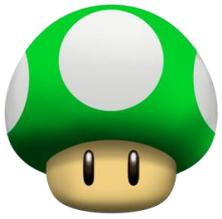


INF 1771 – Inteligência Artificial

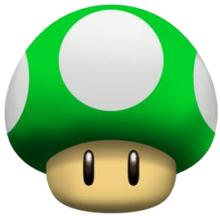
Aula 08 – Introdução ao Prolog

Edirlei Soares de Lima
<elima@inf.puc-rio.br>



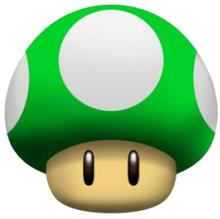
Introdução

- ❏ O Prolog é uma linguagem de programação baseada em **lógica de primeira ordem**.
- ❏ Não é padronizada.
- ❏ Algumas implementações: SICStus Prolog, Borland Turbo Prolog, **SWI-Prolog**...
- ❏ Geralmente é interpretado, mas pode ser compilado.



Prolog x Outras Linguagens

- ❏ **Linguagens Procedimentais (C, Pascal, Basic...):** Especifica-se como realizar determinada tarefa.
- ❏ **Linguagens Orientadas a Objetos (C++, Java, C#...):** Especifica-se objetos e seus métodos.
- ❏ **Prolog:** Especifica-se o quê se sabe sobre um problema e o quê deve ser feito. É mais direcionada ao conhecimento e menos direcionada a algoritmos.



Programação em Prolog

- ❏ **Programar em Prolog envolve:**
 - ❏ Declarar alguns **fatos** a respeito de objetos e seus relacionamentos.
 - ❏ Definir algumas **regras** sobre os objetos e seus relacionamentos.
 - ❏ Fazer **perguntas** sobre os objetos e seus relacionamentos.



SWI-Prolog

- ❏ Open source.
- ❏ Multiplataforma.
- ❏ Possui interface com as linguagens C e C++.
- ❏ www.swi-prolog.org



SWI-Prolog – Interface

```
progenitores.pl - Notepad
File Edit Format View Help
parent(zeus,ares).
male(chaos).
male(cyclope).
male(uranus).
male(chronos).
male(coeus).
male(oceanus).
male(hades).
male(poseidon).
male(zeus).
male(ares).
male(apollo).
male(iapetus).
female(gaea).
female(rhea).
female(leto).
female(hera).
female(phoebe).
female(tethys).
female(artemis).

% -----%
% RULES
% -----%

predecessor(X,Z) :- parent(X,Z).
predecessor(X,Z) :- parent(X,Y),predecessor(Y,Z).

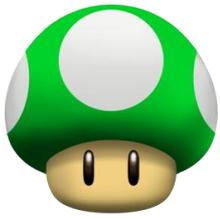
sister(X,Y) :- female(X), parent(Z,X), parent(Z,Y), not(X=Y).

grandparent(X,Z) :- parent(X,Y),parent(Y,Z).
```

```
SWI-Prolog (AMD64, Multi-threaded, version 5.10.2)
File Edit Settings Run Debug Help
% library(win_menu) compiled into win_menu 0.00 sec, 20,952 bytes
% library(swi_hooks) compiled into pce_swi_hooks 0.00 sec, 3,784 bytes
Welcome to SWI-Prolog (Multi-threaded, 64 bits, Version 5.10.2)
Copyright (c) 1990-2010 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

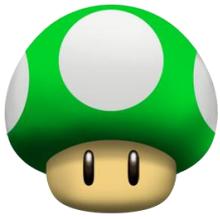
For help, use ?- help(Topic). or ?- apropos(Word).

1 ?-
```



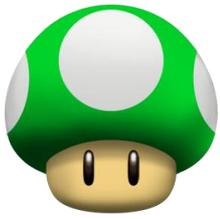
Sentenças Prolog

- ❏ **Nomes de constantes e predicados** iniciam sempre com letra **minúscula**.
- ❏ O **predicado** (relação unária, n-ária ou função) é escrito primeiro e os objetos relacionados são escritos depois entre parênteses.
- ❏ **Variáveis** sempre começam por letra **maiúscula**.
- ❏ Toda sentença termina com ponto “.”
- ❏ **Exemplo:** gosta(maria, jose).



