

Paradigma Lógico



Prof. Dr. Rafael Giusti
rgiusti@icompu.ufam.edu.br

Programação lógica

Paradigma procedural	Paradigma lógico
Programas são sequências de comandos	Os programas são declarações lógicas
Baseados em procedimentos	Baseado em fatos e regras
Execução equivale à simulação de uma máquina de Turing	Execução consiste em provar que uma declaração lógica é verdadeira
Imperativo: programas dizem como cada comando deve modificar o estado do programa	Declarativo: os programas declaram o quê deve ser provado, não como

Programação lógica

Paradigma funcional	Paradigma lógico
Programas são declarações e chamadas de funções	Não tem funções
Execução equivale à uma chamada de função e a saída é o retorno da função	Execução consiste em provar que uma declaração lógica é verdadeira
Declarativo: os programas declaram o quê deve ser feito, mas não como	Declarativo: os programas declaram o quê deve ser provado, não como

Agenda

- » Lógica formal e programação lógica
- » Programação em PROLOG
- » O conceito de unificação
- » Unificação em mais detalhes
- » Backtracking
- » Trabalhando com listas

Lógica formal

- » A base fundamental do paradigma lógico é a lógica formal (Matemática Discreta, de novo!)
- » Vamos rever alguns conceitos
 - » Proposições
 - » Predicados
 - » Fatos
 - » A proposição condicional ($A \rightarrow B$)
 - » Cláusulas de Horn
 - » Inferência lógica (prova de argumentos)

Proposições

- » Uma **proposição** é uma declaração que deve ser verdadeira *ou* falsa (nunca ambos)
 - » Manaus é uma cidade
 - » Manaus é a capital do Amazonas
 - » $1 + 1 = 3$
 - » Como você está?
 - » Estudem regularmente
 - » Ela é feliz

Proposições

- » Uma **proposição** é uma declaração que deve ser verdadeira *ou* falsa (nunca ambos)
 - » Manaus é uma cidade
 - » Manaus é a capital do Amazonas
 - » $1 + 1 = 3$
 - » ~~Como você está?~~
 - » ~~Estudem regularmente~~
 - » ~~Ela é feliz~~

Simples questionamentos ou conselhos, mesmo que relevantes, não são verdadeiros ou falsos.

Não há contexto para saber quem é "ela". Portanto isto não pode ser uma proposição.

Predicados

- » Linguagens lógicas são baseadas em uma parte da lógica formal chamada **cálculo de predicados**
- » Um **predicado** é uma declaração que pode ou não se aplicar a algum sujeito

Exemplo

Isaac Newton Silva está matriculado em PLP.

Sujeito

Predicado

Predicados

- » Em lógica formal, existe uma relação (função) entre o predicado e o sujeito

$P(x)$: “ x está matriculado em PLP”



x é variável do predicado

O predicado é verdade sse é propriedade do objeto

Predicados

- » Proposições lógicas podem usar predicados
 - » Qual das proposições é verdadeira?

$P(x)$: “ x é o professor de PLP”

$P(Einstein)$: “*Einstein* é o professor de PLP”

$P(Rafael)$: “*Rafael* é o professor de PLP”

Predicados

- » Predicados podem ter múltiplas variáveis

$P(x, y)$: “ x é o professor de y ”

$P(\text{Einstein}, \text{PLP})$: “ Einstein é o professor de PLP ”

$P(\text{Einstein}, \text{Física})$: “ Einstein é o professor de Física ”

$P(\text{Einstein}, \text{Zelda})$: “ Einstein é o professor de Zelda ”

Fatos

- » Predicados associados a constantes podem ser utilizados para declarar fatos
 - » Um **fato** é uma proposição lógica verdadeira

Predicado	Fato declarado
cidade(manaus)	Manaus é uma cidade
capital(amazonas, manaus)	Manaus é a capital do Amazonas
	Amazonas é a capital de Manaus

Proposição condicional

- » Na lógica formal, uma **proposição condicional** assume a forma **antecedente \rightarrow consequente**

$P(x)$: “ x é humano”

$Q(x)$: “ x é mortal”

$P(x) \rightarrow Q(x)$

Se x é humano, então x é mortal.

Cláusula de Horn

- » Uma **cláusula de Horn** é uma proposição na forma $P_1 \wedge P_2 \wedge \dots \wedge P_n \rightarrow Q$

$P(x, y)$: “ x é pai de y ”
 $Q(x, y)$: “ x é avô de y ”

$P(x, y) \wedge P(y, z) \rightarrow Q(x, z)$

Se x é pai de y e y é pai de z , então
 x é avô de z .

Inferência

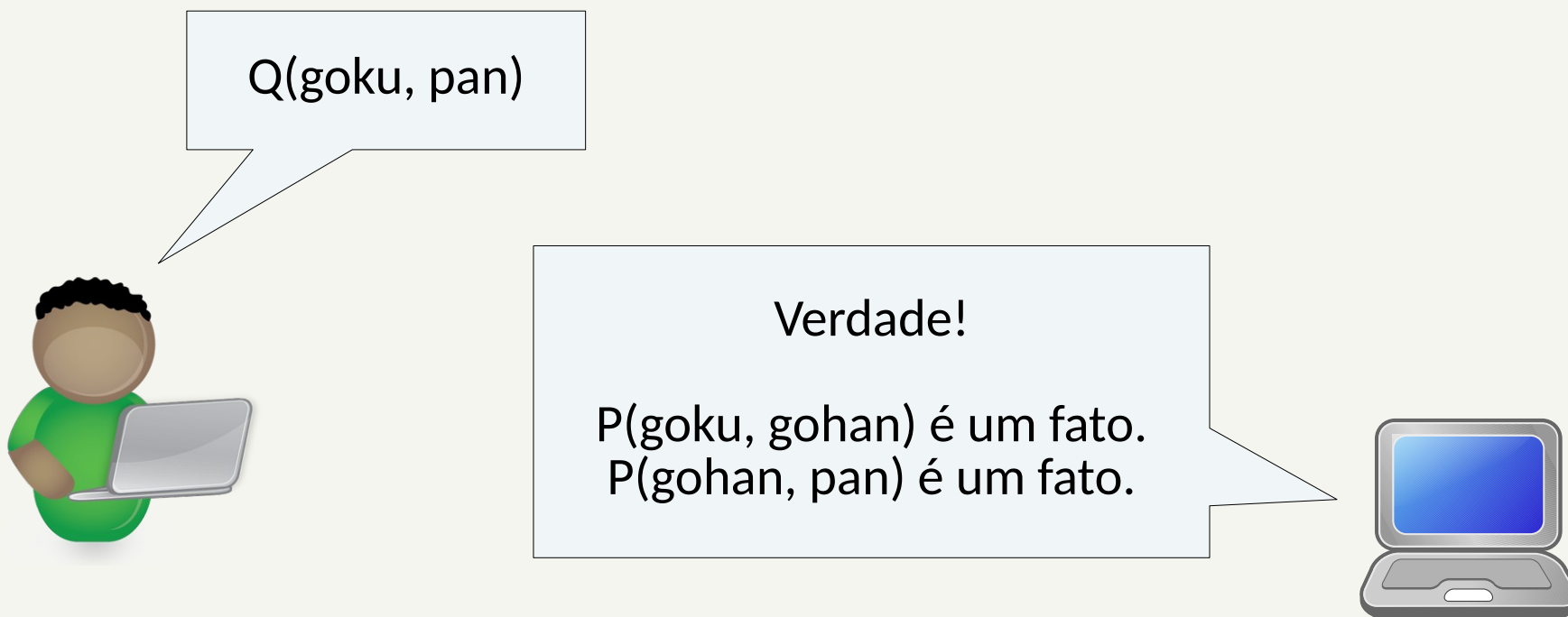
- » As cláusulas de Horn são importantes porque permitem provar proposições através de **inferência**

