Linguagem Lógica

Instituto Federal de Educação, Ciência e Tecnologia do Triângulo Mineiro
Prof. Edwar Saliba Júnior
Setembro de 2021

Introdução

- A lógica possui uma longa história, de mais de 23 séculos, que remonta aos antigos filósofos gregos, principalmente Aristóteles, que estabeleceu os fundamentos da lógica de maneira sistemática;
- a lógica trata fundamentalmente de dois conceitos: *verdade* e *prova*;
- a lógica entrou para o campo da matemática com George Boole (1815-1864) que propôs, em 1847, uma linguagem formal que permite a realização de inferências.

Introdução

- A chamada *lógica moderna* teve início em 1879, data em que Friedrich Ludwig Gottlob Frege (1848-1925), matemático e filósofo alemão, publicou a primeira versão do que hoje é conhecido como *cálculo de predicados*;

- no final do século XIX a lógica passou a ser utilizada como base formal para outros campos da matemática; e seus métodos e limites passaram a ser estudados com rigor.

Sistemas Lógicos

- Um sistema lógico consiste em um conjunto de fórmulas e um conjunto de regras de inferência;
- as fórmulas são sentenças pertencentes a uma linguagem formal cuja sintaxe é dada;
- cada fórmula pode ser associada a um valor *verdade* (V) ou *falso* (F) e
- a parte lógica que estuda os valores verdade é chamada *teoria de modelos.*
- Dois problemas importantes estão relacionados com o valor verdade de uma fórmula:
  - o problema da validade, isto é, se é possível que uma dada fórmula apresente o valor verdade `verdadeiro` e
  - o problema da tautologia, ou seja, se uma fórmula é ou não sempre verdadeira.

Sistemas Lógicos

- A partir da introdução, por Robinson e Smullyan em 1960, de procedimentos eficientes para demonstração automática de teoremas por computador, a lógica passou a ser estudada também como método computacional para a solução de problemas.
- Para que funcione, expressões lógicas têm que ser colocadas na forma canônica. Ou seja, restringe-se o número de operadores: `AND`, `OR` e `NOT` e especifica-se a sintaxe a ser respeitada.
- É possível associar uma semântica operacional a um procedimento de prova automática de teoremas. Isto permitiu a definição de linguagens de programação baseada em lógica, a linguagem PROLOG que aqui será estudada, por exemplo.
- Esta linguagem já foi restrita a laboratórios de pesquisa. Mas, hoje é amplamente utilizada e seus aperfeiçoamentos e possíveis extensões são objetos de intensas pesquisas.

Linguagem Lógica - Sintaxe

- Formalmente uma linguagem lógica de primeira ordem - notada L(P, F, C, V) - é determinada pela especificação dos seguintes conjuntos:
  - um conjunto P de símbolos de Predicado;
  - um conjunto F de símbolos de Função;
  - um conjunto C de símbolos de Constante e
  - um conjunto V de símbolos de Variável;
- a cada símbolo de Predicado e de Função é associada uma aridade (número de argumentos ou de operandos que uma função ou operação requer);
- os símbolos de Predicado com aridade zero são chamados de *símbolos proposicionais*. Constantes podem ser consideradas como funções de aridade zero;
- estes conjuntos formam o alfabeto da linguagem lógica.

Linguagem Lógica - Sintaxe

- Os símbolos de operadores lógicos têm os seguintes nomes:

$$\neg = \textit{negação}$$
$$\wedge = \textit{e}$$
$$\vee = \textit{ou}$$
$$\rightarrow = \textit{implica}$$
$$\forall = \textit{qualquer}$$
$$\exists = \textit{existe}$$

Linguagem Lógica - Sintaxe

- A sintaxe da linguagem pode ser definida através da seguinte linguagem formal:

$$\langle termo \rangle ::= \langle variável \rangle | \langle constante \rangle | \langle função \rangle (\langle termo \rangle_1, \ldots, \langle termo \rangle_n)$$

$$\langle fórmula-atômica \rangle ::= V | F | \langle predicado \rangle (\langle termo \rangle_1, \ldots, \langle termo \rangle_n)$$

$$\langle fórmula \rangle ::= \langle fórmula-atômica \rangle | \neg (\langle fórmula \rangle) | (\langle fórmula \rangle_1 \wedge \langle fórmula \rangle_2) | (\langle fórmula \rangle_1 \vee \langle fórmula \rangle_2) | (\langle fórmula \rangle_1 \rightarrow \langle fórmula \rangle_2) | (\forall \langle variável \rangle \cdot (\langle fórmula \rangle)) | (\exists \langle variável \rangle \cdot (\langle fórmula \rangle))$$

Para definir a sintaxe da lógica utiliza-se a notação BNF (Backus-Naur Formalism) que consiste em uma série de meta-símbolos (isto é, símbolos que não pertencem à linguagem escrita) utilizados para definir meta-identificadores (notados por símbolos entre os caracteres "<" e ">"), que correspondem a classes de elementos da linguagem a ser definida. Todo meta-identificador é definido a partir de outros meta-identificadores ou a partir de símbolos terminais, isto é, os símbolos da linguagem.

Significado dos meta-símbolos:
::= "definido como"
| "separador de definições alternativas"
[<item>] "zero ou uma ocorrência de <item>"
{<item>} "zero ou mais ocorrências de <item>"

Linguagem Lógica - Sintaxe

`<constantes>::= a, b, c, d` e outras palavras iniciadas por minúsculas

`<variável>::= x, y, z, w` com ou sem índices

`<função>::= f, g, h` e outras palavras iniciadas por minúsculas

`<predicado>::= P, Q, R` e outras palavras iniciadas por maiúsculas

- onde os símbolos `V` e `F` representam `verdadeiro` e `falso`.

Exemplos

- Esta linguagem permite a geração de fórmulas como as seguintes:

$$((Pai(a,b) \wedge Pai(b,c)) \rightarrow Avo(a,c)),$$

$$\forall x.((P(x) \wedge \neg(P(a))) \rightarrow Q(b)),$$

$$Ama(amélia, z),$$

$$\neg(Ama(brutus, cesar)),$$

$$\forall x. \exists y.(Gosta(y,x)),$$

$$F.$$

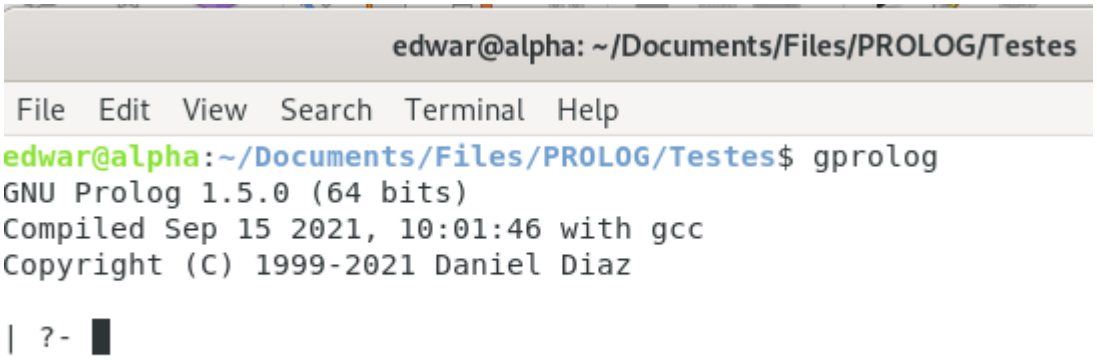
PROLOG

- Do inglês *PROgramming in LOGic*;
- é fruto de uma das mais antigas linhas de pesquisa em Inteligência Artificial (IA) e
- é um provador automático de teoremas, onde a estratégia de escolha de cláusulas adotada é a estratégia SLD (*Selective Linear resolution for Definite clauses*).