Operadores Lógicos

Simbolo	Conectivo	Opração Lógica
:-	IF	Implicação
,	AND	Conjunção
;	OR	Disjunção
not	NOT	Negação



Operadores Relacionais

Operador	Significado
$X = Y$	Igual a
$X \neq Y$	Não igual a
$X < Y$	Menor que
$Y > X$	Maior que
$Y \leq X$	Menor ou igual a
$Y \geq X$	Maior ou igual a



Regras

- ❏ **Regras** são utilizadas para expressar dependência entre um fato e outro fato:
 - ❏ criança(X) :- gosta(X,sorvete).
 - ❏ criança(X) :- not odeia(X,sorvete).
- ❏ Ou grupo de fatos:
 - ❏ avó(X,Z) :- (mãe(X,Y),mãe(Y,Z)); (mãe(X,Y),pai(Y,Z)).
- ❏ Podem conter listas:
 - ❏ compra(ana, [roupa, comida, brinquedo])



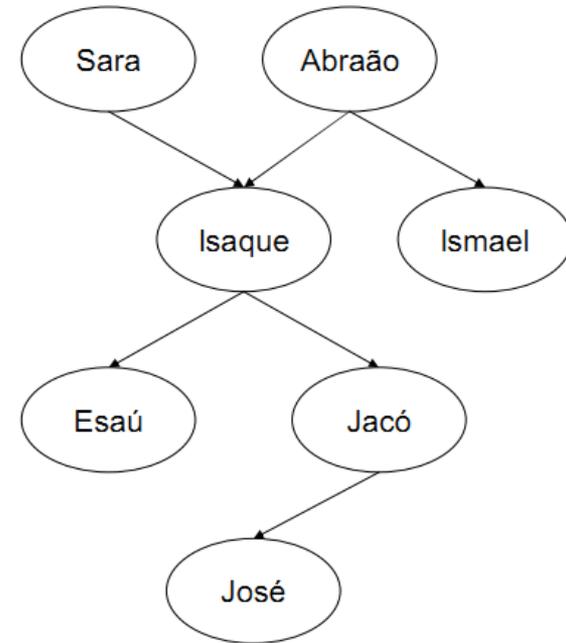
Definindo Relações por Fatos

❏ Exemplo de relações familiares:

- ❏ O fato que **Abraão é um progenitor de Isaque** pode ser escrito em Prolog como:

progenitor(abraão,isaque).

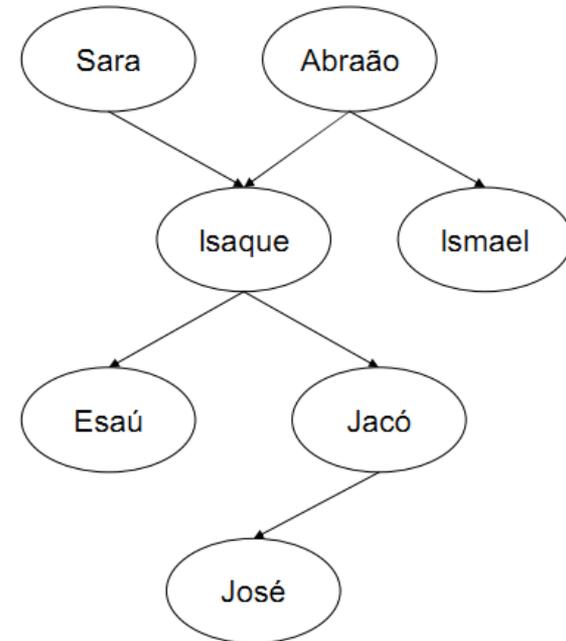
- ❏ Neste caso definiu-se progenitor como o **nome de uma relação**; abraão e isaque são seus argumentos.





Definindo Relações por Fatos

- ❏ Árvore familiar completa em Prolog:
 - ❏ `progenitor(sara,isaque).`
 - ❏ `progenitor(abraão,isaque).`
 - ❏ `progenitor(abraão,ismael).`
 - ❏ `progenitor(isaque,esaú).`
 - ❏ `progenitor(isaque,jacó).`
 - ❏ `progenitor(jacó,josé).`
- ❏ Cada cláusula declara um fato sobre a relação `progenitor`.





