Introdução à Programação Prolog (Tutorial)

Inteligência Artificial

- Esta aula introduz conceitos básicos da linguagem de programação lógica Prolog
- Os conceitos são introduzidos através de um tutorial sobre relações familiares
- Maiores detalhes sobre terminologia e notação serão vistos nas próximas aulas

Introdução

- Prolog = Programming in Logic
- 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

Programação Lógica

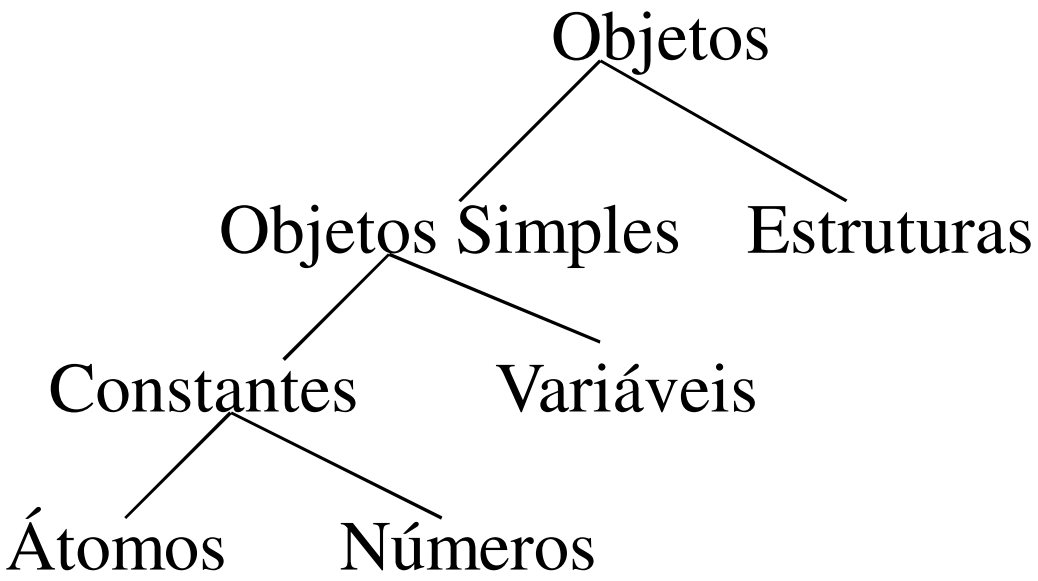
- Programação Procedural (procedimental):
  - Programa = Algoritmo + Estruturas de Dados
- Programação Lógica
  - Algoritmo = Lógica + Controle
  - Programa = Lógica + Controle + Estruturas de Dados
  - Em PL, programa-se de forma ***declarativa***, ou seja, especificando ***o que*** deve ser computado ao invés de ***como*** deve ser computado

Programação em Prolog

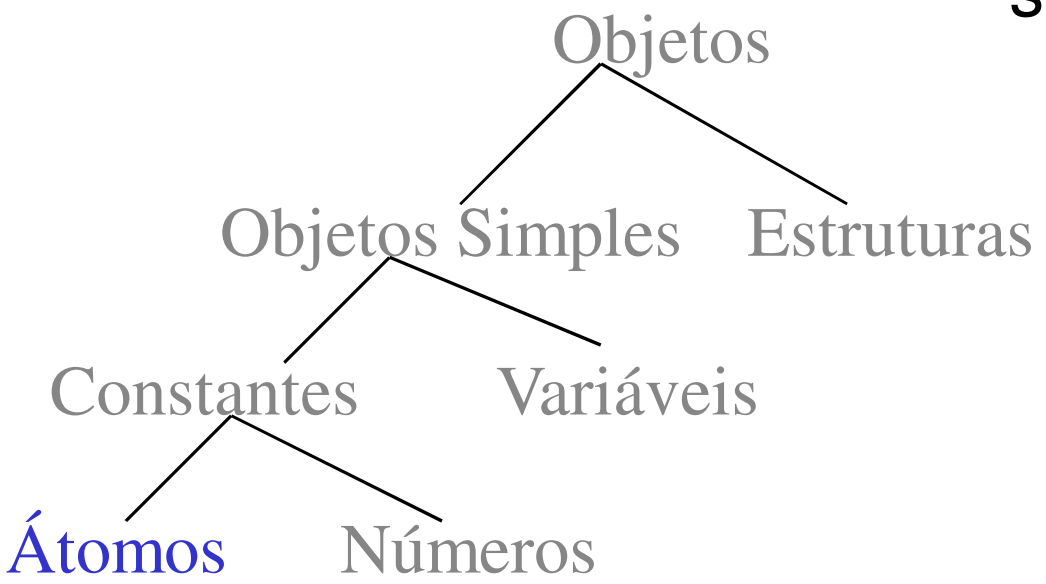
Programar em Prolog envolve:

- declarar alguns ***fatos*** a respeito de objetos e seus relacionamentos
- definir algumas ***regras*** sobre os objetos e seus relacionamentos e
- fazer ***perguntas*** sobre os objetos e seus relacionamentos

Objetos de dados e Prolog



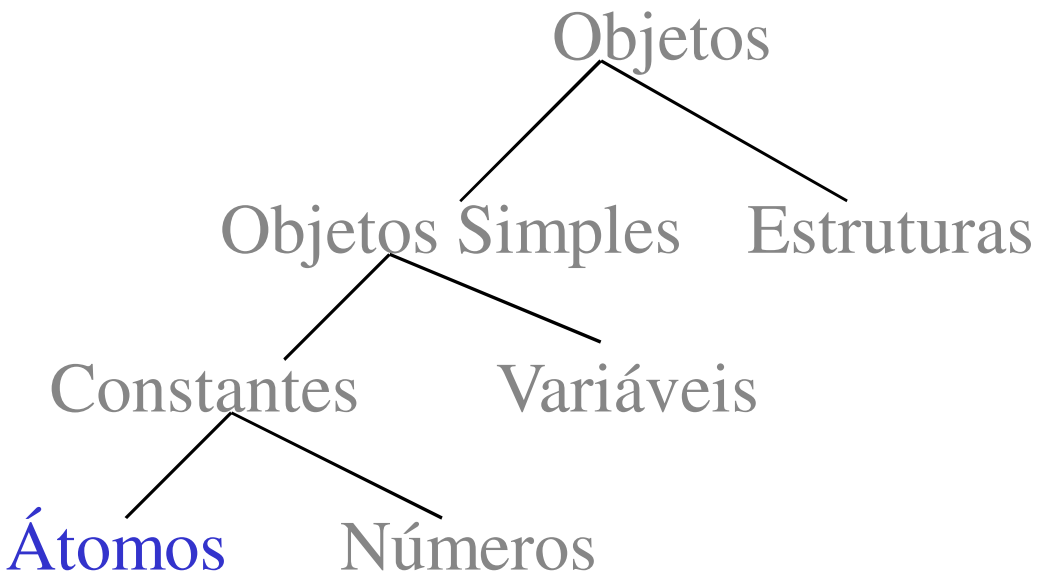
Átomos



São cadeias compostas pelos seguintes caracteres:

- letras maiúsculas: A, B, ..., Z
- letras minúsculas: a, ..., z
- dígitos: 1, 2, ..., 9
- caracters especiais, tais como: *, +, >, _, =, :, ., &.

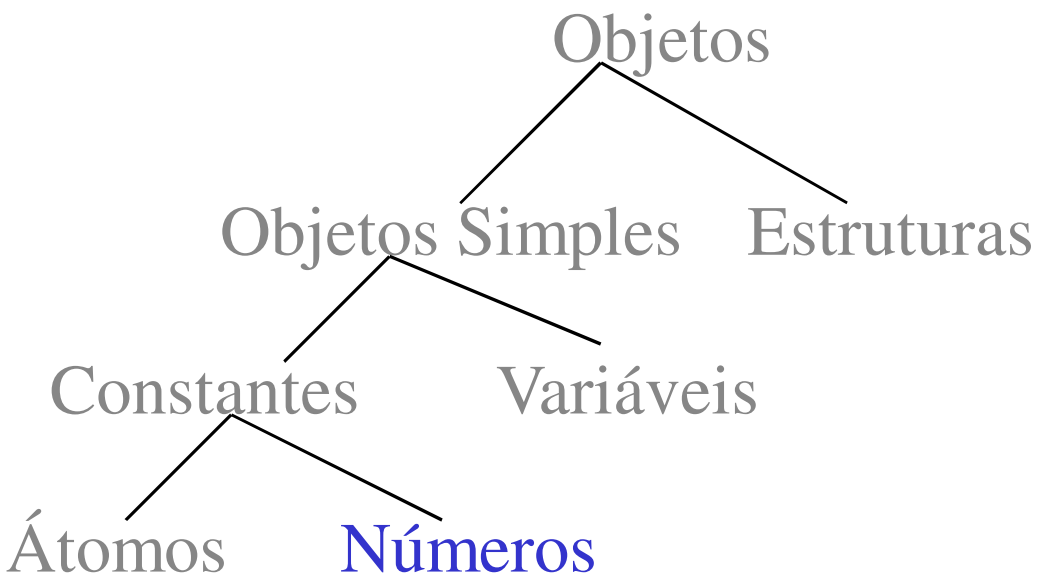
Átomos (cont)



Podem ser construídos de três maneiras:

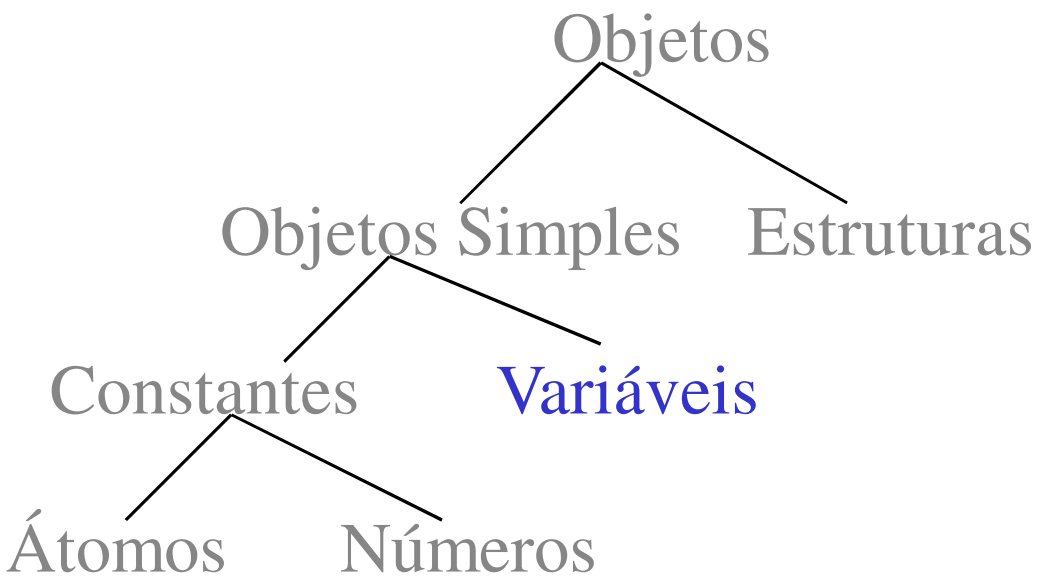
- cadeias de letras, dígitos e o caractere _, começando com uma letra minúscula.
- cadeias de caracteres especiais
- cadeias de caracteres entre apóstrofos