PROLOG – Comandos Básicos

- Iniciar o ambiente:

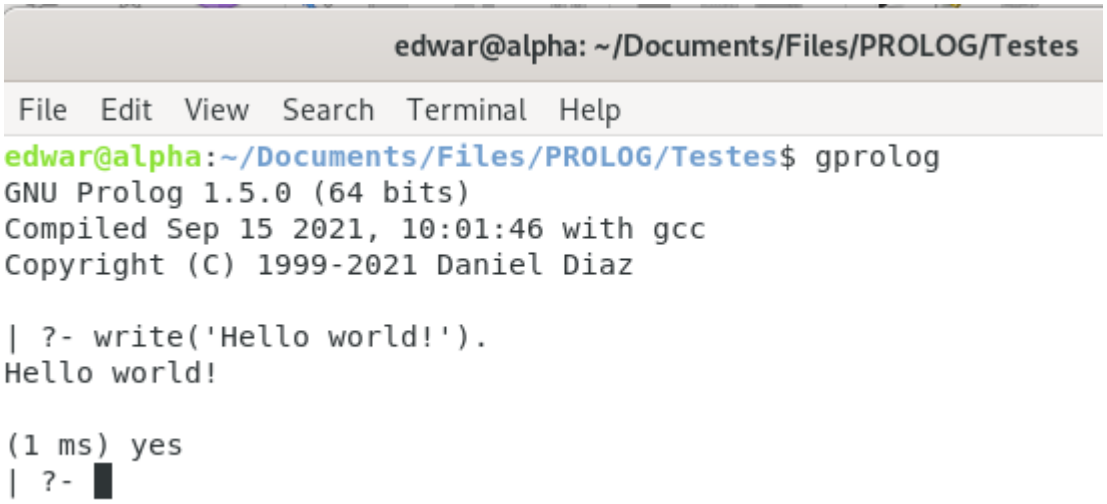
```
$ gprolog
```



PROLOG – Comandos Básicos

- Hello world!:

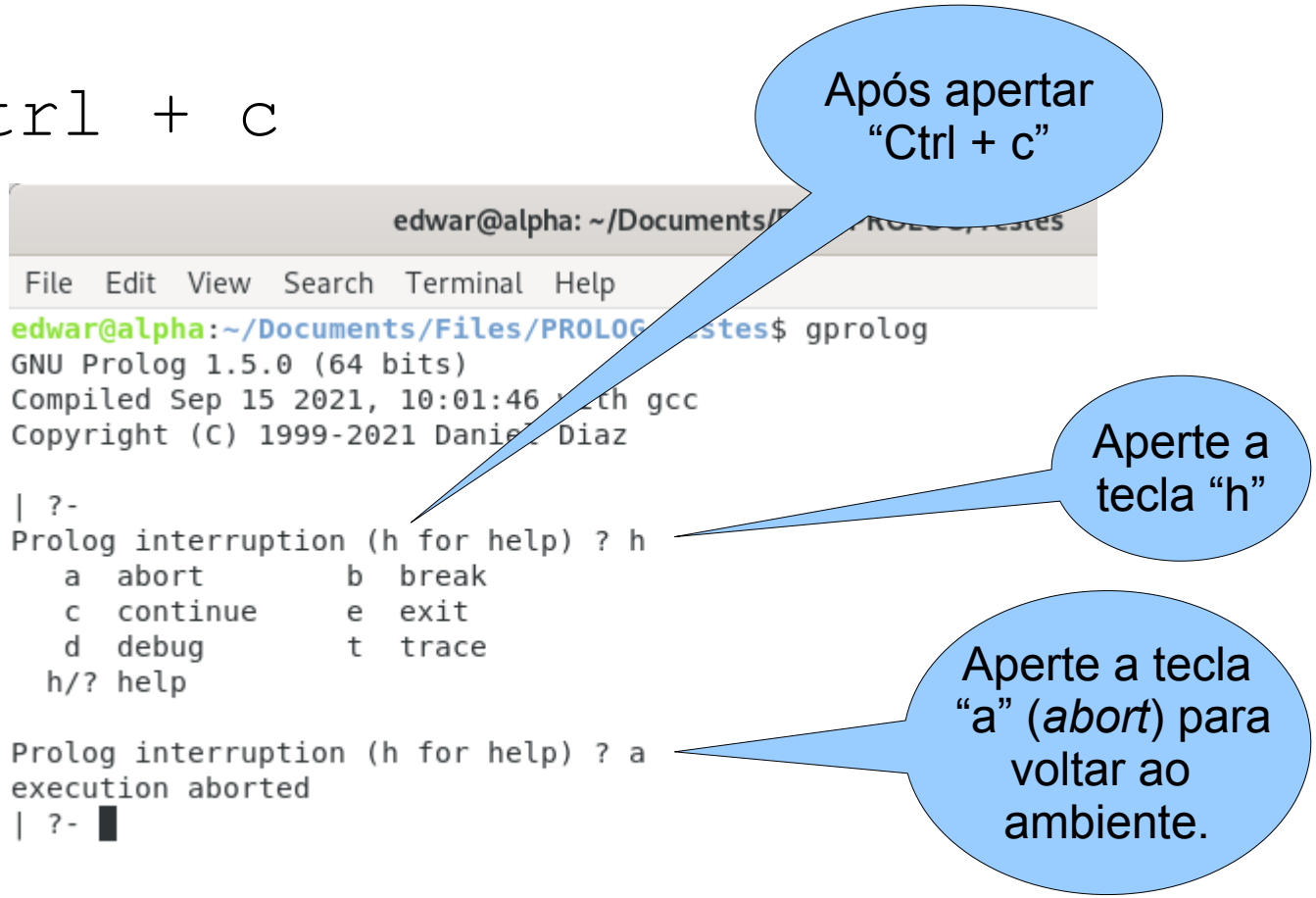
```
$ gprolog
?- write('Hello world!').
```



PROLOG – Comandos Básicos

- Dentro do ambiente, como chamar o *help*:

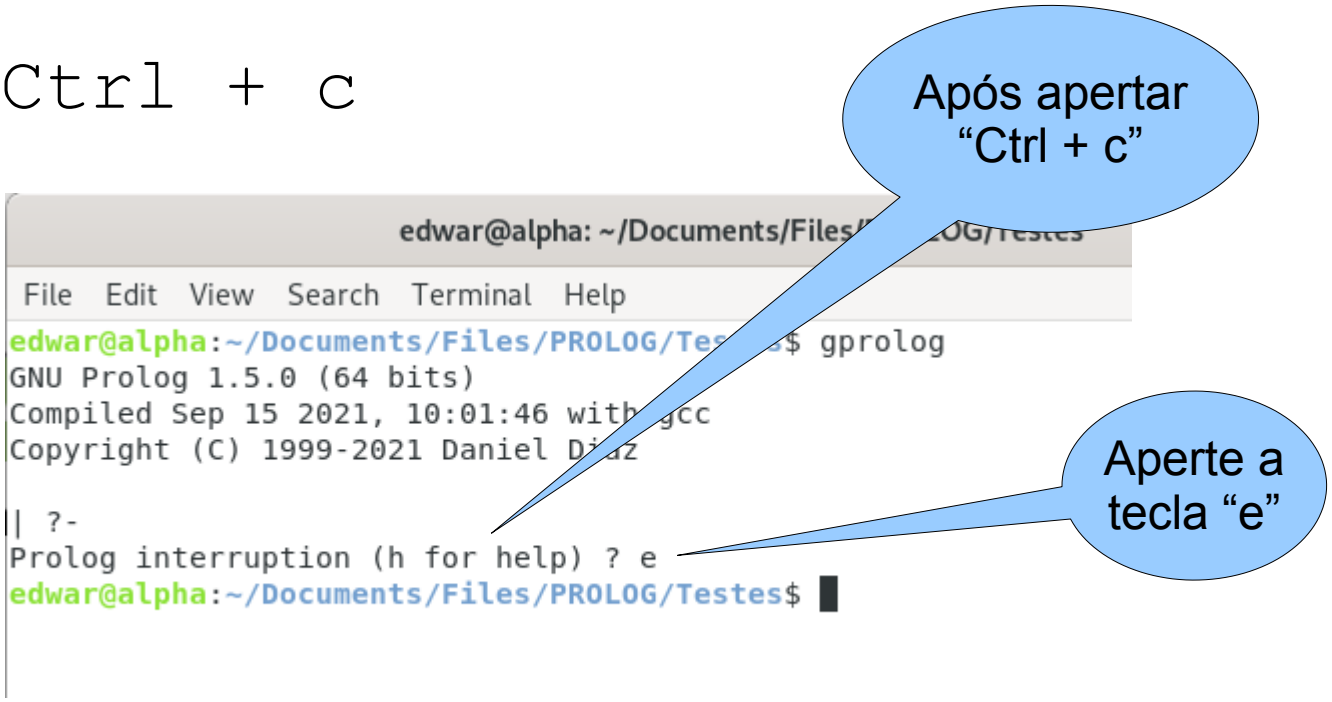
```
?- Ctrl + c
```



PROLOG – Comandos Básicos

- Dentro do ambiente, como voltar ao cursor do terminal:

```
?- Ctrl + c
```



Compilando uma Base de Dados em `gprolog`

- Primeiramente deve-se montar uma base de dados (ou de fatos e/ou regras) preliminar como apresentado a seguir, “Família de Abraão”;

```
progenitor(sara,isaque).
progenitor(abraao,isaque).
progenitor(abraao,ismael).
progenitor(isaque,esau).
progenitor(isaque,jaco).
progenitor(jaco,jose).
```

- salve esta base em um arquivo com a extensão `.pl`
- no terminal compile o arquivo com a seguinte linha de comando: `gplc nomeDoArquivo.pl`
- execute o arquivo compilado: `./nomeDoArquivo`
- seu ambiente PROLOG funcionará normalmente para a base de dados preestabelecida.

PROLOG

- Características linguagem:
  - O conteúdo a seguir foi extraído do material: DANTAS, L. A. **Descobrindo o Prolog**. Disponível em: <http://www.linhadecodigo.com.br/artigo/1697/descobrindo-o-prolog.aspx>. Ascesso em: 01 Nov. 2021.
- **principais:**
  - é uma linguagem de programação lógica e declarativa, ou seja, ao invés do programa estipular a maneira de chegar à solução passo a passo, ele fornece uma descrição do problema que se pretende processar utilizando uma coleção de fatos e regras (lógica) que indicam como deve ser resolvido o problema proposto.

PROLOG

- Prolog é mais direcionado ao conhecimento do que aos próprios algoritmos;
- além de ser uma linguagem declarativa, outro fato que a difere das outras linguagens é a questão de não possuir estruturas de controle (`if-else`, `do-while`, `for`, `switch`) presentes na maioria das linguagens de programação e
- um programa em Prolog pode rodar em um modo interativo, o usuário poderá formular questões utilizando os fatos e as regras para produzir a solução através do ambiente de interação.

PROLOG

- **conceitos básicos:**
  - os tipos de dados comumente existentes em outras linguagens não são em empregados em Prolog e
  - todos os dados são tratados como sendo de um único tipo, este conhecido como **termo** e que pode ser:
    - uma constante,
    - uma variável ou
    - um termo composto.

PROLOG

- **fatos:**
  - um **fato** é formado por um predicado, seus argumentos (objetos) e finalizados com um ponto (.). Exemplo:

```
predicado(argumento1,argumento2).
```

  - o predicado é a **relação** sobre a qual os objetos vão interagir. Exemplo:

```
amiga(joana, maria).
```

  - Definimos uma relação de amizade entre dois objetos, **joana** e **maria**;
  - reforçando! A **aridade** é o número de argumentos de uma relação. E é denotada como uma barra seguida pela aridade:
    - `amiga/2` significa que a relação `amiga` possui 2 argumentos, ou que a relação `amiga` tem aridade 2.

PROLOG

- Outro exemplo:

```
homem(jose).
```

- Note que quando usamos apenas um objeto, o predicado passa a ser uma característica do próprio objeto;
- é importante ressaltar que os nomes dos predicados e dos objetos devem sempre começar por letra minúscula;
- os predicados são escritos primeiro, seguido pelos objetos que são separados por vírgula e
- também é importante lembrar que a ordem dos objetos poderá interferir no resultado de uma aplicação.

PROLOG

- **variáveis:**
  - em Prolog uma variável não é um contêiner cujo valor pode ser atribuído como ocorre nas linguagens convencionais. Inicialmente uma variável é uma incógnita, mas quando atribuímos um objeto a ela, então a mesma não poderá mais ser modificada;
  - os nomes das variáveis devem começar sempre com letra maiúscula ou o caractere *underscore*;
- **conjunções e disjunções:**
  - para montarmos uma lógica mais específica pode-se utilizar os conceitos de conjunção e disjunção, ou seja, o equivalente ao "AND" e "OR" de outras linguagens e, para tanto, basta separar os fatos por **vírgulas** no caso da conjunção **AND** ou **ponto e vírgula** no caso da disjunção **OR**.

PROLOG

- Exemplo:
  - dados os seguintes fatos:

```
amiga(joana,maria).
amiga(clara,maria).
```

  - Podemos realizar a seguinte questão:

```
?- amiga(joana,X),amiga(clara,X).
```

- Neste exemplo foram fornecidos dois fatos e utilizado o operador de conjunção `AND` (`,`). Como `maria` é amiga de `joana` e `carla`, então a variável `X` receberá o objeto `maria`. E a questão retornará `yes`.
- Outro exemplo aproveitando os mesmos fatos:

```
?- amiga(joana,X);amiga(roberta,X).
```

- Nesse exemplo utilizou-se o operador de disjunção `OR` (`;`), ou seja, se uma das questões coincidirem com os fatos, será retornada `yes`.

PROLOG

- **regras:**
  - utiliza-se as regras para fazer a construção de questões complexas. Especifica-se situações que podem ser verdadeiras se algumas condições forem satisfeitas;
  - para criar uma regra usa-se o símbolo "`:-`", que indica uma condição (se). Exemplo simples:
    - dados os fatos:

```
pai(arthur,silvio).
pai(arthur,carlos).
pai(carlos,xico).
pai(silvio,ricardo).
```

  - utiliza-se a seguinte regra:

```
avo(X,Z) :- pai(X,Y), pai(Y,Z).
```

  - isso significa que se a pessoa `X` é pai de uma pessoa `Y` e, por sua vez, `Y` é pai de `Z`, então, `X` é avô de `Z`.

PROLOG

- Questão para conferir a regra:

```
?- avo(arthur,xico),avo(arthur,ricardo).
```

- retornará a saída: `true`

- Ou seja, `arthur` é avô de `xico` e `ricardo`, pois `arthur` é pai de `silvio` e `carlos`. Que por sua vez são pais de `xico` e `ricardo`.

PROLOG

- **operadores:**
  - pode-se utilizar operadores para construir regras ainda mais específicas. Operadores relacionais:
    - Igualdade: `=`
    - Diferença: `\=`
    - Menor que: `<`
    - Maior que: `>`
    - Menor ou igual: `=<`
    - Maior ou igual: `>=`
  - Exemplo:

```
positivo(Numero) :- Numero > 0.
```

  - Exemplo de consulta:

```
?- positivo(2).
```

  - retornará `yes`.

PROLOG

- Os operadores ariméticos são:

```
+, -, *, /, mod, is
```

- observe a seguinte regra:

```
o_dobro(Numero,Result) :- Result is Numero * 2.
```

- Questão:

```
?- o_dobro(5,X).
```

- Obtém-se o resultado:

```
X = 10
```

- é importante lembrar que a variável receberá o valor da operação aritmética através do operador `is`.

PROLOG

- **entrada e saída:**
  - como em outras linguagens, o Prolog também possui propriedades de entrada e saída de dados, são elas: `read()` e `write()`.
  - Exemplo:

```
ola :- read(X), write('Olá '), write(X).
```

  - execução:

```
?- ola. 'Edward'.
```

  - resultado:

```
Olá Edward
```

  - Observe que a regra não possuiu nenhum parâmetro, portanto coloca-se apenas o nome da regra e o dado a ser lido pelo comando `read`. Lembre-se que cada instrução é sempre finalizada por ponto.
  - É importante ressaltar que dados equivalentes ao tipo `String` de outras linguagens, devem sempre ser utilizados entre apóstrofos. Isto, para não serem confundidos com variáveis ou objetos.