Fatos:	Cláusulas:
$P(\text{goku}, \text{goohan}).$ $P(\text{goohan}, \text{pan}).$	$P(x, y) \wedge P(y, z) \rightarrow Q(x, z)$

Conclusão: Goku é avô de Pan

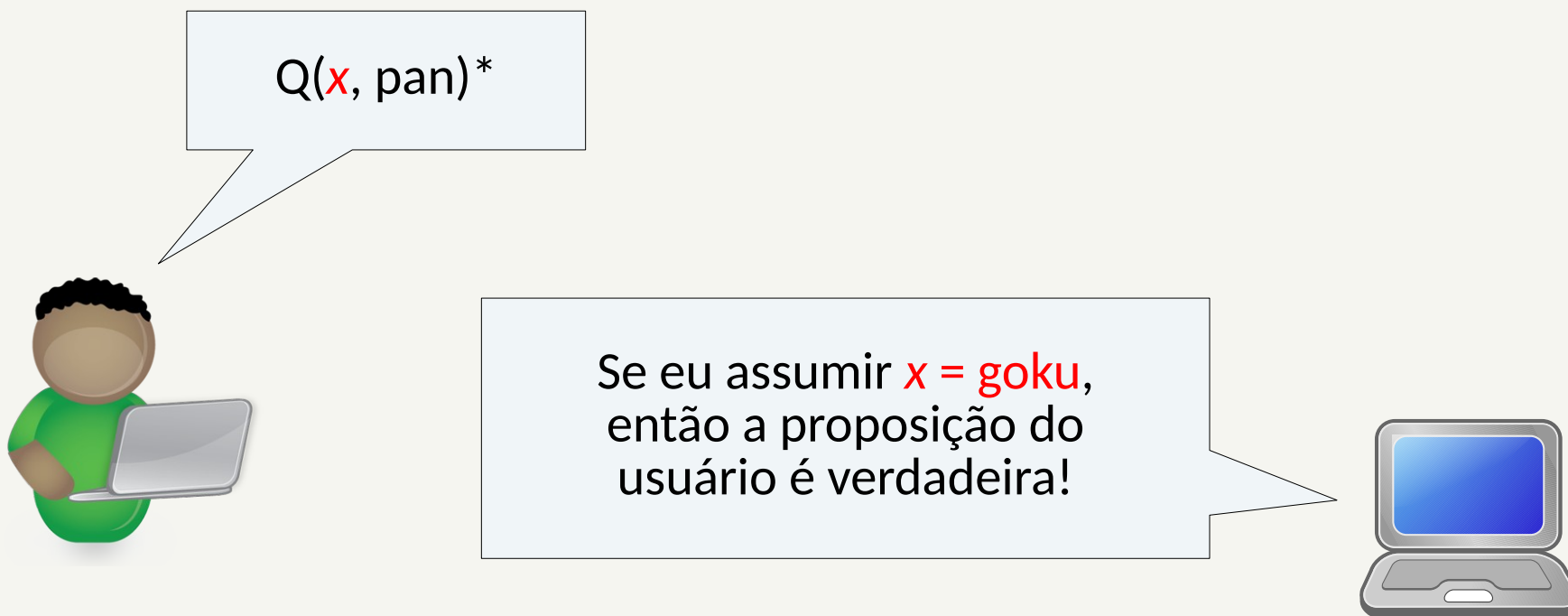
Inferência e objetivos

- » Inferência nos permite cumprir um **objetivo**: provar uma declaração do usuário