Números



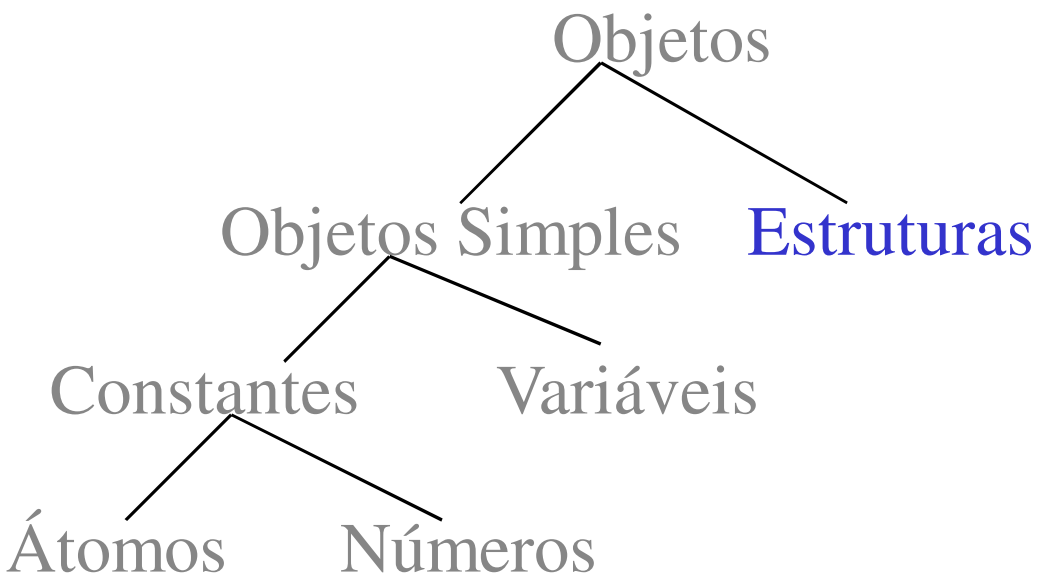
Incluem números inteiros e números reais.

Variáveis



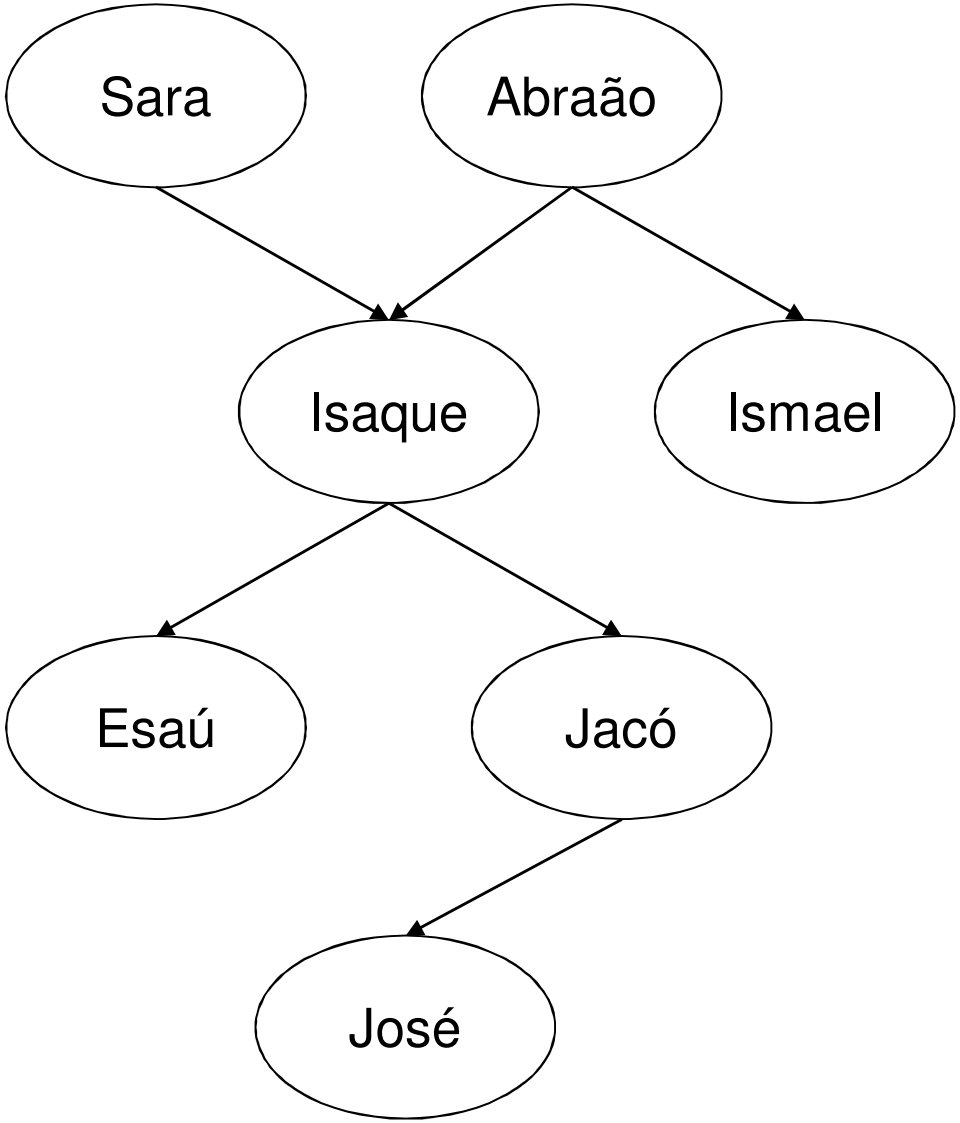
São cadeias de letras, dígitos e caracteres _, sempre começando com letra maiúscula ou com o caractere _.