PROLOG

- **comentário:**
  - Exemplo:

```
% Isto é um comentário de uma linha.
```

  - outro exemplo:

```
/*
Isto é um comentário que poderá ocupar mais de uma linha. Ou
seja, o tanto de linhas que for necessário.
*/
```

PROLOG

- O conteúdo a seguir foi extraído do material:
  BARANAUSKAS, José Augusto. **Teaching Material – Inteligência Artificial**. Disponível em: <https://dcm.ffclrp.usp.br/~augusto/teaching.htm>. Acesso em; 16 Ago. 2021.
- Linguagem de programação utilizada para resolver problemas envolvendo **objetos** e **relações** entre objetos;
- **conceitos básicos:** fatos, perguntas, variáveis, conjunções e regras;
- **conceitos avançados:** listas e recursão.

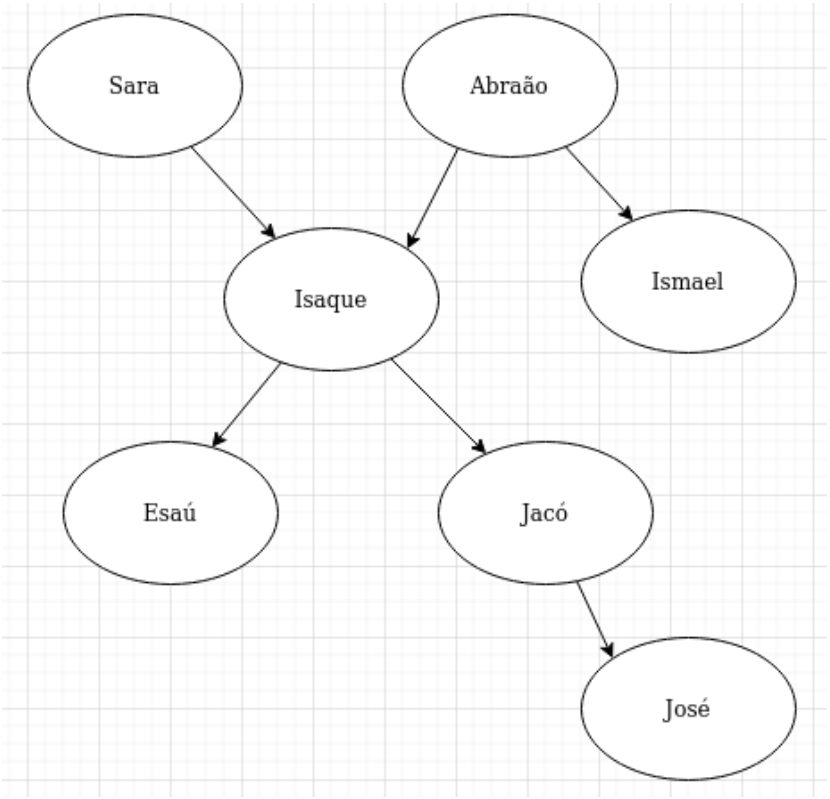
Comparativo

- Programação Convencional:
  - **programação procedural:**
    - programa = algoritmo + estruturas de dados;
  - **programação lógica:**
    - algoritmo = lógica + controle
    - programa = lógica + controle + estruturas de dados
    - em programação lógica programa-se de forma declarativa, ou seja, especificando-se o que deve ser processado, ao invés de como deve ser processado.

Programação PROLOG

- Envolve:
  - declarar **fatos** a respeito de objetos e seus relacionamentos;
  - definir algumas **regras** sobre os objetos e seus relacionamentos e
  - fazer **perguntas** sobre os objetos e seus relacionamentos.
- Para mostrar estes conceitos, será utilizado uma árvore genealógica e
- devido a limitações da linguagem não serão usados: acentos, cedilha ou caracteres especiais.

Definindo Relações por Fatos



- Observe que Abraão é pai (progenitor) de Isaque e de Ismael;
- em Prolog:

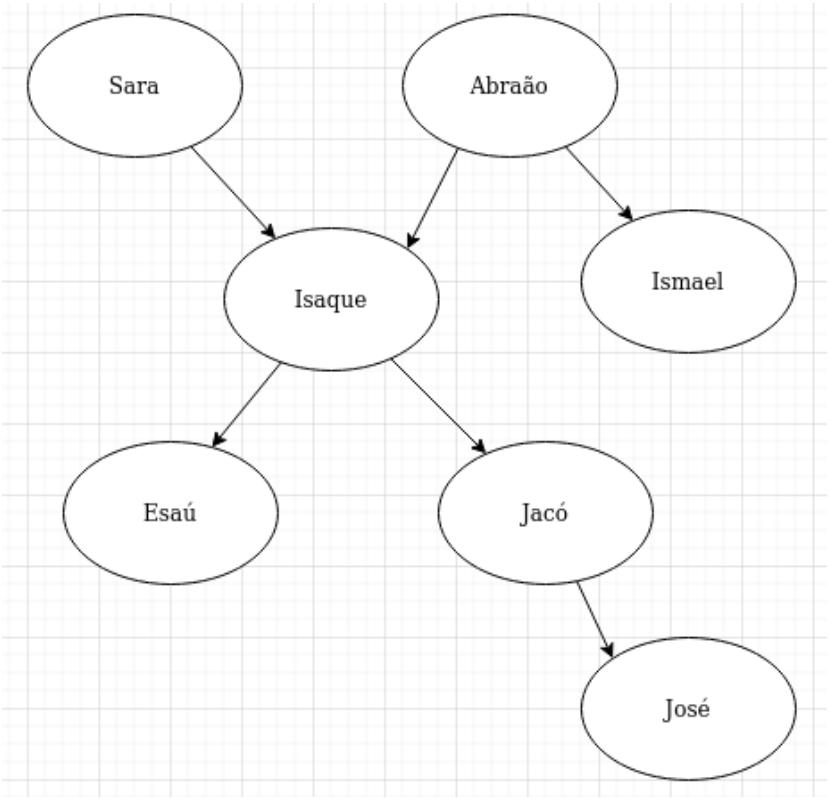
```
progenitor(abraao,isaque).
progenitor(abraao,ismael).
```

- neste:

  progenitor = nome de uma relação e

  abraão e **isaque** = argumentos (ou objetos).

Definindo Relações por Fatos

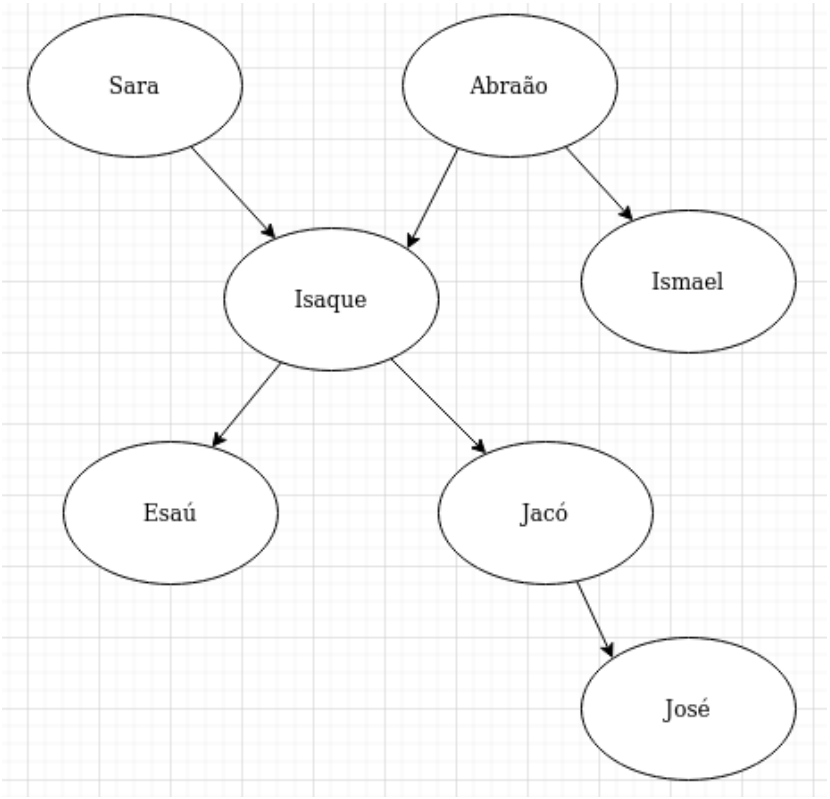


- A árvore completa em Prolog:

```
progenitor(sara,isaque).
progenitor(abraao,isaque).
progenitor(abraao,ismael).
progenitor(isaque,esau).
progenitor(isaque,jaco).
progenitor(jaco,jose).
```