Inferência e objetivos

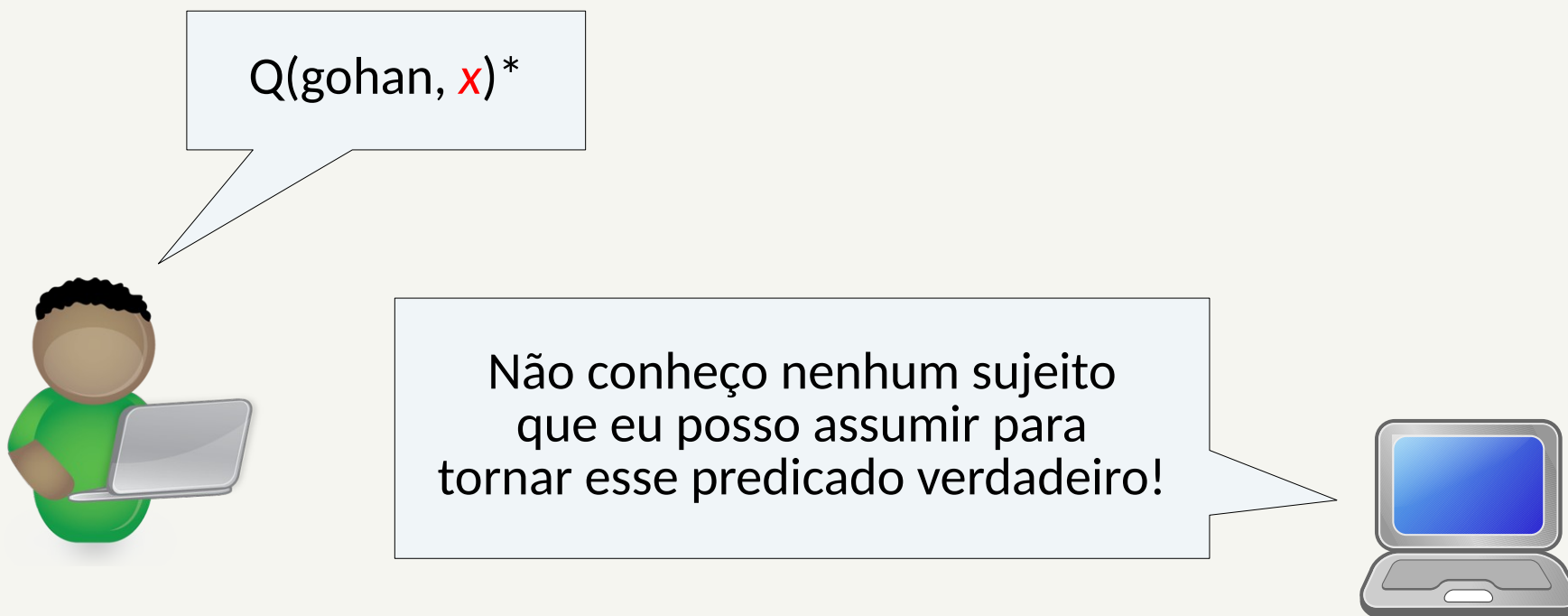
- » Se o usuário apresentar informação parcial, podemos inferir o restante



*Isso equivale a perguntar para o programa
"quem é o avô de Pan?"

Inferência e objetivos

- » Uma informação que não pode ser provada também é um resultado do programa lógico



*Isso equivale a perguntar para o programa
"quem é o neto de Gohan?"

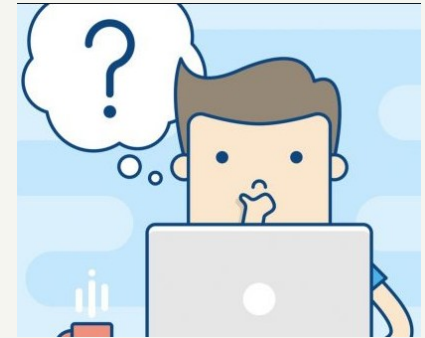
Em síntese...



<https://programadorviking.com.br/>

O papel do **programador** é definir um conjunto de **fatos** e **regras** que descrevem o programa

O papel do **usuário** é apresentar uma **consulta** que contém um ou mais **objetivos**



<https://programadorviking.com.br/>



<https://github.com/SWI-Prolog>

O papel da **máquina de inferência** é **provar os objetivos** e apresentar o resultado ao usuário

Agenda

- » Lógica formal e programação lógica
- » Programação em PROLOG
- » O conceito de unificação
- » Unificação em mais detalhes
- » Backtracking
- » Trabalhando com listas

PROLOG

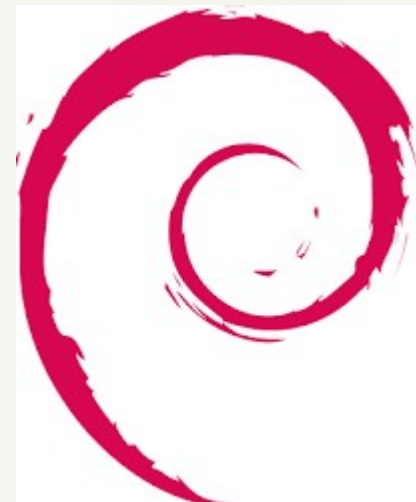
- » O PROLOG foi a primeira linguagem de programação do paradigma lógico
 - » Criado em 1972
 - » Inicialmente utilizado por pesquisadores de Inteligência Artificial
 - ~ Durante as décadas de 1970 e 1980 houve grande interesse em desenvolver sistemas capazes de provar teoremas
 - ~ Linguagens lógicas também foram bastante utilizadas para processamento de linguagem natural

PROLOG

- » Um programa em PROLOG é uma **base de conhecimento**
 - » Na base de conhecimento, o programador define predicados (fatos e regras)
 - » O programador também pode definir um ou mais predicados para servir de interface para o usuário
 - » O usuário fará uma declaração, que pode ser um objetivo ou uma consulta
 - » O interpretador fará a prova