Estruturas



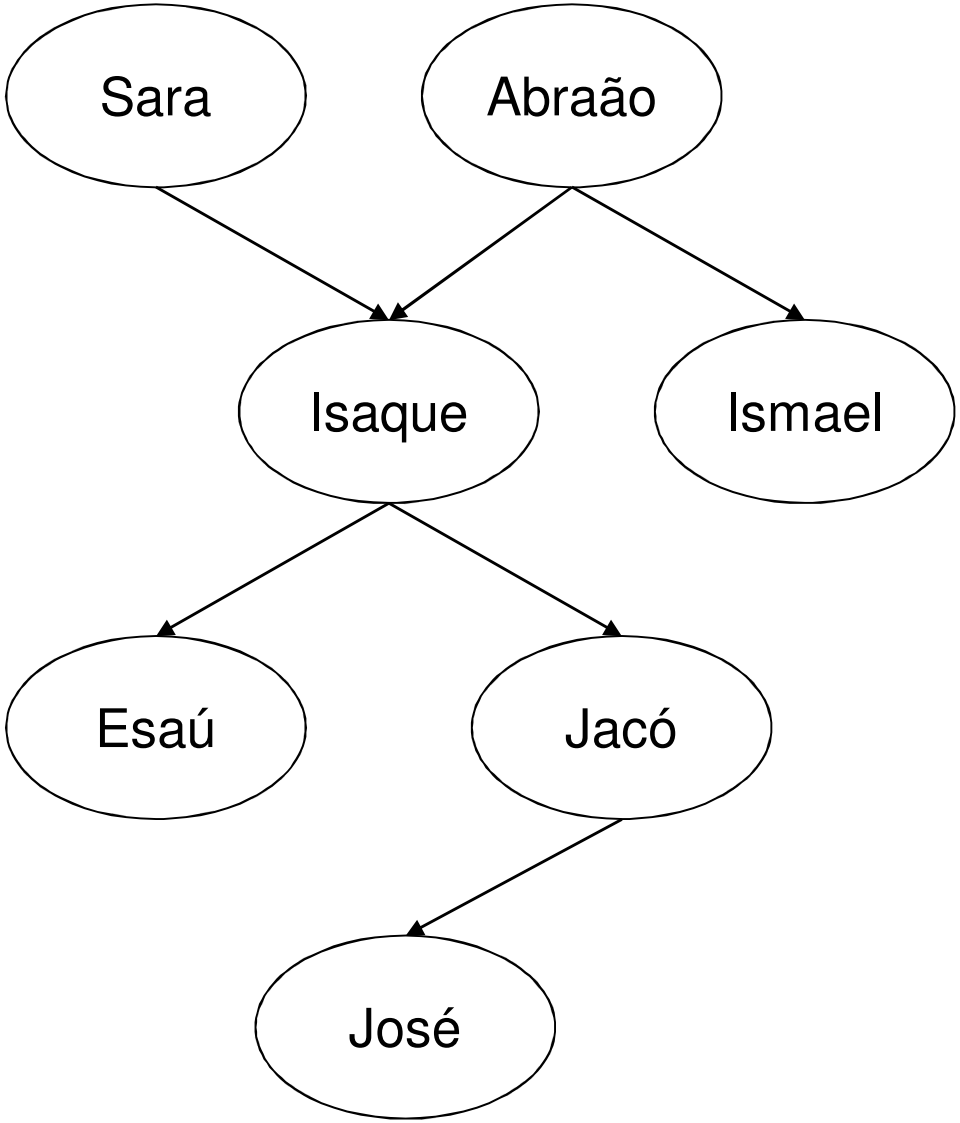
São objetos de dados que têm vários componentes, podendo cada um deles, por sua vez, ser uma estrutura.

Definindo Relações por Fatos



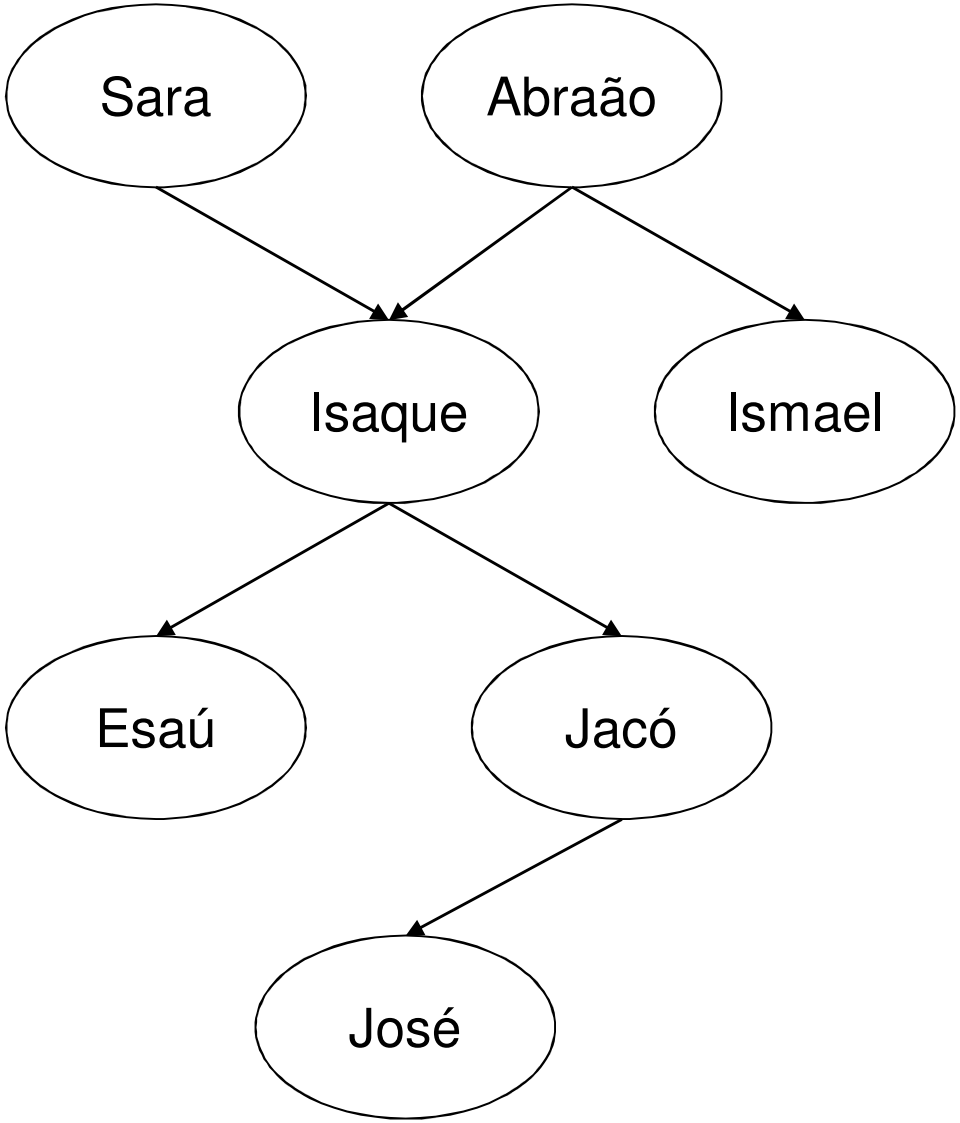
- A figura ao lado mostra um exemplo da relação *família*
- O fato que Abraão é um progenitor de Isaque pode ser escrito em Prolog como:
  - progenitor(abraão,isaque).
- Neste caso definiu-se **progenitor** como o nome de uma relação; **abraão** e **isaque** são seus argumentos

Definindo Relações por Fatos



- A árvore familiar completa em Prolog é:
  - progenitor(sara,isaque).
  - progenitor(abraão,isaque).
  - progenitor(abraão,ismael).
  - progenitor(isaque,esaú).
  - progenitor(isaque,jacó).
  - progenitor(jacó,josé).
- Este programa consiste de seis ***cláusulas***
- Cada uma dessas cláusulas declara um fato sobre a ***relação*** progenitor

Definindo Relações por Fatos



- Por exemplo
  - progenitor(abraão,isaque).