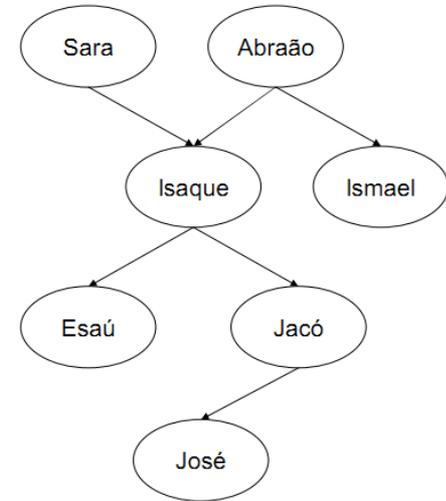
Definindo Relações por Fatos

- Quando o programa é interpretado, pode-se questionar o Prolog sobre a relação progenitor, por exemplo:
Isaque é o pai de Jacó?

?- progenitor(isaque,jacó).

- Como o Prolog encontra essa pergunta como um fato inserido em sua base, ele responde:

true



progenitor(sara,isaque).
progenitor(abraão,isaque).
progenitor(abraão,ismael).
progenitor(isaque,esaú).
progenitor(isaque,jacó).
progenitor(jacó,josé).



Definindo Relações por Fatos

❏ **Uma outra pergunta pode ser:**

?- progenitor(ismael,jacó).

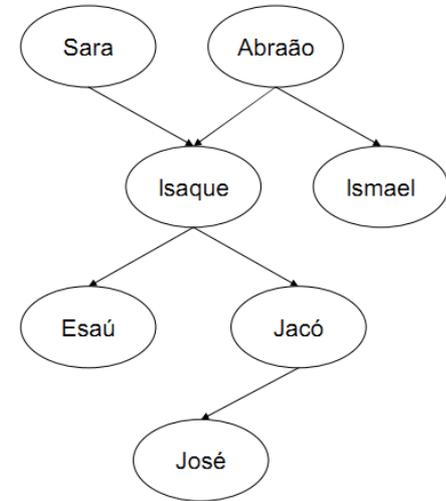
❏ **O Prolog responde:**

false

❏ **O Prolog também pode responder a pergunta:**

?- progenitor(jacó,moisés).

false



progenitor(sara,isaque).
progenitor(abraão,isaque).
progenitor(abraão,ismael).
progenitor(isaque,esaú).
progenitor(isaque,jacó).
progenitor(jacó,josé).



Definindo Relações por Fatos

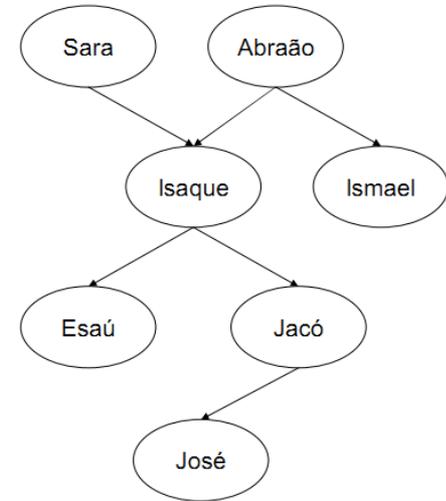
- ❏ Perguntas mais interessantes também podem ser efetuadas:

Quem é o progenitor de Ismael?

?- progenitor(X,ismael).

- ❏ Neste caso, **o Prolog não vai responder apenas true ou false.** O Prolog fornecerá o valor de X tal que a pergunta acima seja verdadeira. Assim a resposta é:

X = abraão



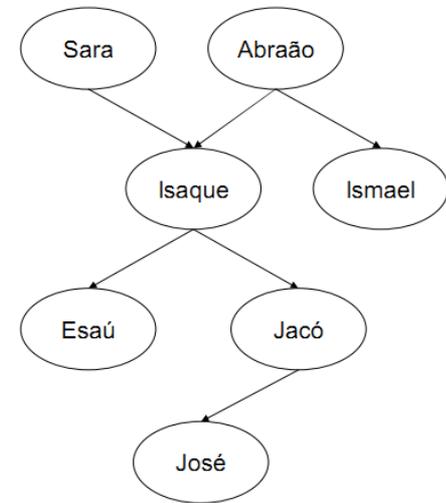
```
progenitor(sara,isaque).  
progenitor(abraão,isaque).  
progenitor(abraão,ismael).  
progenitor(isaque,esaú).  
progenitor(isaque,jacó).  
progenitor(jacó,josé).
```