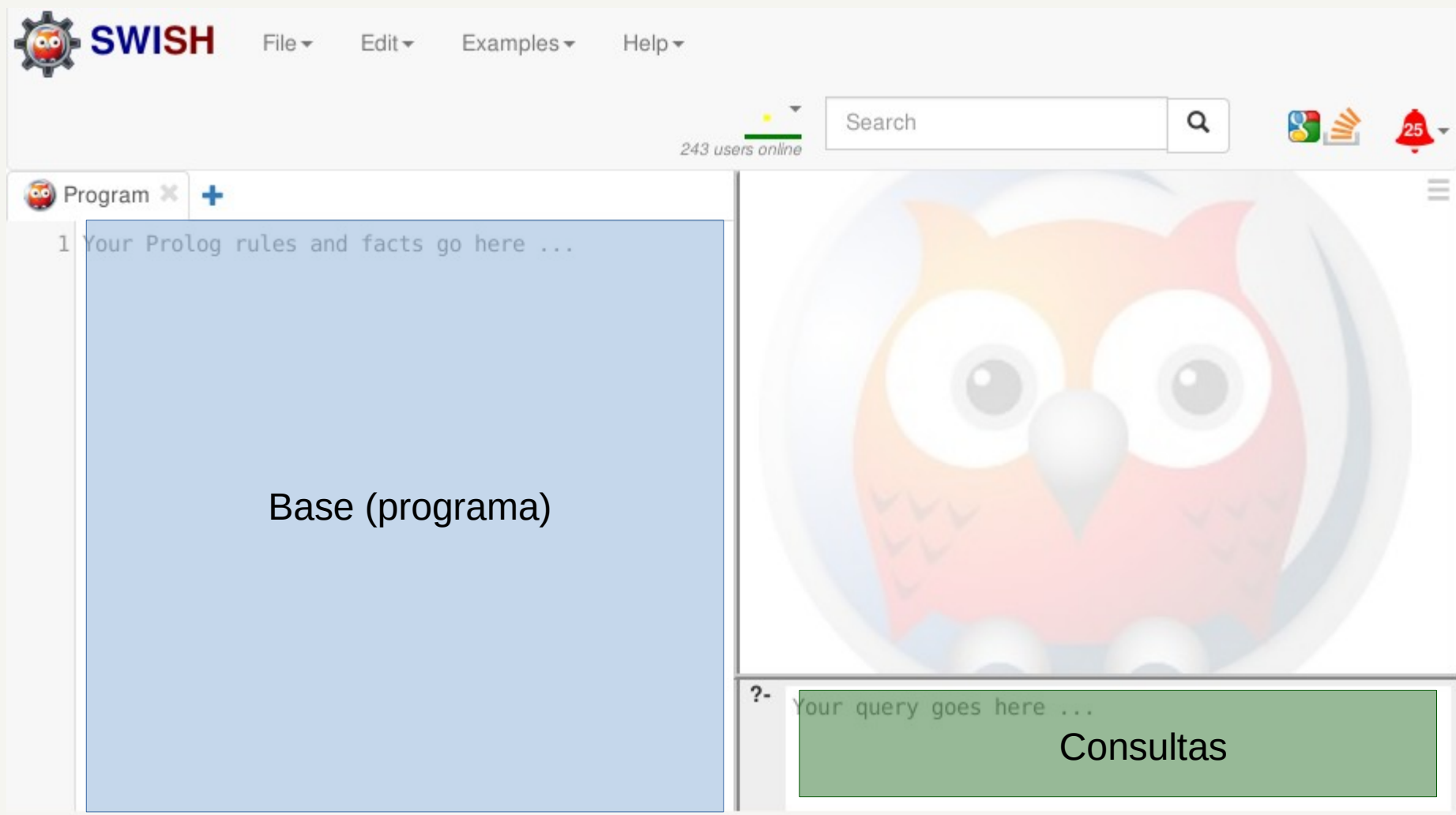
SWI-PROLOG

- » Uma das implementações mais populares de PROLOG é o SWI-PROLOG
- » Está disponível para Windows e Linux
 - » Windows: <http://swi-prolog.org>
 - » Linux: mesmo site
 - » Debian, Ubuntu etc.
 - ~ `apt-get install swi-prolog-x`



SWI-PROLOG *online*

» <https://swish.swi-prolog.org/>




PROLOG: sintaxe

- » Um programa em PROLOG é uma sequência de declarações finalizadas com ponto final.

```
% Cidades do Brasil  
cidade(manuel).  
cidade(benjamin).  
cidade(teresina).  
  
% Estados do Brasil  
estado(amazonas).  
estado(para).  
  
% Cidades que são capitais  
capital(amazonas, manuel).  
capital(para, teresina).
```

*Se a base diz
que é, então é!*



PROLOG: tipos de dados

- » PROLOG possui vários tipos de dados
- » Alguns são atômicos, outros são compostos
- » Para nosso estudo do paradigma lógico, vamos nos contentar em utilizar
 - » Constantes
 - » Inteiros
 - » [Listas]
 - » Predicados

PROLOG: tipos de dados

- » Os tipos **atômicos** de PROLOG são números, strings ou constantes
 - » **1**
 - » **42**
 - » **"Efigenio"**
 - » **efigenio**
 - » **matematicaDiscreta**
 - » **true**
 - » **false**

Constantes devem sempre começar com uma letra minúscula!

As constantes true e false raramente são utilizadas na base de conhecimento

PROLOG: variáveis

- » Variáveis são identificadores que começam com uma letra maiúscula

```
% Cidades do Brasil  
cidade(Manaus).  
cidade(Benjamin).  
cidade(Teresina).  
  
% Estados do Brasil  
estado(Amazonas).  
estado(Para).  
  
% Cidades que são capitais  
capital(Amazonas, Manaus).  
capital(Para, Teresina).
```

Este programa não declara fatos válidos porque utiliza variáveis onde deveria utilizar constantes!!!

PROLOG: listas

- » Listas em PROLOG são dados compostos
 - » []
 - » [1, 2, 3, 4, 5]
 - » [2, 3, 5, 7, 11]
 - » [azul, 2, joao]
 - » [[nome, idade, trabalho],
[joao, 33, jogador]]

PROLOG: predicados

- » Predicados são compostos por duas partes
 - » **Funtor**: o "nome" do predicado
 - » **Argumentos**: uma lista de sujeitos aos quais o predicado se aplica
 - ~ O número de argumentos de um predicado é sua **aridade** (unário, binário, ternário etc.)
 - ~ É comum nos referirmos a um predicado com a notação funtor/aridade

PROLOG: predicados

- » PROLOG possui alguns predicados pré-definidos
- » Alguns deles comportam-se como operadores!
 - » `=/2`
 - ~ Predicado de unificação (não é atribuição!)
 - » `-/1` e `+/1`
 - ~ Usados para representar inteiros com sinal
 - » `-/2`, `+/2`, `*/2`, `//2`, `div/2`, `//2`
 - ~ Operações aritméticas: subtração, multiplicação, divisão real e divisão inteira (`div` e `//`)

PROLOG: regras

- » Uma **regra** é uma cláusula de Horn escrita na forma **consequente** \leftarrow **antecedente**
 - » A conjunção é implícita pela vírgula

$$P(x, y) \wedge P(y, z) \rightarrow Q(x, z)$$

```
avo(V, N) :- pai(V, P), pai(P, N).
```


PROLOG: consulta

» Para consultar a base de conhecimento, apresentamos ao PROLOG um ou mais objetivos

» O PROLOG irá tentar provar esse objetivo

» Base:

```
% pessoa/1: define quem é pessoa (?)  
pessoa(azul).  
pessoa(42).  
pessoa(cachorro).
```