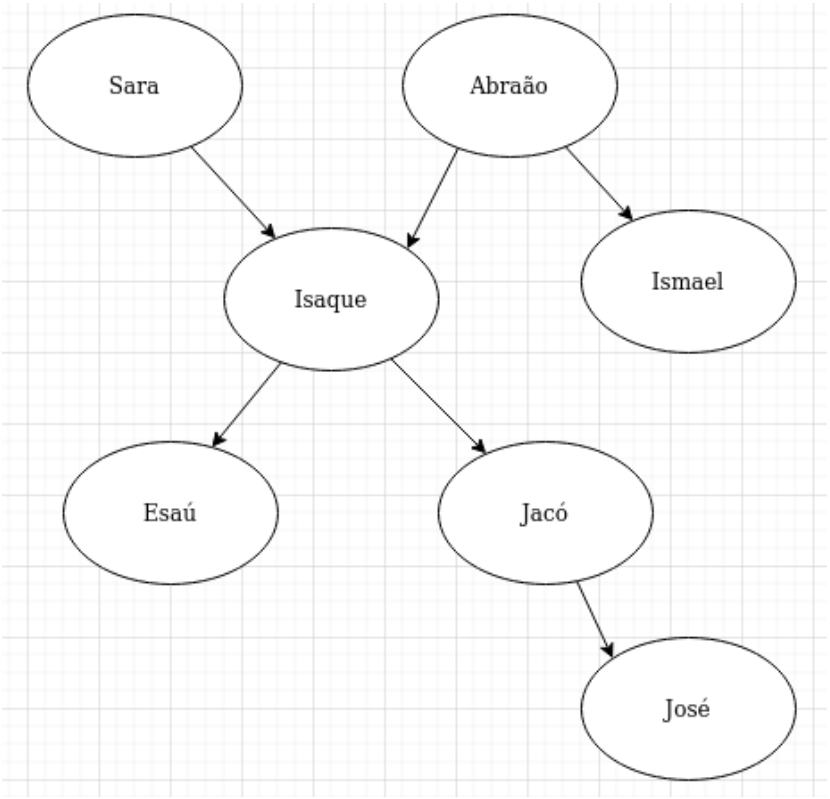
- este programa possui 6 **cláusulas**;
- cada cláusula declara um **fato** sobre a **relação** `progenitor`.

Definindo Relações por Fatos



- `progenitor(abraao,isaque).`
  - é uma **instância** particular da relação `progenitor`;
  - esta instância também é chamada de **relacionamento** e
  - uma **relação** é definida como o conjunto de todas as suas instâncias.

Atenção!



- A ordem dos argumentos, em uma relação deve ser consistente:
  - `progenitor(abraao,isaque).`
    - significa que “Abraão é progenitor de Isaque”;
  - `progenitor(isaque,abraao).`
    - significa que “Isaque é progenitor de Abraão”
  - ou seja:

`progenitor(abraao,isaque).`

$\neq$

`progenitor(isaque,abraao).`

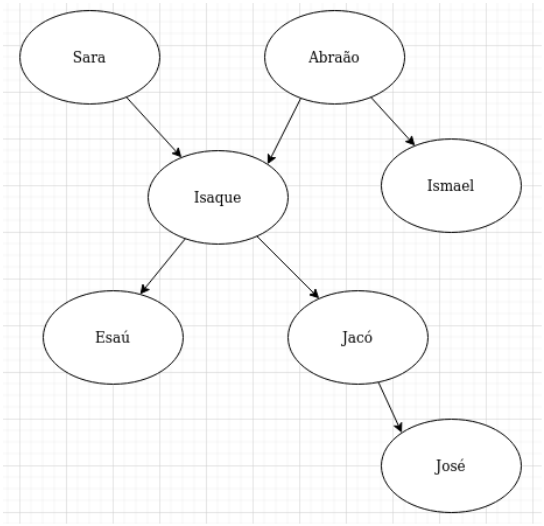
Definindo Relações por Fatos

- Os nomes das relações e seus argumentos são arbitrários, ou seja:

```
progenitor(abraao,isaque).
a(b,c).
```

- são semanticamente equivalentes desde que:
  - `a` signifique `progenitor`,
  - `b` signifique `abraão` e
  - `c` signifique `isaque`.

Exemplo de Funcionamento



- Base de dados

```
progenitor(sara,isaque).
progenitor(abraao,isaque).
progenitor(abraao,ismael).
progenitor(isaque,esau).
progenitor(isaque,jaco).
progenitor(jaco,jose).
```

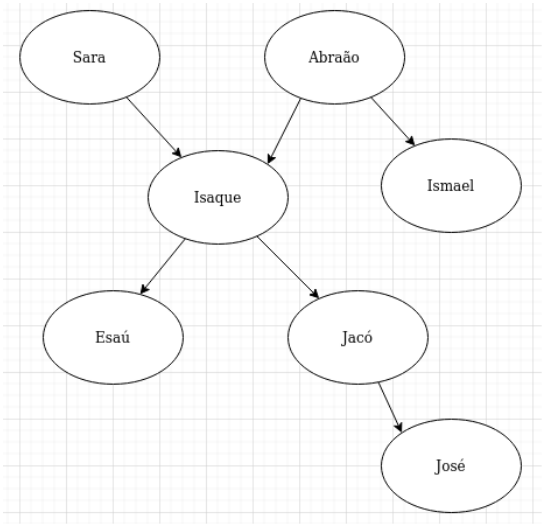
- Após a compilação da base de dados (ou base de fatos) pode-se questionar a PROLOG sobre a relação `progenitor`. Exemplo:
  - *Isaque é o pai de Jacó?*
- Esta pergunta em PROLOG é feita assim:

```
?- progenitor(isaque,jaco).
```

- a resposta de PROLOG será:

```
yes
```

Exemplo de Funcionamento



- Base de dados

```
progenitor(sara,isaque).
progenitor(abraao,isaque).
progenitor(abraao,ismael).
progenitor(isaque,esau).
progenitor(isaque,jaco).
progenitor(jaco,jose).
```

- **Outra pergunta:**

```
?- progenitor(ismael,jaco).
```

- a resposta:

```
no
```

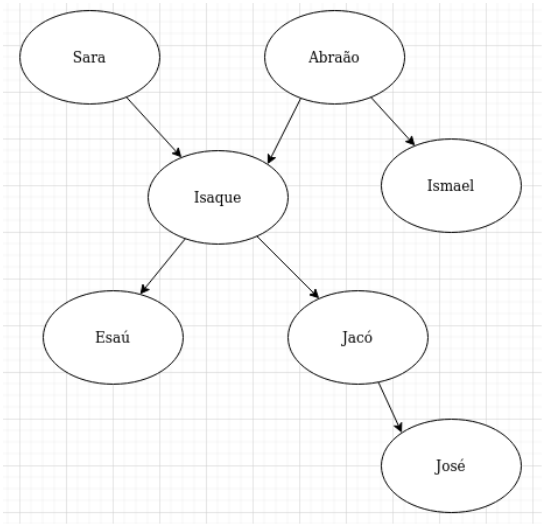
- esta resposta se dá devido ao fato de não existir a relação “Ismael como progenitor de Jacó” na base de dados.
- **Outra pergunta:**

```
?- progenitor(jaco,moises).
```