Definindo Relações por Fatos

- ❏ A pergunta “**Quais os filhos de Isaque?**” pode ser escrita como:

?- progenitor(isaque,X).
- ❏ Neste caso, há **mais de uma resposta possível**. O Prolog primeiro responde com uma solução:
 - ❏ X = esaú
- ❏ Pode-se requisitar uma **outra solução** (digitando ;) e o Prolog a encontra:
 - ❏ X = jacó
- ❏ Se mais soluções forem requisitadas, o Prolog ira responder “false”, pois todas as soluções foram retornadas (false = sem mais soluções).



progenitor(sara,isaque).
progenitor(abraão,isaque).
progenitor(abraão,ismael).
progenitor(isaque,esaú).
progenitor(isaque,jacó).
progenitor(jacó,josé).



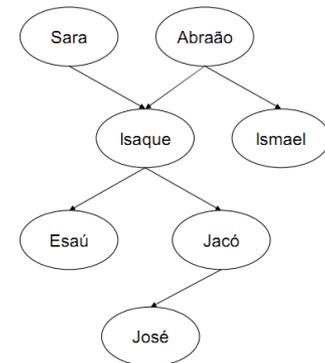
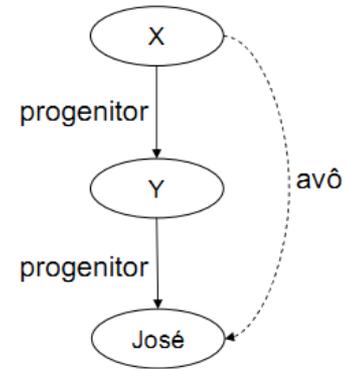
Definindo Relações por Fatos

- ❏ Perguntas mais complexas também podem ser efetuadas, tais como: **Quem é o avô de José?**
- ❏ Como o programa não conhece diretamente a relação avô, esta pergunta deve ser desmembrada em dois passos
 - ❏ (1) Quem é o progenitor de José? Assuma que é um Y
 - ❏ (2) Quem é o progenitor de Y? Assuma que é um X
- ❏ Esta pergunta composta pode ser escrita em Prolog como:

?- progenitor(Y,josé), progenitor(X,Y).

X = isaque

Y = jacó





Definindo Relações por Fatos

- De maneira similar, podemos perguntar:
Quem são os netos de Abraão?

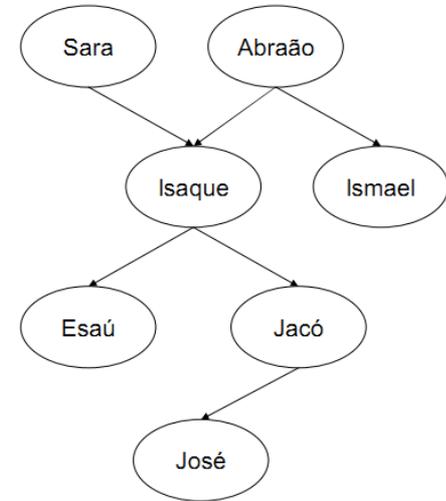
?- progenitor(abraão,X), progenitor(X,Y).

X = isaque

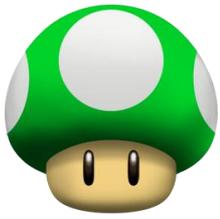
Y = esaú

X = isaque

Y = jacó



progenitor(sara,isaque).
progenitor(abraão,isaque).
progenitor(abraão,ismael).
progenitor(isaque,esaú).
progenitor(isaque,jacó).
progenitor(jacó,josé).



Definindo Relações por Fatos

- É possível **expandir o programa** sobre relações familiares de várias formas. Pode-se, por exemplo, adicionar a informação sobre o **sexo das pessoas** envolvidas.

mulher(sara).

homem(abraão).

homem(isaque).

homem(ismael).

homem(esaú).

homem(jacó).

homem(josé).