» Consulta:

```
?- pessoa(fulano).  
false.
```

PROLOG: consulta

- » Se utilizarmos variáveis na consulta, PROLOG tentará encontrar valores para essas variáveis que tornam o fato verdadeiro

```
mae(chichi, gohan).  
mae(videl, pan).
```

```
?- mae(chichi, gohan).  
true.
```

```
?- mae(M, pan).  
M = videl.
```

PROLOG: consulta

- » Se houver mais de uma resposta possível, podemos pressionar ; e PROLOG mostrará a próxima resposta

```
mae(chichi, gohan).  
mae(chichi, goten).  
mae(videl, pan).
```

```
?- mae(chichi, Filho).  
Filho = gohan;  
Filho = goten.
```

Agenda

- » Lógica formal e programação lógica
- » Programação em PROLOG
- » O conceito de unificação
- » Unificação em mais detalhes
- » Backtracking
- » Trabalhando com listas

Variáveis "de verdade"

- » Embora PROLOG tenha variáveis, elas não se comportam como as variáveis do paradigma imperativo
 - » Não existe atribuição
 - » O operador = é um predicado

```
? - X = 5.  
X = 5.
```

O que nós escrevemos: "o predicado $=(X, 5)$ é verdadeiro?"

PROLOG respondeu: "sim, contanto que X seja 5."

Variáveis "de verdade"

- » Também não se comportam como no paradigma funcional
 - » O operador = não define apelidos

```
? - X = 5.
```

```
X = 5.
```

```
? - X.
```

```
% erro
```

Pergunta mal-formulada

Unificação

- » O motor de inferência tenta provar um predicado realizando unificações
- » **Unificar** significa estabelecer que dois termos representam o mesmo objeto

Termo A	Termo B	Unificam?
10	10	Sim
homem(adao)	homem(adao)	Sim
[1, 2, 3, 4]	[1, 2, 3, 4]	Sim

Unificação

- » O motor de inferência tenta provar um predicado realizando unificações
- » **Unificar** significa estabelecer que dois termos representam o mesmo objeto

Termo A	Termo B	Unificam?
pai(goku, gohan)	pai(gohan, goku)	Não
soma(7, 3)	10	Não
7 + 3	10	Não

Unificação

- » O operador $=/2$ é na verdade um predicado de unificação
 - » $X = Y$ é verdade se X e Y unificam
- » As regras de unificação de PROLOG são
 - » Dois átomos unificam se forem iguais
 - » Dois objetos complexos unificam se forem iguais em todos os seus termos
 - » Para unificar variáveis, PROLOG tentará instanciá-las

Unificação

» Testes de unificação:

SWI-PROLOG carregado sem uma base.

```
~$ swipl  
Welcome to SWI-Prolog (threaded, 64 bits, version 8.0.3)  
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free soft  
ware.  
Please run ?- license. for legal details.  
  
For online help and background, visit http://www.swi-prolog.org  
For built-in help, use ?- help(Topic). or ?- apropos(Word).
```

```
?- 1 = 1.  
true.
```

O número 1 unifica com o número 2

```
?- complexo(arg1, arg2) = complexo(arg1, arg2).  
true.
```

```
?- □
```

Esses objetos complexos unificam porque possuem o mesmo funtor e os mesmos argumentos.

Unificação

- » Mesmos testes, mas no SWISH

The screenshot shows the SWISH Prolog environment. On the left, a 'Program' tab is open, displaying a Prolog program with two facts: `1 Your Prolog rules and facts`. The main area on the right shows the results of two queries. The first query is `1 = 1.`, which returns `true`. The second query is `complexo(arg1, arg2) = complexo(arg1, arg2).`, which also returns `true`. At the bottom, there is a query input field with the text `?- complexo(arg1, arg2) = complexo(arg1, arg2).` and buttons for 'Examples', 'History', 'Solutions', 'table results', and 'Run!'.

Por enquanto a base está vazia

Faça uma consulta por vez. Pressione CTR+ENTER; os resultados serão apresentados na parte de cima da tela

Unificação


» Testes de unificação:

```
?- pai(goku, gohan) = pai(gohan, goku).  
false.
```

```
?- [1, 2, 3, 4] = [1, 2, 3, 4].  
true.
```

```
?- [1, 2, 3] = [1, 2].  
false.
```

```
?- □
```



Não unifica: o funtor é o mesmo,
mas os argumentos são diferentes.

Unificação

» Testes de unificação:

```
?- 1 + 1 = 2.  
false.
```

```
?- +(1, 1) = 2.  
false.
```

```
?- 
```

O operador `+/2` é um predicado.

`+(1, 1)` não unifica com um átomo.

Unificação

» Testes de unificação:

?- X = 5.
X = 5.

X é uma variável livre.
PROLOG consegue provar essa consulta **unificando** a variável livre com o átomo 5.
O resultado é a unificação realizada.

?- X = 5, Y = 6.
X = 5,
Y = 6.

Tanto X quanto Y são variáveis livres.
Existem duas proposições, unidas por uma
conjunção (operador vírgula)

Cada proposição é um objetivo.

PROLOG irá tentar provar primeiro X = 5. Isso é
feito **unificando** a variável livre X ao átomo 5.