- a resposta:

```
no
```

Exemplo de Funcionamento



- Base de dados

```
progenitor(sara,isaque).
progenitor(abraao,isaque).
progenitor(abraao,ismael).
progenitor(isaque,esau).
progenitor(isaque,jaco).
progenitor(jaco,jose).
```

- **Pode-se modificar a pergunta para:** *quem é o progenitor de Ismael?*

```
?- progenitor(X,ismael).
```

- a resposta:

```
X = abraao
```

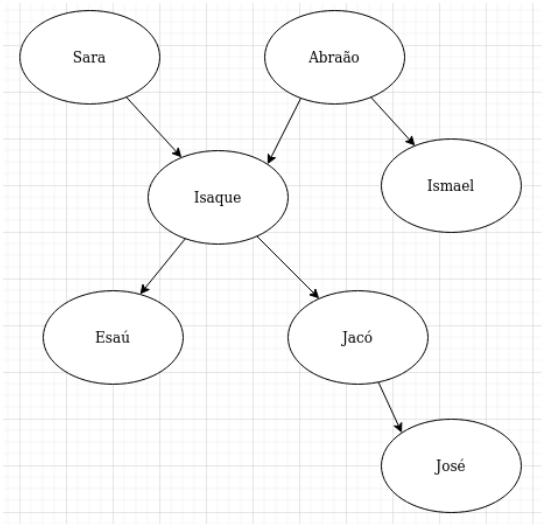
- **Outra pergunta:**

```
?- progenitor(R,jaco).
```

- a resposta:

```
R = isaque
```

Exemplo de Funcionamento



- Base de dados

```
progenitor(sara,isaque).
progenitor(abraao,isaque).
progenitor(abraao,ismael).
progenitor(isaque,esau).
progenitor(isaque,jaco).
progenitor(jaco,jose).
```

- **Outra pergunta:** *quais são os filhos de Isaque?*

```
?- progenitor(isaque,K).
```

- neste caso há mais de uma resposta possível então PROLOG responde com uma solução:

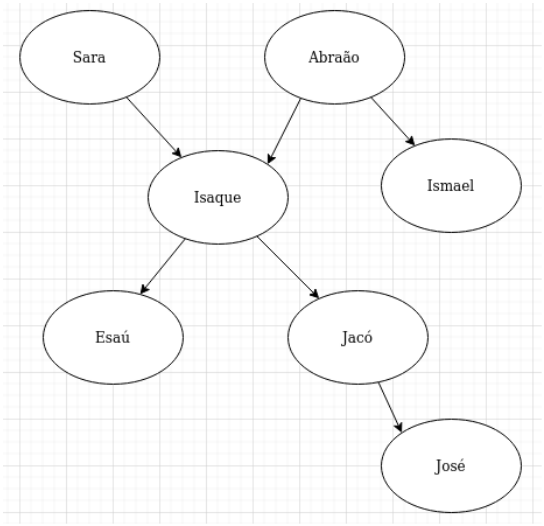
```
K = esau
```

- pode-se requisitar outra solução pressionando-se a tecla "`;`" (ponto e vírgula). Automaticamente será mostrado:

```
K = jaco
```

- e `yes`, pois todas as soluções já foram exauridas.

Exemplo de Funcionamento



- Base de dados

```
progenitor(sara,isaque).
progenitor(abraao,isaque).
progenitor(abraao,ismael).
progenitor(isaque,esau).
progenitor(isaque,jaco).
progenitor(jaco,jose).
```

- **Outra pergunta:** *quem é o progenitor de quem?*

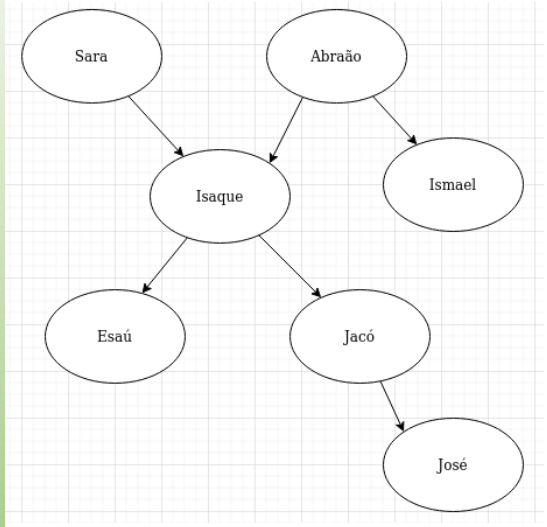
```
?- progenitor(P,S).
```

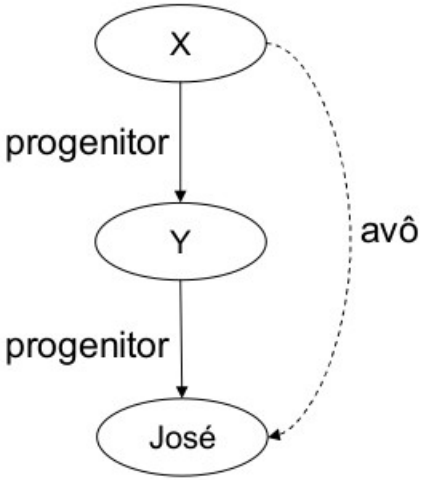
- PROLOG encontra todos os pares progenitor-filho, um após o outro. As soluções são mostradas uma de cada vez:

```
P = sara     S = isaque ?;

P = abraao   S = isaque ?;

P = abraao   S = ismael ?;

...
```

- todas as soluções serão vistas se apertarmos a tecla "`a`" ao invés de "`;`". Caso queira interromper a apresentação aperte `Enter`.

Exemplo com Conjunção AND





- Base de dados

```
progenitor(sara,isaque).
progenitor(abraao,isaque).
progenitor(abraao,ismael).
progenitor(isaque,esau).
progenitor(isaque,jaco).
progenitor(jaco,jose).
```

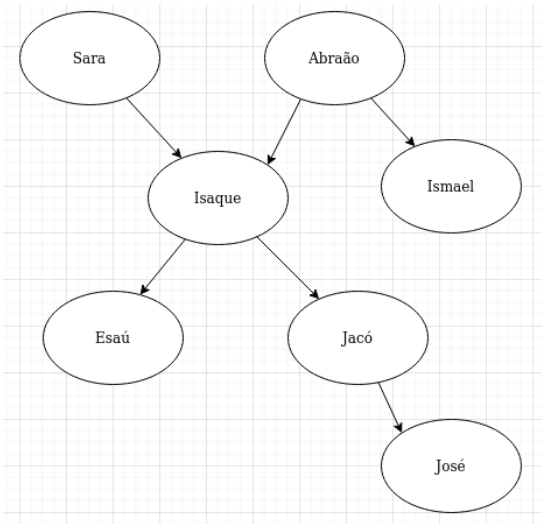
- **Pergunta:** *quem é o avô de José?*
- na base de dados não está especificada a relação "**avô**". Assim sendo, PROLOG não sabe nos responder esta pergunta. Temos que desmembrar a pergunta em duas partes:
  - quem é o progenitor de José? (Y)
  - quem é o progenitor de Y? (X)
- estas perguntas em PROLOG:

```
?- progenitor(Y,jose),
progenitor(X,Y).
```

- resposta:

```
X = isaque

Y = jaco
```

Exemplo com Conjunção AND



- Base de dados

```
progenitor(sara,isaque).
progenitor(abraao,isaque).
progenitor(abraao,ismael).
progenitor(isaque,esau).
progenitor(isaque,jaco).
progenitor(jaco,jose).
```