Definindo Relações por Regras

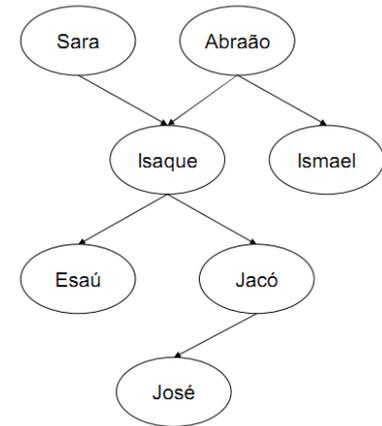
- ❏ Pode-se estender o programa utilizando **regras**. Por exemplo, criando a **relação filho** como o inverso da relação progenitor.
- ❏ É possível definir filho de maneira similar à relação progenitor, ou seja enumerando uma lista de fatos sobre a relação filho, mas **esta não é a forma correta!**

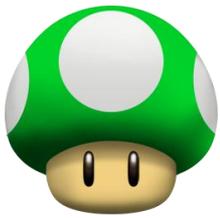
filho(isaque,sara).

filho(isaque,abraão).

filho(ismael,abraão).

...





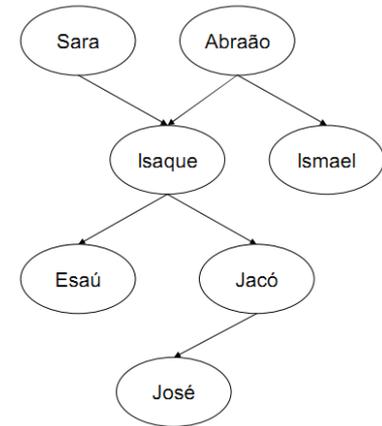
Definindo Relações por Regras

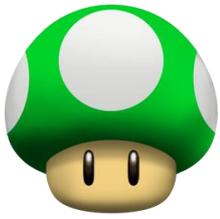
- ❏ A relação filho pode ser definida de modo mais elegante: \square

Para todo X e Y, Y é um filho de X se X é um progenitor de Y.

- ❏ Em Prolog:

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





Definindo Relações por Regras

- ❏ A cláusula Prolog `filho(Y,X) :- progenitor(X,Y)` é chamada de **regra** (rule).
- ❏ Há uma diferença importante entre fatos e regras:
 - ❏ Um fato é sempre verdadeiro (verdade incondicional).
 - ❏ Regras especificam coisas que são verdadeiras se alguma condição é satisfeita.



Definindo Relações por Regras

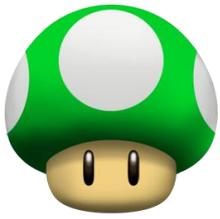
- Após definir a regra filho, é possível perguntar ao Prolog se **Ismael é filho de Abraão**:

?- filho(ismael, abraão).

- Como não existem fatos sobre a relação filho, a única forma do Prolog responder esta pergunta é aplicando a **regra filho**:

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

- A regra filho é aplicável a qualquer objeto X e Y; portanto ela pode também ser aplicada a objetos ismael e abraão.



Definindo Relações por Regras

- Para aplicar a regra a ismael e abraão, Y tem que ser substituído por ismael e X por abraão. Ou seja, as variáveis X e Y estão instanciadas a:

X = abraão e Y = ismael

- Depois da instanciação, obtêm-se um caso especial da regra:

filho(ismael,abraão) :- progenitor(abraão,ismael).

- Se o Prolog **provar** que progenitor(abraão,ismael) é **verdadeiro**, então ele pode afirmar que filho(ismael,abraão) também é **verdade**.



