



**UNIVERSIDADE DO ESTADO DO AMAZONAS
ESCOLA SUPERIOR DE TECNOLOGIA**

**MARCOS COSTA ANTUNES AFONSO
KAIQUE OLEGAR AMARO DOS SANTOS
VALÉRIA RIBEIRO DOS SANTOS**

**ATIVIDADE DE INTELIGÊNCIA ARTIFICIAL
PROLOG 1**

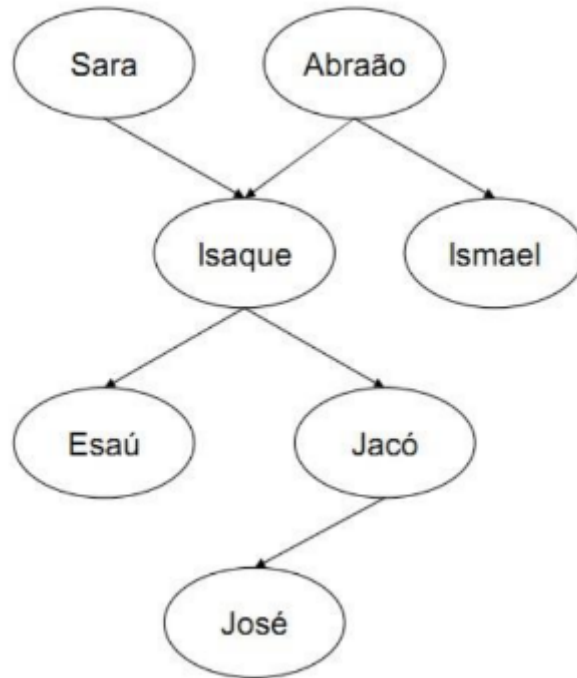
MANAUS

2024
SUMÁRIO

SUMÁRIO	2
EXEMPLO: ÁRVORE GENEALÓGICA	3
EXEMPLO: MACACO E AS BANANAS	7
EXEMPLO: OPERAÇÕES EM LISTA (BUSCA)	9

EXEMPLO: ÁRVORE GENEALÓGICA

- Árvore genealógica usada como exemplo para o uso do PROLOG:

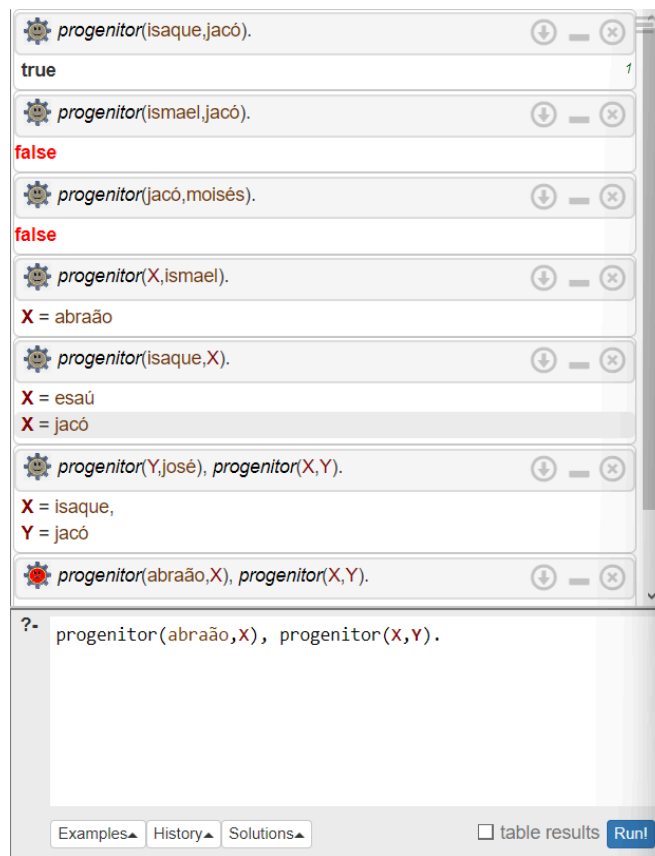


- Inserindo as pessoas apresentadas na árvore genealógica no SWISH:

The screenshot shows the SWISH Prolog editor interface. The menu bar includes File, Edit, Examples, and Help. A single tab titled 'Program' is open. The editor contains the following Prolog facts:

```
1 progenitor(sara,isaque).
2 progenitor(abraão,isaque).
3 progenitor(abraão,ismael).
4 progenitor(isaque,esaú).
5 progenitor(isaque,jacó).
6 progenitor(jacó,josé).
```

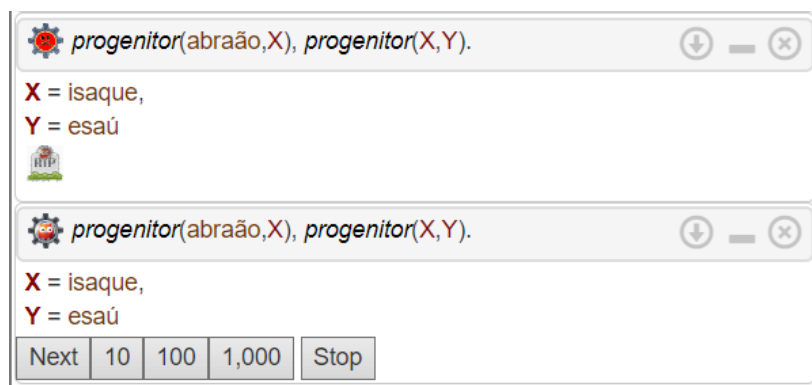
- Resultado obtido no SWISH após a inserção das pessoas apresentadas na árvore genealógica:



The screenshot shows the SWISH Prolog environment with a list of queries and their results. The queries are as follows:

- `progenitor(isaque,jacó).` Result: `true`
- `progenitor(ismael,jacó).` Result: `false`
- `progenitor(jacó,moisés).` Result: `false`
- `progenitor(X,ismael).` Result: `X = abraão`
- `progenitor(isaque,X).` Result: `X = esaú`, `X = jacó`
- `progenitor(Y,josé), progenitor(X,Y).` Result: `X = isaque,`, `Y = jacó`
- `progenitor(abraão,X), progenitor(X,Y).` Result: `?- progenitor(abraão,X), progenitor(X,Y).`

At the bottom, there are buttons for `Examples`, `History`, `Solutions`, a checkbox for `table results`, and a `Run!` button.



The screenshot shows the SWISH Prolog environment with a specific query and its results. The query is:

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

The results are:

- `X = isaque,`
- `Y = esaú`

Below the results, there is a small icon of a person. At the bottom, there are buttons for `Next`, `10`, `100`, `1,000`, and `Stop`.

```

progenitor(abraão,X), progenitor(X,Y).
X = isaque,
Y = esaú
X = isaque,
Y = jacó
false

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

Examples History Solutions table results Run!

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

filho(ismael, abraão).
true

?- filho(ismael, abraão).|

```

- Inserção de novas condições no SWISH:

```

8 mulher(sara).
9 homem(abraão).
10 homem(isaque).
11 homem(ismael).
12 homem(esaú).
13 homem(jacó).
14 homem(josé).
15 mãe(X,Y) :- progenitor(X,Y), mulher(X).
16 pai(X,Y) :- progenitor(X,Y), homem(X).
17 irmão(X,Y) :- progenitor(Z,X), progenitor(Z,Y).
18 avo(X,Y) :- pai(X, Z), pai(Z, Y).|

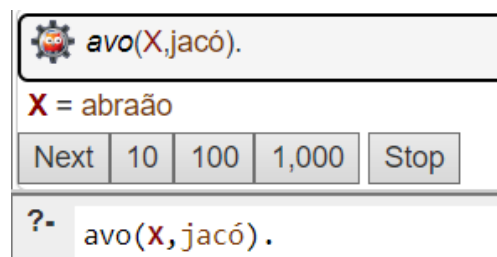
```

- Observando os resultados do SWISH após a inserção das condições acima:



The screenshot shows the SWISH interface with four queries and their results:

- Query: `mãe(sara, isaque).` Result: `true`
- Query: `pai(abraão, isaque).` Result: `true`
- Query: `avo(abraão, isaque).` Result: `false`
- Query: `avo(abraão, jacó).` Result: `true`



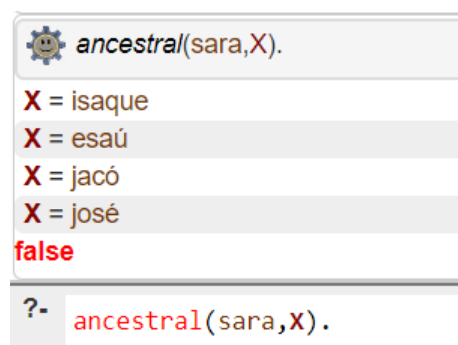
The screenshot shows the SWISH interface with a query and its result:

- Query: `avo(X,jacó).`
- Result: `X = abraão`
- Buttons: Next, 10, 100, 1,000, Stop
- Query: `?- avo(X,jacó).`

- Inserção de novas condições no SWISH:

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

- Validando os resultados do SWISH após a inserção das novas condições:



The screenshot shows the SWISH interface with a query and its result:

- Query: `ancestral(sara,X).`
- Result: `X = isaque`
- Result: `X = esaú`
- Result: `X = jacó`
- Result: `X = josé`
- Result: `false`
- Query: `?- ancestral(sara,X).`

EXEMPLO: MACACO E AS BANANAS

- Inserindo no SWISH (com PROLOG) as condições desejadas para o problema:

```
1 move(  
2 estado(no_centro, acima_caixa, no_centro, não_tem),  
3 pegar_banana,  
4 estado(no_centro, acima_caixa, no_centro,tem)  
5 ).  
6 move(  
7 estado(P, no_chão, P, Banana),  
8 subir,  
9 estado(P, acima_caixa, P, Banana)  
10 ).  
11 move(  
12 estado(P1, no_chão, P1, Banana),  
13 empurrar(P1, P2),  
14 estado(P2, no_chão, P2, Banana)  
15 ).  
16 move(  
17 estado(P1, no_chão, Caixa, Banana),  
18 caminhar(P1, P2),  
19 estado(P2, no_chão, Caixa, Banana)  
20 ).  
21 consegue(Estado1) :- move(Estado1, Movimento, Estado2), consegue(Estado2).
```

- Observando os resultados obtidos após fazer algumas chamadas nas condições anteriormente estabelecidas:

The screenshot shows a Prolog interpreter window with the following content:

```
consegue(estado(na_porta, no_chão, na_janela, não_tem)).
Singleton variables: [Movimento]
true
```

Below the first call are buttons: Next, 10, 100, 1,000, and Stop.

```
consegue(estado(na_porta, no_chão, na_janela, na_porta)).
Singleton variables: [Movimento]
Stack limit (0.2Gb) exceeded
Stack sizes: local: 0.2Gb, global: 48.8Mb, trail: 10.8Mb
Stack depth: 710,068, last-call: 0%, Choice points: 710,050
Possible non-terminating recursion:
[710,067] consegue(<compound estado/4>)
[710,066] consegue(<compound estado/4>)
```

```
consegue(estado(na_porta, no_chão, na_janela, tem)).
Singleton variables: [Movimento]
true
```

Below the third call are buttons: Next, 10, 100, 1,000, and Stop.

At the bottom, a fourth call is shown with a question mark icon, indicating it failed:

```
?- consegue(estado(na_porta, no_chão, na_janela, tem)).
```

- Visualizando os resultados obtidos juntamente com as condições estabelecidas:

The screenshot shows a Prolog interpreter window with the following content:

```
1 move(
2 estado(no_centro, acima_caixa, no_centro, não_tem),
3 pegar_banana,
4 estado(no_centro, acima_caixa, no_centro, tem)
5 ).
6 move(
7 estado(P, no_chão, P, Banana),
8 subir,
9 estado(P, acima_caixa, P, Banana)
10 ).
11 move(
12 estado(P1, no_chão, P1, Banana),
13 empurrar(P1, P2),
14 estado(P2, no_chão, P2, Banana)
15 ).
16 move(
17 estado(P1, no_chão, Caixa, Banana),
18 caminhar(P1, P2),
19 estado(P2, no_chão, Caixa, Banana)
20 ).
21 consegue(estado(., ., tem)).
22 consegue(Estado1 :- move(Estado1, Movimento, Estado2), consegue(Estado2).
```

Below the code is a button: Singleton variables: [Movimento].