  é uma ***instância*** particular da relação progenitor
- Esta instância é também chamada de ***relacionamento***
- Em geral, uma ***relação*** é definida como o conjunto de todas suas instâncias

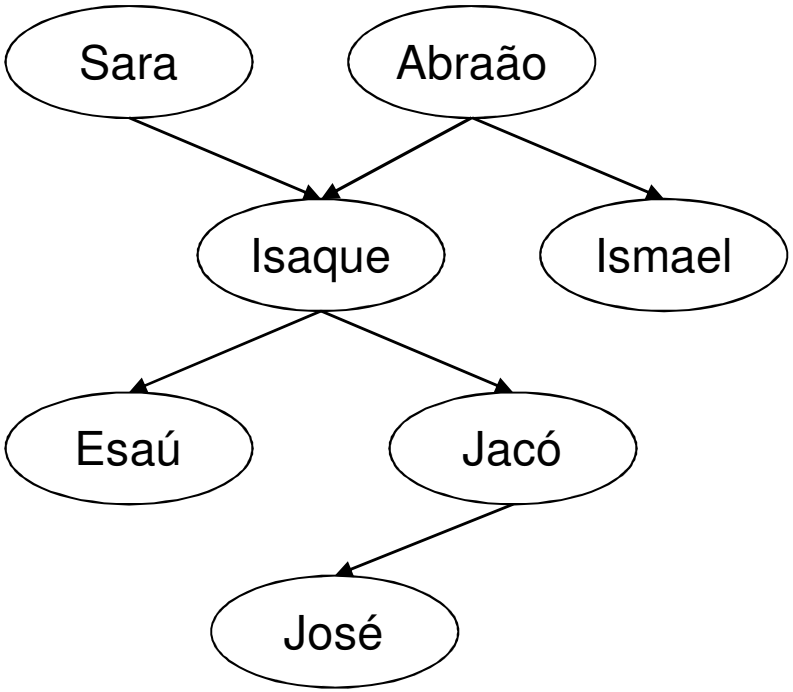
Definindo Relações por Fatos

- A ordem dos argumentos em uma relação é definida arbitrariamente, mas deve ser seguida e usada de forma consistente
- progenitor(abraão,isaque)
  - significa que “Abraão é progenitor de Isaque”
- progenitor(isaque,abraão)
  - significa que “Isaque é progenitor de Abraão”
- Note que progenitor(abraão,isaque) não tem o mesmo significado que progenitor(isaque,abraão)

Definindo Relações por Fatos

- Os nomes das relações e seus argumentos são arbitrários, ou seja:
  - progenitor(abraão,isaque)
  - a(b,c)
- são semanticamente equivalentes desde que
  - “a” signifique “progenitor”
  - “b” signifique “abraão” e
  - “c” signifique “isaque”
- Normalmente, o programador escolhe nomes significativos

Definindo Relações por Fatos



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

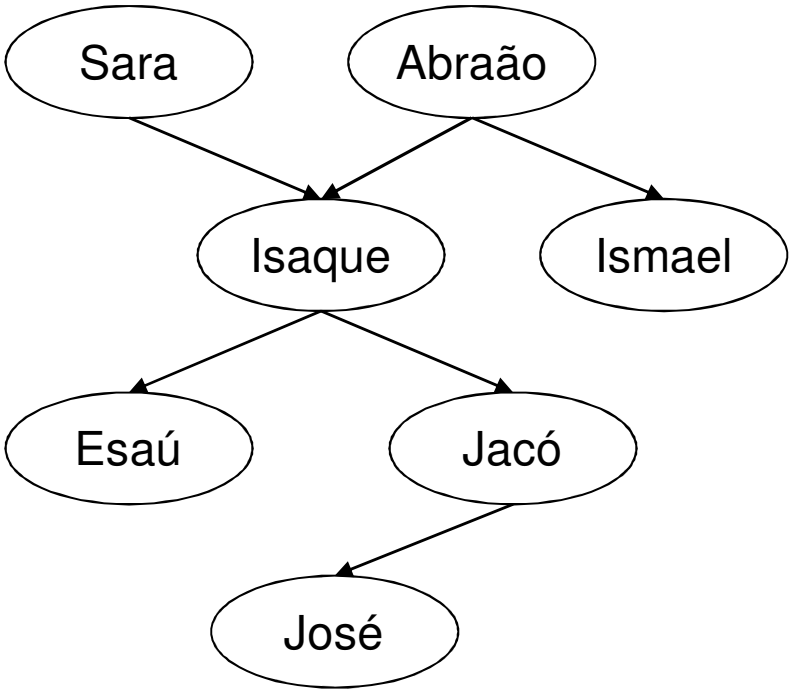
- Quando o programa é interpretado/compilado, pode-se questionar Prolog sobre a relação progenitor, por exemplo: Isaque é o pai de Jacó?
- Esta pergunta pode ser comunicada à Prolog digitando:

```
?- progenitor(isaque,jacó).
```

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

```
yes
```

Definindo Relações por Fatos



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

- Uma outra pergunta pode ser

```
?- progenitor(ismael,jacó).
```

- Prolog responde:

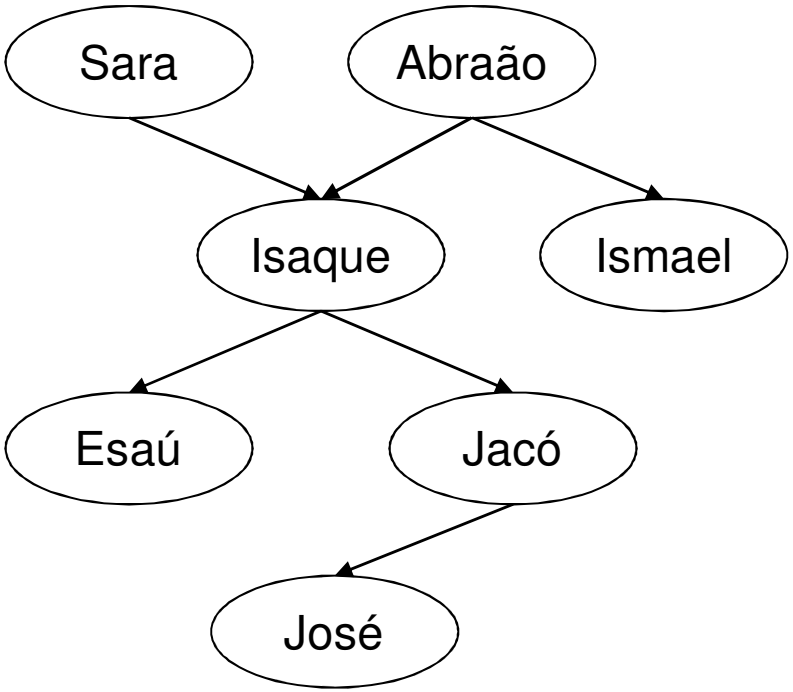
```
no
```

porque o programa não menciona nada sobre Ismael como sendo o progenitor de Jacó

- Prolog também responde `no` à pergunta:

```
?- progenitor(jacó,moisés).
no
```

Definindo Relações por Fatos



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

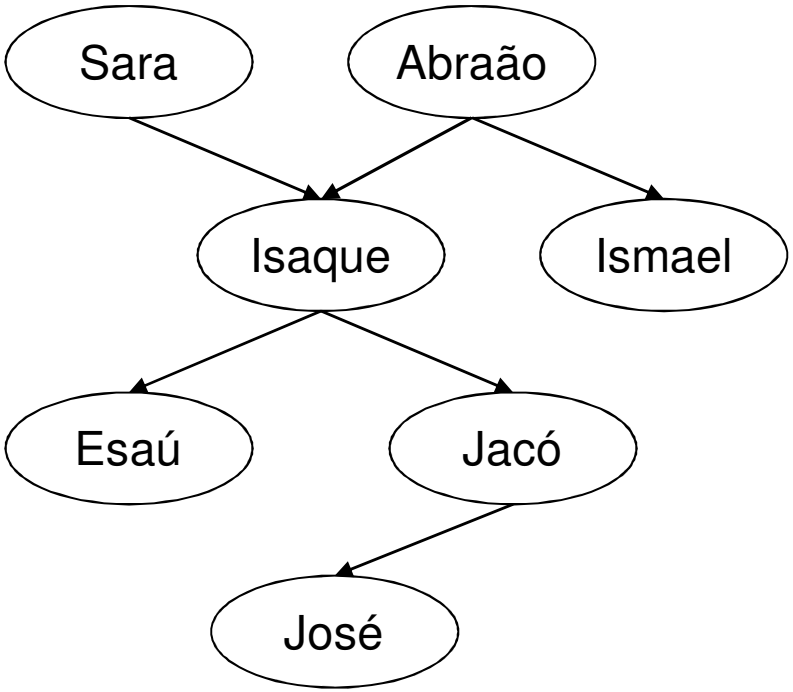
- Perguntas mais interessantes também podem ser efetuadas: Quem é o progenitor de Ismael?

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

- Neste caso, Prolog não vai responder apenas **yes** ou **no**. Prolog fornecerá o valor de X tal que a pergunta acima seja verdadeira
- Assim a resposta é:

```
X = abraão
```

Definindo Relações por Fatos




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

- A pergunta "Quais os filhos de Isaque?" pode ser escrita como:

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

- Neste caso, há mais de uma resposta possível; Prolog primeiro responde com uma solução:

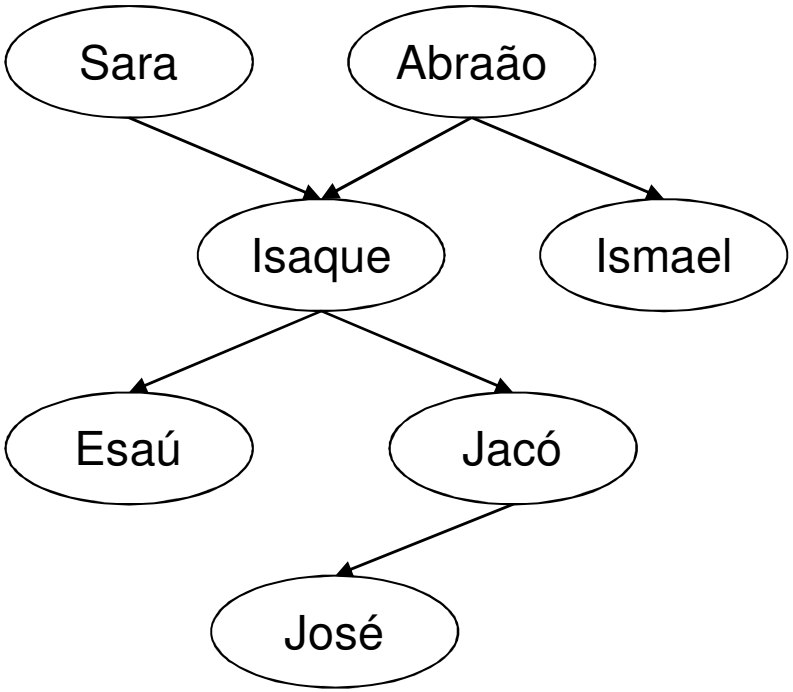
```
X = esaú
```

- Pode-se requisitar uma outra solução (digitando `;`) e Prolog encontra:

```
X = jacó
```

- Se mais soluções forem requisitadas, Prolog responde **`no`** pois todas as soluções foram exauridas (**`no`** = sem mais soluções)

Definindo Relações por Fatos



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

- Questões mais amplas podem ser efetuadas: Quem é o progenitor de quem?
- Reformulando: encontre X e Y tais que X é o progenitor de Y

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

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

```
X = sara        Y = isaque;
X = abraão      Y = isaque;
X = abraão      Y = ismael;
...
```