Definindo Relações por Regras

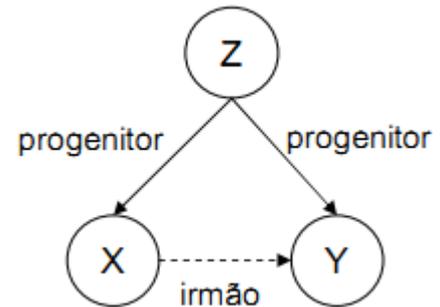
- É possível também incluir a especificação da **relação mãe**, com base no seguinte fundamento lógico:
- Para todo X e Y,
 - X é a mãe de Y se
 - X é um progenitor de Y e
 - X é uma mulher.
- Traduzindo para Prolog:

$\text{m\~{a}e}(X,Y) \text{ :- progenitor}(X,Y), \text{mulher}(X).$



Definindo Relações por Regras

- ❏ A relação **irmão** pode ser definida como:
- ❏ Para todo X e Y,
 - ❏ X é irmão de Y se
 - ❏ ambos X e Y têm um progenitor em comum.



- ❏ Em Prolog:

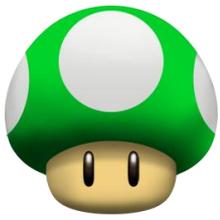
`irmão(X,Y) :- progenitor(Z,X), progenitor(Z,Y).`



Interpretação Prolog

- ❏ A **interpretação** do programa pode Prolog ser lógica ou procedimental.
- ❏ A interpretação procedimental corresponde a satisfazer cada **subgoal** mediante processos sucessivos de **matching**.
- ❏ Exemplo:
pai(fred, marcos).
pai(ricardo, pedro).
pai(pedro, paulo).
avo(X,Y) :- pai(X, Z), pai(Z, Y).

?- avo(X,paulo).



Programas Prolog

- ❏ Programas Prolog podem ser estendidos simplesmente pela adição de novas cláusulas.
- ❏ Cláusulas Prolog são de três tipos: fatos, regras e perguntas.
 - ❏ **Fatos** declaram coisas que são sempre (incondicionalmente) verdadeiras.
 - ❏ **Regras** declaram coisas que são verdadeiras dependendo de determinadas condições.
 - ❏ Através de **perguntas**, o usuário pode questionar o programa sobre quais coisas são verdadeiras.



Regras Recursivas

- Para criar uma relação **ancestral** é necessária a criação de uma regra recursiva:

`ancestral(X,Z) :- progenitor(X,Z).`

`ancestral(X,Z) :- progenitor(X,Y), ancestral(Y,Z).`

- Quais os descendentes de Sara?

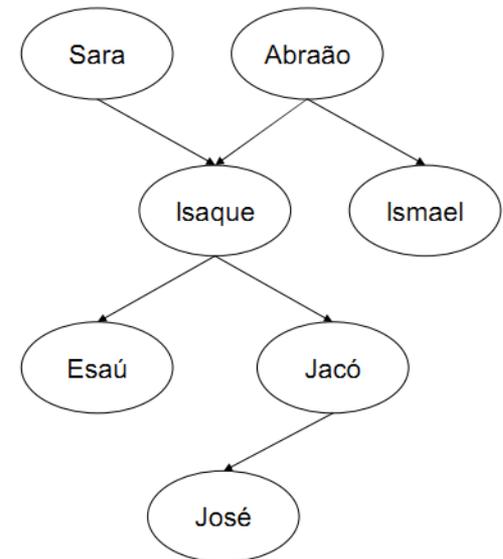
`ancestral(sara,X).`

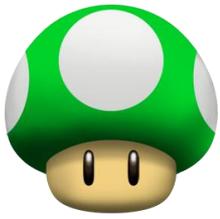
`X = isaque;`

`X = esaú;`

`X = jacó;`

`X = josé`





Programa Exemplo

progenitor(sara,isaque).
progenitor(abraão,isaque).
progenitor(abraão,ismael).
progenitor(isaque,esaú).
progenitor(isaque,jacó).
progenitor(jacó,josé).

mulher(sara).
homem(abraão).
homem(isaque).
homem(ismael).
homem(esaú).
homem(jacó).
homem(josé).

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

mae(X,Y) :- progenitor(X,Y),
mulher(X).

avo(X,Z) :- progenitor(X,Y),
progenitor(Y,Z).

irmao(X,Y) :- progenitor(Z,X),
progenitor(Z,Y).

ancestral(X,Z) :- progenitor(X,Z).

ancestral(X,Z) :- progenitor(X,Y),
ancestral(Y,Z).