On the right, a large owl icon is visible. Below it, a Prolog interpreter window shows the following content:

```
consegue(estado(na_porta, no_chão, na_janela, não_tem)).
Singleton variables: [Movimento]
true
```

Below the first call are buttons: Next, 10, 100, 1,000, and Stop.

At the bottom, a fourth call is shown with a question mark icon, indicating it failed:

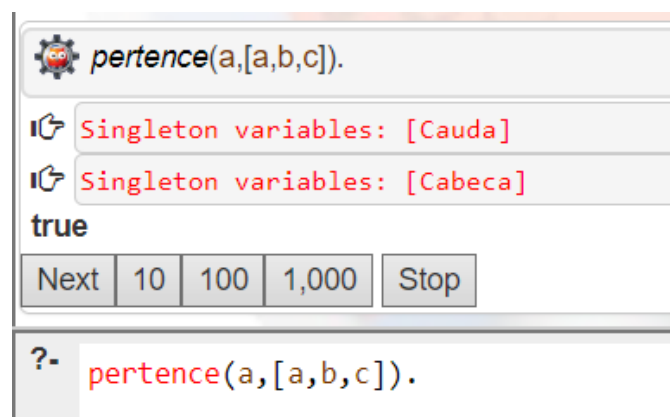
```
?- consegue(estado(na_porta, no_chão, na_janela, não_tem)).
```


EXEMPLO: OPERAÇÕES EM LISTA (BUSCA)

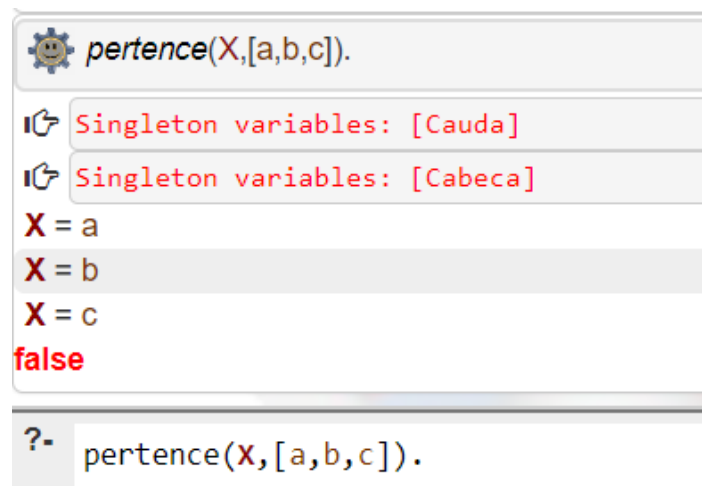
- Inserindo o exemplo das operações em lista:

```
1 pertence(Elemento,[Elemento|Cauda]).  
Singleton variables: [Cauda]  
2 pertence(Elemento,[Cabeca|Cauda]) :- pertence(Elemento,Cauda).  
Singleton variables: [Cabeca]
```

- Observando resultados obtidos a partir do exemplo das operações em lista:



```
⚙️ pertence(a,[a,b,c]).  
📋 Singleton variables: [Cauda]  
📋 Singleton variables: [Cabeca]  
true  
Next | 10 | 100 | 1,000 | Stop  
?- pertence(a,[a,b,c]).
```



```
⚙️ pertence(X,[a,b,c]).  
📋 Singleton variables: [Cauda]  
📋 Singleton variables: [Cabeca]  
X = a  
X = b  
X = c  
false  
Next | 10 | 100 | 1,000 | Stop  
?- pertence(X,[a,b,c]).
```