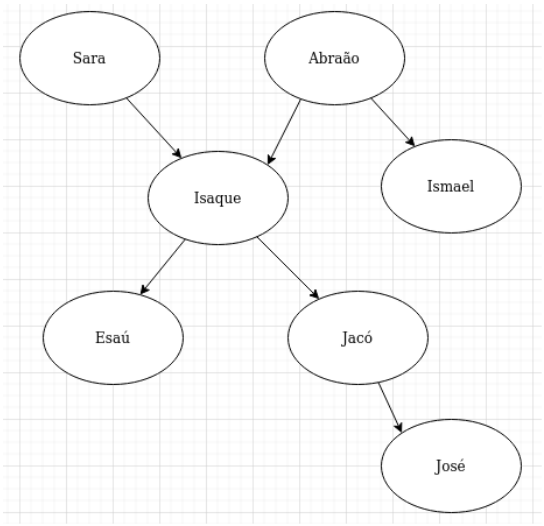
- **Pergunta:** *quem são os netos de Abraão?*
- na base de dados não está especificada a relação "**neto**". Assim sendo, PROLOG tampouco nos sabe responder esta pergunta. Temos que desmembrar a pergunta em duas partes:

```
?- progenitor(abraao,S),
progenitor(S,G).
```

- respostas:

```
S = isaque    G = esau ? a

S = isaque    G = jaco

no
```

Exemplo com Conjunção AND



- Base de dados

```
progenitor(sara,isaque).
progenitor(abraao,isaque).
progenitor(abraao,ismael).
progenitor(isaque,esau).
progenitor(isaque,jaco).
progenitor(jaco,jose).
```

- **Outro tipo de pergunta:** *Esaú e Jacó têm o mesmo pai (progenitor)?*
- Esta pergunta também pode ser expressa em duas etapas:
  - *quem é o progenitor, X, de Esaú?*
  - *É este mesmo X o progenitor de Jacó?*
- Em PROLOG:

```
?- progenitor(X,esau),
progenitor(X,jaco).
```

- resposta:

```
X = isaque
```

Ampliando a Base de Dados

- Base de dados

```
progenitor(sara,isaque).
progenitor(abraao,isaque).
progenitor(abraao,ismael).
progenitor(isaque,esau).
progenitor(isaque,jaco).
progenitor(jaco,jose).
mulher(sara).
homem(abraao).
homem(isaque).
homem(ismael).
homem(esau).
homem(jaco).
homem(jose).
```

- A relação **mulher** e **homem** são unárias;
- como já visto, isto define uma característica do objeto na relação.
- Ou seja, *Sara é uma mulher.*

Pode-se fazer das duas formas:

- Informação sobre o sexo das pessoas envolvidas na relação `progenitor`:

```
mulher(sara).

homem(abraao).

homem(isaque).

homem(ismael).

homem(esau).

homem(jaco).

homem(jose).
```

- Informação sobre o sexo das pessoas envolvidas na relação `progenitor`, escrita em uma relação binária:

```
sexo(sara,feminino).

sexo(abraao,masculino).

sexo(isaque,masculino).

sexo(ismael,masculino).

sexo(esau,masculino).

sexo(jaco,masculino).

sexo(jose,masculino).
```

Escolhendo Objetos e Relações

- Como representar "*Sara é uma mulher.*" ?

  `mulher(sara).`

  - Permite responder: *quem é mulher?*
  - Não permite responder: *qual é o sexo de Sara?*

  `sexo(sara,feminino).`

  - Permite responder:
    - *Quem é mulher?*
    - *Qual é o sexo de Sara?*
  - Não permite responder:
    - *Qual a propriedade de Sara que tem o valor feminino?*

  `propriedade(sara,sexo,feminino).`

  - Permite responder todas as perguntas e é
  - a representação: **objeto-atributo-valor**.

Definindo Outras Relações

- Pode-se estender a base de dados introduzindo a relação **filho** como sendo o inverso de progenitor;
- ou seja, pode-se criar uma lista que inverte a relação progenitor:

```
filho(isaque,sara).
filho(isaque,abraao).
filho(ismael,abraao).
...
```

- Porém a relação **filho** pode ser definida de modo mais inteligente:
  - *para todo X e Y,*

    Y é um filho de X,

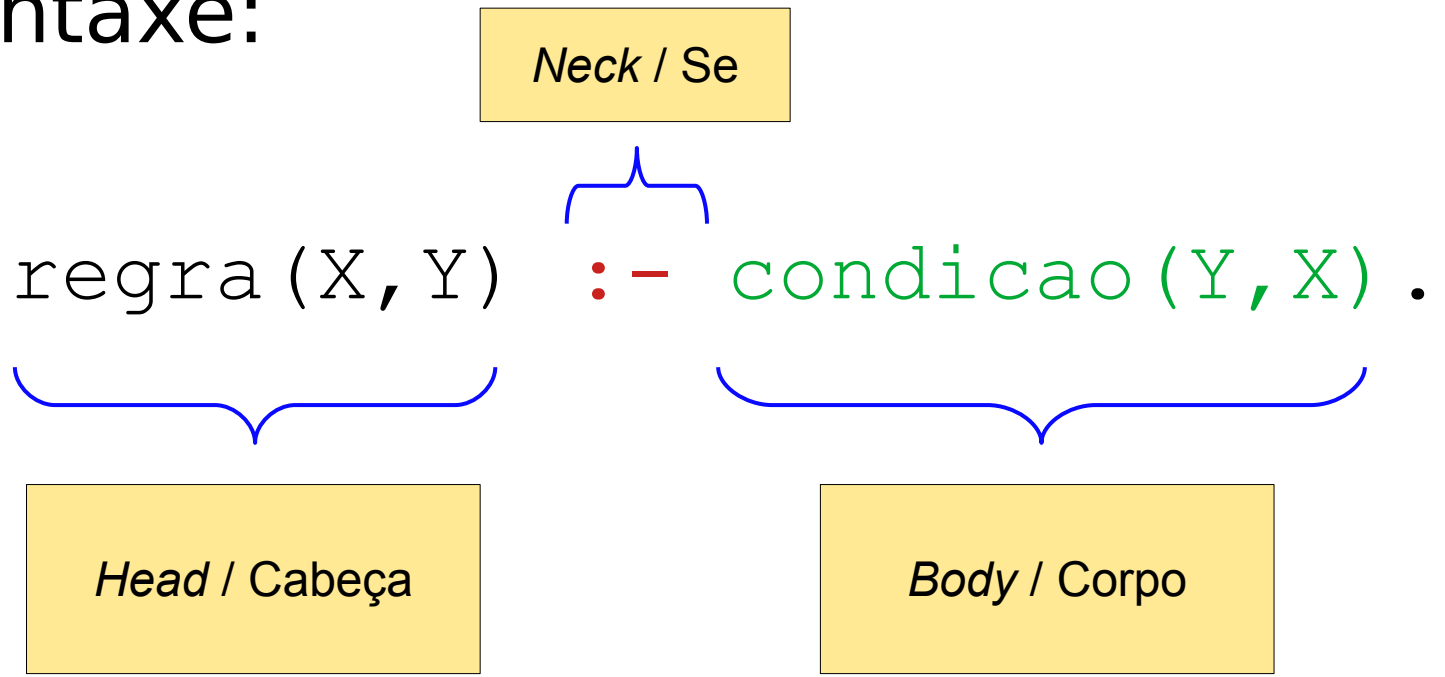
    se X é progenitor de Y.
- Em PROLOG:

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

Isto é uma "regra".

Regras (*rules*)

Sintaxe:



se a condição for satisfeita, então a `regra` é verdadeira.

Regra `filho`

- Definindo a regra **filho**:

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

- Diferença entre **fatos** e **regras**:
  - um fato é sempre verdadeiro;
  - uma regra especifica algo que só é verdadeiro se uma condição for satisfeita.

Regra `mae`

- Definindo a regra **mãe**, com base no seguinte fundamento lógico:
  - *para todo X e Y,*

    X é mãe de Y se X é progenitor de Y e

    X é mulher.
- Em PROLOG:

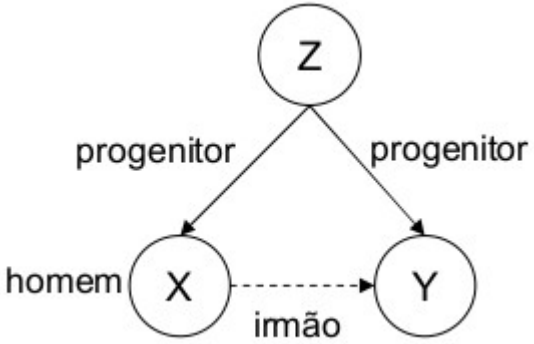
```
mae(X,Y) :- progenitor(X,Y),mulher(X).
```

- se ambas as condições forem verdadeiras, então a regra `mae` retornará `yes`.