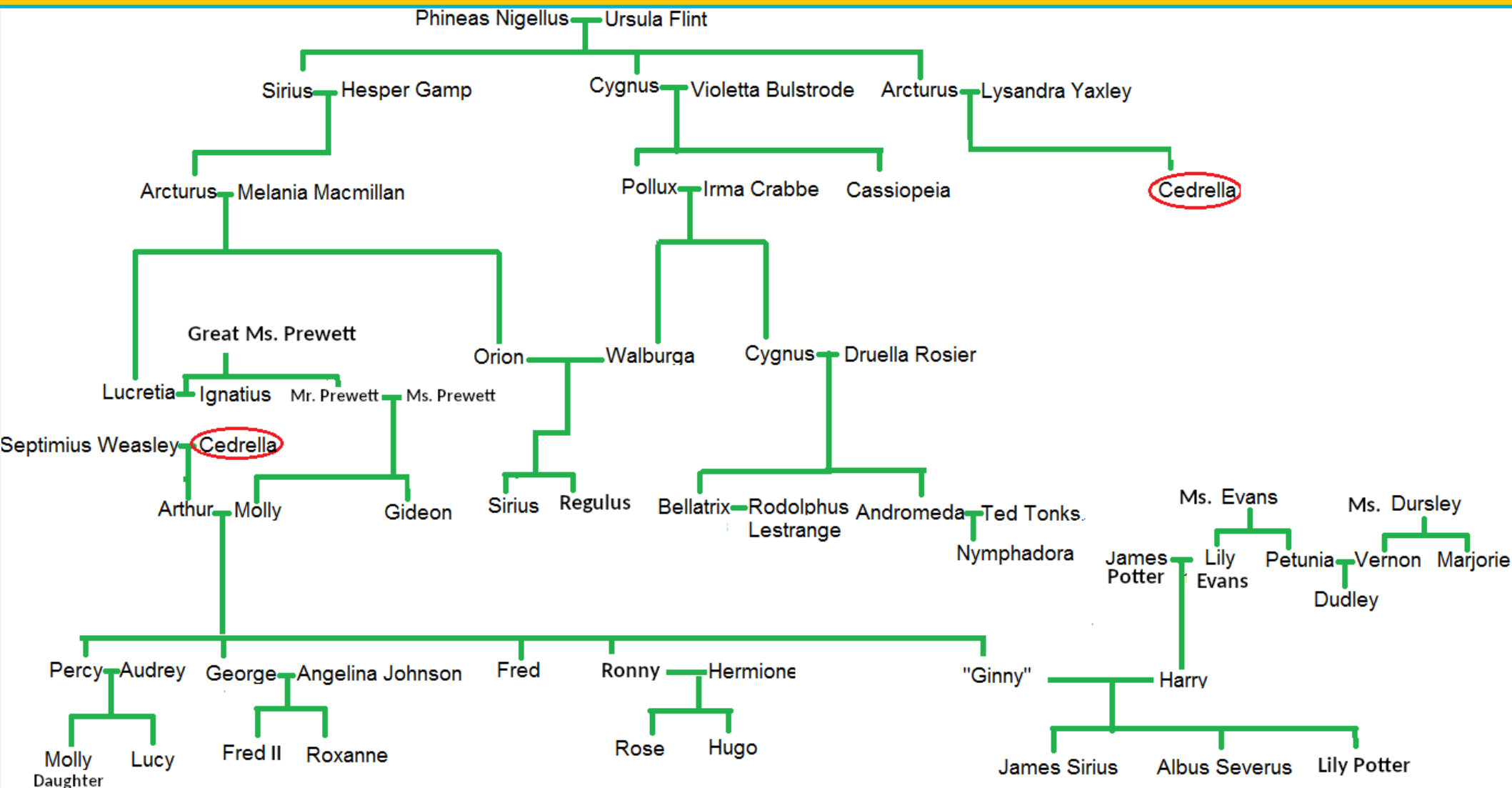
Em seguida, PROLOG tentará provar Y = 6. Isso
é feito **unificando** a variável livre Y ao átomo 6.

?- ☐

Unificação

- » Para os próximos exemplos de unificação, vamos considerar uma árvore genealógica da família Black, do romance de Harry Potter
 - » Problemas de árvore genealógica são muito comuns no estudo do paradigma lógico
 - » As relações familiares são intuitivas e podemos escrever muitas regras com base nelas
 - » Ao mesmo tempo, esses exemplos são ideais para mostrar as limitações do paradigma lógico, pois a abstração é quebrada quando precisamos pensar no processo de busca do motor de inferências

Árvore genealógica de Harry Potter



Descrição da base

- » O predicado `homem/1` e o predicado `mulher/1` estabelecem os gêneros das personagens
- » Usaremos o predicado `mae/2` para estabelecer em `mae(M, F)` que `M` é mãe de `F`
- » Usaremos o predicado `casal/2` para estabelecer em `casal(H, M)` que `H` é casado com `M`
 - » Nesse "mundo fechado" não existe divórcio, nem mães solteiras, nem casais sem filhos, nem adoções etc. etc.
 - » O objetivo é limitar as relações para simplificar o exercício

Descrição da base

- » A base está disponível do ColabWeb
- » Exemplos do que você encontrará nela:

```
1 homem(ronny).
2 homem(harry).
3 mulher(hermione).
4 mulher(ginny).
5
6 mae(molly, percy).
7 mae(molly, ronny).
8 mae(molly, ginny).
9 mae(lily_evans, harry).
10
11 casal(ronny, hermione).
12 casal(harry, ginny).
```

~

~

Consultas à base Harry Potter

SWI-PROLOG carregado com a base de conhecimentos potter.pl

» Testes de unificação

```
~/plp/prog/aula22$ swipl potter.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.0.3)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.
```

```
For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).
```

```
?- casal(ronny, hermione).
true.
```

Pode ser provado porque unifica com o fato `casal(ronny, hermione)` que está na linha 148

```
?- casal(hermione, ronny).
false.
```

```
?- □
```

Não existe nenhum fato na base que unifica com o complexo `casal(hermione, ronny)`

Consultas à base Harry Potter

- » Para fazer os mesmos testes no SWISH
 - » Copie toda a base do arquivo potter.pl e cole no espaço da aba "Program"
 - » Digite as consultas no espaço marcado pela interrogação
 - ~ Canto inferior direito
 - » Os resultados serão mostrados acima da área para consultas

Consultas à base Harry Potter

» Testes de unificação

```
?- casal(ronny, X).  
X = hermione.
```

A consulta `casal(ronny, X)` **unifica** com o fato da linha 148 se PROLOG **instanciar** a variável livre `X` à constante `hermione`.

```
?- casal(X, hermione).  
X = ronny.
```

```
?- □
```

Explicação semelhante, mas agora a variável livre `X` pode ser **instanciada** e **unificada** com a constante `ronny`.

Consultas à base Harry Potter

» Testes de unificação

Agora nossa consulta possui **dois** objetivos

```
?- casal(ronny, X), casal(Y, X).  
X = hermione,  
Y = ronny.
```

?- □

O primeiro, `casal(ronny, X)`, pode ser provado se prolog **instanciar** a variável livre `X`, **unificando-a** com a constante `hermione`

Mas ao provar o segundo objetivo, a variável `X` já **não está mais livre**.

O comportamento agora é equivalente a tentar provar `casal(Y, hermione)`.

Isso é verdade, de acordo com a base de conhecimento, se prolog **instanciar** a variável livre `Y`, unificando-a com a constante `ronny`.