- As soluções podem ser interrompidas digitando **[enter]** ao invés de **;**

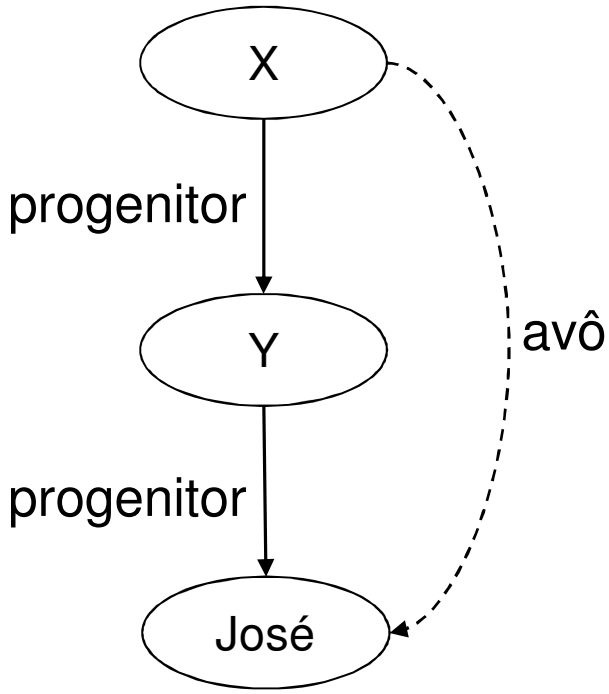
Definindo Relações por Fatos

- Perguntas mais complexas também podem ser efetuadas, tais como: Quem é o avô de José?
- Como nosso 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

- A pergunta composta

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

- Pode ser lida como:
  - Encontre X e Y tais que satisfaçam os seguintes requisitos
  - `progenitor(Y,josé)` e `progenitor(X,Y)`
- 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
```

Definindo Relações por Fatos

- Outro tipo de pergunta pode ser efetuado: Esaú e Jácó têm um progenitor em comum?
- Isso pode ser expresso em duas etapas:
  - Quem é o progenitor, X, de Esaú?
  - É (este mesmo) X um progenitor de Jacó?
- A pergunta correspondente em Prolog é:

```
?- progenitor(X,esaú), progenitor(X,jacó).
X = isaque
```

Pontos Importantes

- O nome de uma relação deve começar com uma letra minúscula
- A relação é escrita primeiro e os seus argumentos são separados por vírgulas e colocados entre parênteses
- O ponto final "." deve seguir o final do fato
- É fácil definir uma relação em Prolog, por exemplo a relação ***progenitor***, escrevendo n-tuplas de objetos que satisfazem a relação
- O usuário pode perguntar ao sistema Prolog sobre relações definidas no programa

Pontos Importantes

- Um programa Prolog consiste de ***cláusulas***; cada cláusula termina com um ponto final
- Os argumentos das relações podem (entre outras coisas) ser: objetos concretos ou constantes (tais como ***abraão*** e ***isaque***) ou objetos gerais tais como ***X*** e ***Y***. Objetos do primeiro tipo são chamados ***átomos***; objetos do segundo tipo são chamados ***variáveis***
- A ***aridade*** de uma relação é o seu número de argumentos e é denotada como uma barra seguida pela aridade
  - progenitor/2 significa que a relação progenitor possui 2 argumentos, ou que a relação progenitor tem aridade 2

Pontos Importantes

- Perguntas consistem em uma ou mais ***cláusulas***
  - Uma seqüência de cláusulas, tal como:

    progenitor(X,esaú), progenitor(X,jacó)
  - Significa a conjunção das cláusulas
    - X é um progenitor de Esaú **e**
    - X é um progenitor de Jacó
- A resposta a uma pergunta pode ser
  - Positiva: a pergunta é satisfatível e teve sucesso (*suceeded*)
  - Negativa: a pergunta é insatisfatível e falhou (*failed*)
- Se várias respostas satisfazem uma pergunta, Prolog encontra tantas quantas possíveis
  - Se o usuário estiver satisfeito com a resposta, basta digitar **return**
  - Se desejar mais respostas, usa-se ponto-e-vírgula "**;**"

Exercício

Expressar em português:

- valioso(ouro).
- femea(jane).
- possui(joao,ouro).
- pai(joao,maria).
- da(joao,livro,maria).

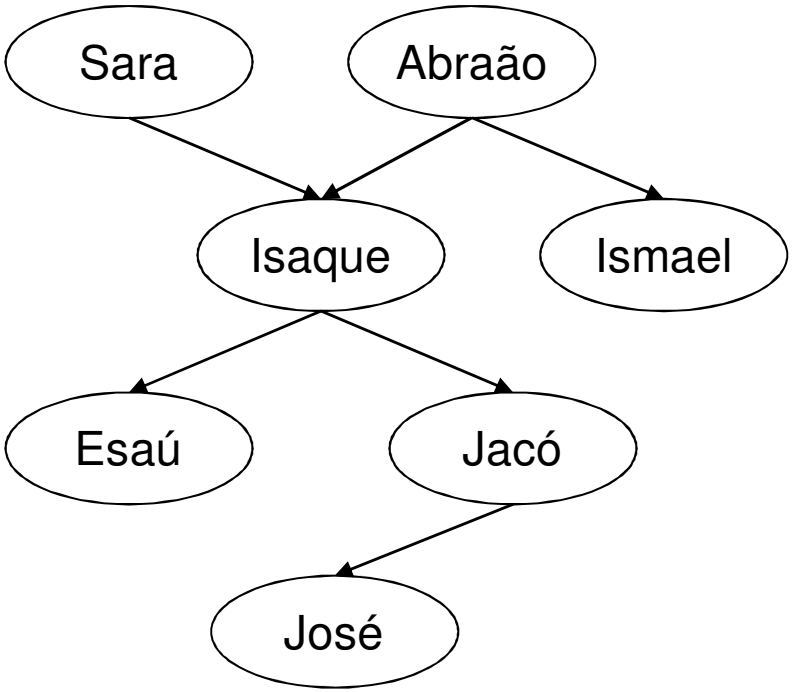
Exercício

```
gosta(joao, peixe).
gosta(joao,maria).
gosta(maria,livro).
gosta(pedro,livro).
gosta(maria,flor).
gosta(maria,vinho).
```

Quais as respostas dadas por Prolog?

