliação dos dois argumentos (da direita e da esquerda)

```
?- 2 < 4+1.
yes

?- 4+3 > 5+5.
no
```


Operadores Relacionais

- Têm o significado óbvio
- Forçam a avaliação dos dois argumentos (da direita e da esquerda)

```
?- 4 = 4.
yes

?- 2+2 = 4.
no

?- 2+2 =:= 4.
yes
```

© IPG-ESTG EI 2020-21 Inteligência Artificial

Operadores de Igualdade: Resumo

- Há 3 tipos de operadores para testar igualdade

=	Predicado de Unificação Predicado de identidade com unificação
\=	Negação do predicado de unificação
==	Predicado de Identidade Predicado de identidade sem unificação
\==	Negação do predicado de identidade
=:=	Predicado de Igualdade Aritmética
=\=	Negação da Igualdade Aritmética

Operadores de Igualdade: Resumo

- Igualdade aritmética (=:=)

```
?- 6+4 =:= 6*3-8.
yes

?- 10 =\= 8+3.
yes

?-
```

Operadores de Igualdade: Resumo

- Igualdade de termos (==)

```
?- 6+4 \== 3+7.
yes

?- likes(X,prolog) == likes(Y,prolog).
no

?-
```


Operadores de Igualdade: Resumo

- Igualdade de termos com unificação (=)

```
?- 6+X = 6+3.
X=3
yes

?- likes(X,prolog) \= likes(john,Y).
no

?-
```

Operadores Lógicos

not	O operador de negação
	cao(milord). ?- not cao(milord). no
;	O operador de disjunção
	?- 6<3; 7 is 5+2. yes

Entrada e Saída (IO)

- write/1

  escrita de termos no dispositivo de saída
- nl/0

  escrita de linha em branco (*new line*)

```
?- write(26),nl.
26
yes

?-
```

Entrada e Saída (IO)

- read/1

  leitura (entrada) de termos

```
?- read(X).
: jim.
X=jim

?-
```


Entrada e Saída (IO)

- put/1

  escrita de um caracter no dispositivo de saída

  (valor ASCII: 0 – 255)

```
?- put(97),nl.
a
yes

?-
```

Entrada e Saída (IO)

- get0/1

  leitura de um único caracter

```
?- get0(N).
: a
N = 97

?-
```


Entrada e Saída (IO)

- get/1

  leitura do primeiro caracter não-branco (*non-white-space*)

```
?- get(X).
: Z
X = 90

?- get(M).
:          Z
M = 90
```