Consultas à base Harry Potter

» Testes de unificação

```
?- casal(X, Y).  
X = phineas,  
Y = ursula ;  
X = great_sirius,  
Y = hesper ;  
X = cygnus,  
Y = violetta ;  
X = arcturus,  
Y = lysandra ;  
X = arcturus_II,  
Y = melania ;  
X = pollux,  
Y = irma .
```

```
?- □
```

Agora temos **duas** variáveis livres

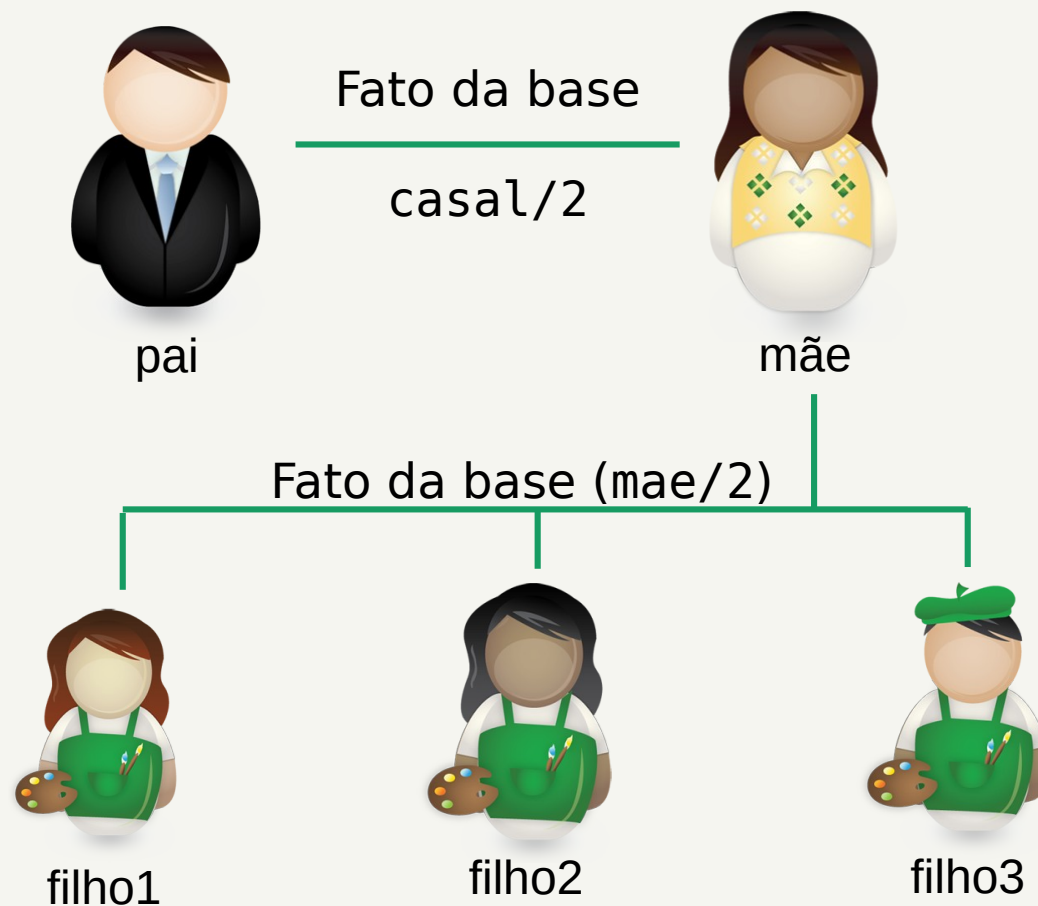
Enquanto continuarmos pressionando ;
PROLOG continuará realizando
instanciações até que tenha esgotado
todas as possibilidades para provar o
objetivo casal(X, Y)

```
?- casal(X, Y).  
X = phineas,  
Y = ursula ;  
X = great_sirius,  
Y = hesper ;  
X = cygnus,  
Y = violetta ;  
X = arcturus,  
Y = lysandra ;  
X = arcturus_II,  
Y = melania ;  
X = pollux,  
Y = irma ;  
X = ignatius,  
Y = lucretia ;  
X = mr_prewett,  
Y = ms_prewett ;  
X = orion,  
Y = walburga ;  
X = cygnus_II,  
Y = drueella ;  
X = septimus,  
Y = cedrella ;  
X = arthur,  
Y = molly ;  
X = rodolphus,  
Y = bellatrix ;  
X = james_potter,  
Y = lily_evans ;  
X = vernon,  
Y = petunia ;  
X = rodolphus,  
Y = bellatrix ;  
X = ted,  
Y = andromeda ;  
X = percy,  
Y = audrey ;  
X = george,  
Y = angelina ;  
X = ronny,
```

A list completa não cabe no slide

Regras familiares

- » Como podemos definir quem é o pai de alguém?



Regras familiares

- » Podemos usar uma relação condicional
 - » Fulano é pai de Beltrano se e somente se existe Sicrana que é mãe de Beltrano e Fulano é casado com Beltrano
 - ~ (De acordo com nossa base!)

```
% pai(P, F)
%   0 predicado pai/2 indica se P
%   é pai de F. Exige que F possua
%   uma mãe e um casamento entre o
%   pai e essa mãe.
pai(P, F) :- mae(M, F), casal(P, M).
```

O predicado pai/2

» Testes de unificação

```
?- pai(arthur, ronny).  
true.
```

```
?- □
```

O fato `pai(arthur, ronny)` não está na base, portanto a unificação não é trivial.

Mas temos uma regra com o predicado `pai/2`

$$\text{pai}(P, F) \text{ :- mae}(M, F), \text{casal}(P, M)$$

Para tentar unificar `pai(arthur, ronny)` e `pai(P, F)`, prolog primeiro instancia as variáveis livres unificando **P = arthur** e depois **F = ronny**

Agora PROLOG precisa provar dois objetivos:
`mae(M, ronny), casal(arthur, M)`

PROLOG tentará várias instanciações diferentes para **M**, até encontrar uma instanciação que permita unificar `mae(M, ronny)` e também `casal(arthur, M)`.

Essa unificação é **M = molly**. Portanto PROLOG consegue provar todos os objetivos e responde **true**.

O predicado pai/2

» Testes de unificação

```
?- pai(arthur, F).
F = percy ;
F = george ;
F = fred ;
F = ronny ;
F = ginny ;
false.
```

Agora a consulta possui uma variável livre.