Regra `irmao`

- Definindo a regra **irmão**, com base no seguinte fundamento lógico:
  - *para todo X e Y,*

    X é irmão de Y se ambos, X e Y, têm o mesmo progenitor e X é homem.
- Em PROLOG:
  - `irmao(X,Y) :- progenitor(Z,X), progenitor(Z,Y), homem(X).`
- se ambas as condições forem verdadeiras, então a regra `irmao` retornará `yes`.
- Pode-se perguntar: `?- irmao(esau,jaco).`
- Resposta: `true`.



Falha na Regra `irmao`

- Pergunta: *quem é o irmão de Jacó?*

```
?- irmao(X,jaco).
```

- Resposta:

```
X = esau;
X = jaco
```

- assim, Jacó fica sendo irmão dele mesmo. Isto não era bem o que tínhamos em mente quando definimos a regra `irmao`. Mas, de acordo com a definição a resposta de Prolog é perfeitamente lógica.

Falha na Regra `irmao`

- Pergunta: *quem é o irmão de Jacó?*

```
?- irmao(X,jaco).
```

- Resposta:

```
X = esau;
X = jaco
```

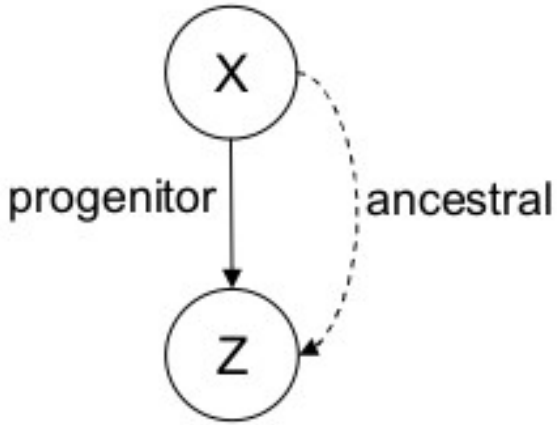
- Jacó não é seu próprio irmão, então, como consertar isto

Regras Recursivas

- Vamos criar a regra **ancestral**;
- esta relação será definida por duas regras:
  - a primeira sem recursividade e
  - a segunda com recursividade;
- *Para todo X e Z,*

  X é um ancestral de Z se

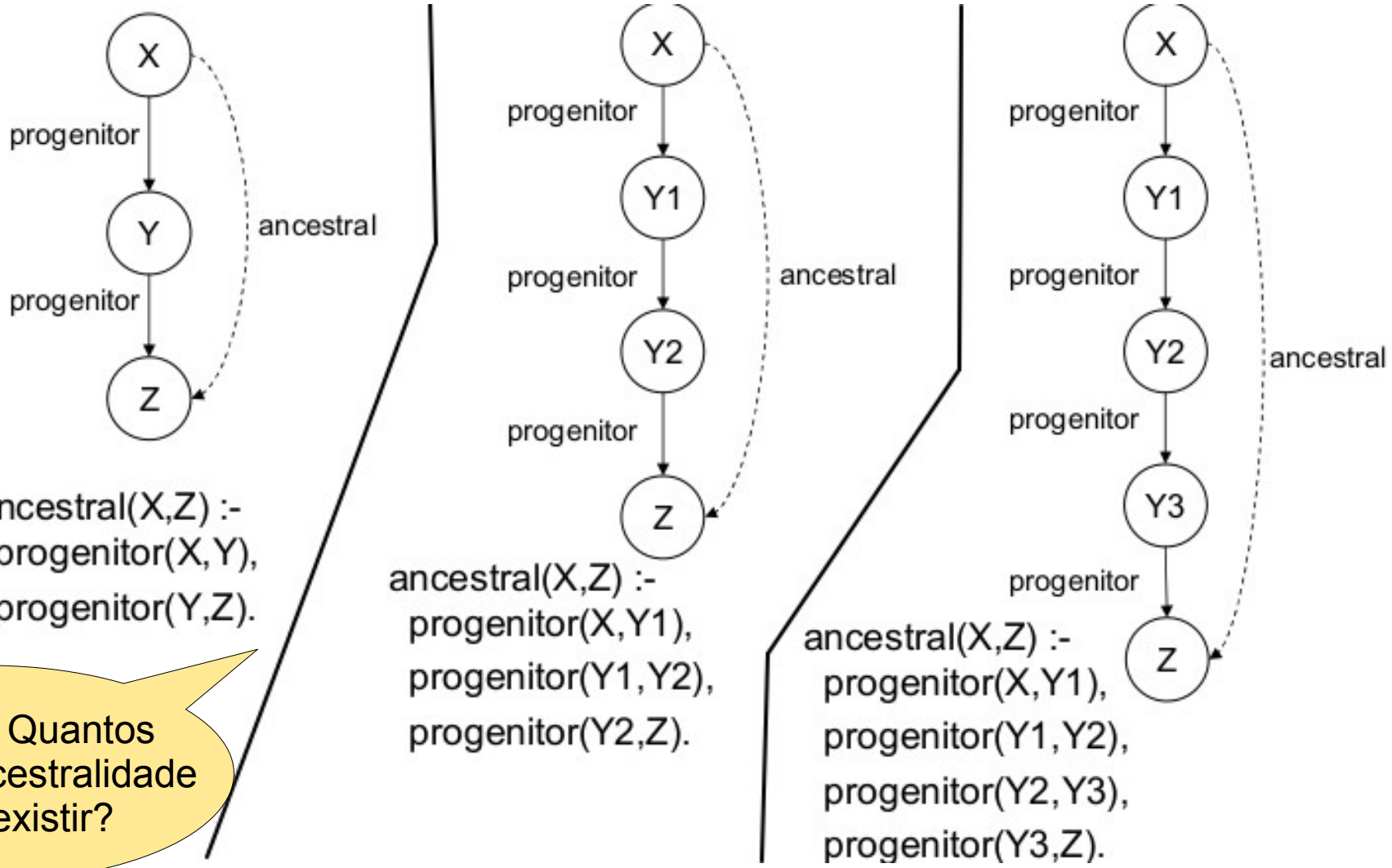
  X é um progenitor de Z.



- Em PROLOG:

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

Regras Recursivas



Problema! Quantos níveis de ancestralidade podem existir?

Regras Recursivas

- Para resolver, de uma vez por todas, o problema da ancestralidade, podemos usar a seguinte definição:
  - *para todo X e Z,*

    X é ancestral de Z se há algum Y tal que

    X é um progenitor de Y e

    Y é um ancestral de Z.
- Em PROLOG:

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

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

Base de Dados Família de Abraão

- Base de dados

```
progenitor(sara,isaque).
progenitor(abraao,isaque).
progenitor(abraao,ismael).
progenitor(isaque,esau).
progenitor(isaque,jaco).
progenitor(jaco,jose).
mulher(sara).
homem(abraao).
homem(isaque).
homem(ismael).
homem(esau).
homem(jaco).
homem(jose).
```

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

Bibliografia

- BAFFA, Augusto. **Prolog Tutotial**. Disponível em: <http://www.augustobaffa.pro.br/wiki/Prolog_Tutorial>. Acesso em: 20 Out. 2021.
- BARANAUSKAS, José Augusto. **Teaching Material – Inteligência Artificial**. Disponível em: <https://dcm.ffclrp.usp.br/~augusto/teaching.htm>. Acesso em; 16 Ago. 2021.
- BITTENCOURT, Guilherme. **Inteligência Artificial - Ferramentas e Teorias**. 2. ed. Florianópolis: Ed. da UFSC, 2001.
- DANTAS, L. A. **Descobrindo o Prolog**. Disponível em: <http://www.linhadecodigo.com.br/artigo/1697/descobrindo-o-prolog.aspx>. Ascesso em: 01 Nov. 2021.