Mas o procedimento é praticamente identico:

```
pai(P, F) :- mae(M, F), casal(P, M)
```

PROLOG unifica **P = arthur**, mas a variável F ainda continua livre.

```
?- □
```

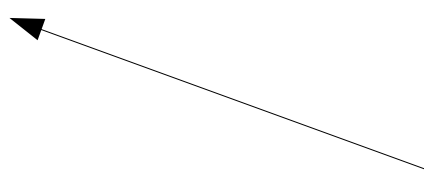
Agora PROLOG precisa provar dois objetivos:
mae(M, F), casal(arthur, M)

Note que tanto M quanto F são variáveis livres, portanto PROLOG encontrará **várias** unificações instanciando **M** e **F**; porém apenas algumas também permitirão unificar casal(arthur, M)... PROLOG imprime todas as instanciação que permitem provar os dois objetivos.

O predicado pai/2

» Testes de unificação

```
?- pai(arthur, F).  
F = percy ;  
F = george ;  
F = fred ;  
F = ronny ;  
F = ginny ;  
false.
```



```
?- ☐
```

Quanto a este **false**... aqui encontramos a primeira limitação do paradigma lógico! A abstração foi quebrada!!

Após todas as instanciações possíveis serem impressas, PROLOG conseguirá fazer a instanciação parcial das variáveis e provar um objetivo... mas não o segundo. Essa prova parcial produz este false.

Para explicar exatamente quais instanciações foram realizadas a fim de produzir esse false... bem, precisaríamos de uma disciplina específica para PROLOG

A regra do progenitor

- » Vamos dizer que P é progenitor de F se P for pai ou mãe de F
- » Um "ou" pode ser expresso com ;
- » Ou implicitamente, com várias proposições usando o mesmo funtor

```
progenitor(P, F) :- pai(P, F); mae(P, F).
```

```
progenitor(P, F) :- pai(P, F).  
progenitor(P, F) :- mae(P, F).
```

Exercícios para fazer **antes** da próxima aula

» *Escreva consultas para provar que*

» James Potter (**james_potter**) é o pai de James Sirius (**james_sirius**) e também de Lily Potter (**lily_potter**)

~ Dica: use uma consulta com dois objetivos

» Rose e Hugo são irmãos

~ Dica: use dois objetivos com **mae/2**

» Nymphadora Tonks é sobrinha de Bellatrix Lestrange :-)

~ Dica: use a relação delas com Druella Rosier

Exercícios para fazer **antes** da próxima aula

» *Escreva consultas para descobrir*

- » Quem são os filhos de Harry Potter (**harry**)
- » Quem são os pais (pai e mãe) de Regulus Black (**regulus**, na base)
- » Quem são os irmãos de Fred Weasley (**fred**)
 - ~ Dica: faça um predicado `irmaos(A, B)` para verificar que A e B possuem mesma mãe
 - ~ Depois faremos os predicados associados a gêneros `irmao/2` e `irma/2`
 - ~ Obs.: não se preocupe se a consulta responder que Fred é irmão dele mesmo

Agenda

- » Lógica formal e programação lógica
- » Programação em PROLOG
- » O conceito de unificação
- » Unificação em mais detalhes
- » Backtracking
- » Trabalhando com listas

Unificações de expressões

- » Em PROLOG, o operador `+` não necessariamente realiza e retorna uma expressão aritmética
- » Sem um contexto especial, a expressão `1 + 1` é equivalente ao termo complexo `+(1, 1)`
- » Esse contexto pode ser fornecidos por operadores como `is` e `==`

Unificações de expressões

- » O operador **is** avalia a expressão do lado direito e unifica com o termo do lado esquerdo

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

```
?- 10 is 5 * 2.  
true.
```

```
?- 2 is 5 / 2.  
false.
```

Unificações de expressões

- » As seguintes expressões não unificam:

```
?- 1 + 1 is 2.  
false.
```

```
?- 1 + 1 is 1 + 1.  
false.
```

O operador `is` avalia a expressão do lado direito.

Em ambos os casos, o objetivo falha porque o termo complexo `+(1, 1)` não unifica com o inteiro `2`.

Unificações de expressões

- » A unificação também não pode ocorrer se os tipos não coincidirem

```
?- 0 is sin(0).  
false.
```

```
?- 0.0 is sin(0).  
true.
```

Unificações de expressões

- » O operador `==` unifica dois termos se eles possuem mesmo valor numérico

```
?- 1 + 1 == 2.  
true.
```

```
?- 1 + 1 == 1 + 1.  
true.
```

```
?- 0 == sin(0).  
true.
```

```
?- cos(0) == sin(pi/2).  
true.
```

Instanciação insuficiente

- » Entretanto, algumas unificações são impossíveis com variáveis livres

```
?- X == 3 + 5.  
ERROR: Arguments are not  
       sufficiently instantiated  
ERROR: In:  
ERROR:    [8] _6424==3+5  
ERROR:    [7] <user>
```

- » O operador `==` tenta avaliar **ambos** os operandos
- » Sem ser capaz de avaliar a variável livre `X`, PROLOG lança uma mensagem de erro

Unificação de listas

- » Em PROLOG, duas listas unificam se todos os seus elementos unificam

Termo A	Termo B	Unificam?
[1, 2, 3, 4]	[1, 2, 3, 4]	Sim
[1, 2, 3]	[1, 2, 2 + 1]	Os termos 3 e 2+1 não unificam
X	[1, 2, 3, 4]	Sim, com $X = [1, 2, 3, 4]$
[1, X]	[1, 2]	Sim, a variável livre X unifica com o termo 2
[1, X, Y]	[1, 2, 3, 4]	As listas possuem tamanhos diferentes e não podem unificar-se

Unificação de listas

- » Além disso, PROLOG possui uma notação especial para representar a cabeça e a cauda de uma lista

Termo A	Termo B	Unificações
$[H \mid T]$	$[1, 2, 3, 4]$	$H = 1$ $T = [2, 3, 4]$
$[A, B \mid \text{Resto}]$	$[1, 2, 3, 4]$	$A = 1$ $B = 2$ $\text{Resto} = [3, 4]$
$[H \mid T]$	$[1]$	$H = 1$ $T = []$
$[H \mid T]$	$[]$	Não unifica!