```
?- gosta(maria,X).
?- gosta(X,livro).
?- gosta(Quem,Oque).
?- gosta(X,Y).
?- gosta(X,X).
?- gosta(_a,_b).
?- gosta(A,peixe).
```

Exercício



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

Quais as respostas dadas por Prolog?

(a) `?- progenitor(josé,X).`

(b) `?- progenitor(X,josé).`

(c)
```
?- progenitor(sara,X),
   progenitor(X,jacó).
```

(d)
```
?- progenitor(sara,X),
   progenitor(X,Y),
   progenitor(Y,josé).
```

Definindo Relações por Regras

- Nosso programa sobre famílias pode ser estendido de várias formas
- Vamos adicionar a informação sobre o sexo das pessoas envolvidas na relação ***progenitor***
- Por exemplo:
  - mulher(sara).
  - homem(abraão).
  - homem(isaque).
  - homem(ismael).
  - homem(esaú).
  - homem(jacó).
  - homem(josé).

- As relações ***mulher*** e ***homem*** são relações unárias
- Uma relação binária, como progenitor, define um relacionamento entre pares de objetos
- Relações unárias são usadas para declarar propriedades simples sim/não dos objetos
- A primeira cláusula unária pode ser lida como “Sara é uma mulher”

Definindo Relações por Regras

- Informação sobre o sexo das pessoas envolvidas na relação ***progenitor***:
  - mulher(sara).
  - homem(abraão).
  - homem(isaque).
  - homem(ismael).
  - homem(esaú).
  - homem(jacó).
  - homem(josé).
- Podemos declarar a mesma informação usando uma relação binária ***sexo***:
  - sexo(sara,feminino).
  - sexo(abraão,masculino).
  - sexo(isaque,masculino).
  - sexo(ismael,masculino).
  - sexo(esaú,masculino).
  - sexo(jacó,masculino).
  - sexo(josé,masculino).

Escolhendo Objetos e Relações

- Como representar: “Sara é uma mulher”
  - mulher(sara).
    - Permite responder: “Quem é mulher?”
    - Não permite responder: “Qual o sexo de Sara?”
  - sexo(sara,feminino).
    - Permite responder: “Quem é mulher?”
    - Permite responder: “Qual 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
    - Representação objeto-atributo-valor

Definindo Relações por Regras

- Vamos estender o programa introduzindo a relação ***filho_geral*** como o inverso da relação ***progenitor***
- Podemos definir ***filho_geral*** de maneira similar à relação ***progenitor***, ou seja enumerando uma lista de fatos sobre a relação ***filho_geral***, por exemplo
  - filho_geral(isaque,sara).
  - filho_geral(iisaque,abraão).
  - filho_geral(ismael,abraão).
  - ...

Definindo Relações por Regras

- Entretanto, a relação ***filho_geral*** pode ser definida de modo mais elegante:
  - Para todo X e Y,
    Y é um filho_geral de X se
    X é um progenitor de Y.
- Em Prolog:
  - filho_geral(Y,X) :-
    progenitor(X,Y).
- Esta cláusula também pode ser lida como:
  - Par todo X e Y,
    se X é um progenitor de Y então
    Y é um filho_geral de X

Definindo Relações por Regras

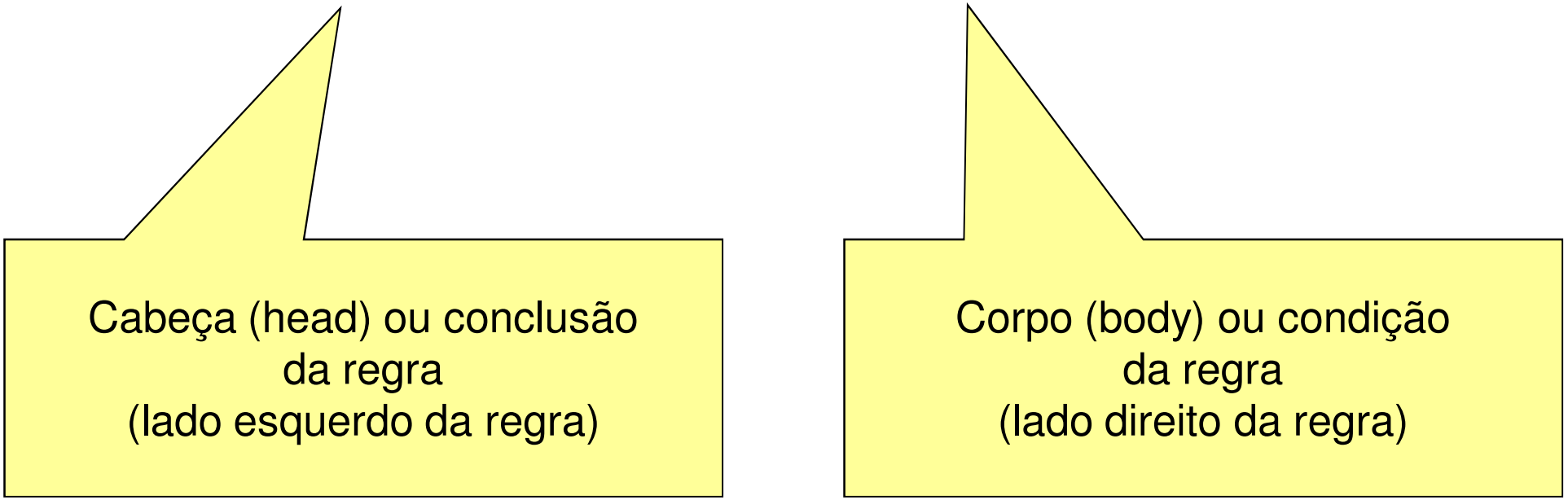
- Cláusulas Prolog como:
  - filho_geral(Y,X) :-
    progenitor(X,Y).

são chamadas ***regras*** (rules)

- 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

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



Definindo Relações por Regras

- Vamos perguntar se Ismael é filho_geral de Abraão:
  - `?- filho_geral(ismael,abraão).`
- Como não há fatos sobre a relação filho_geral, a única forma de Prolog responder esta pergunta é aplicando a regra sobre filho_geral
  - filho_geral(Y,X) :-
    progenitor(X,Y).
- A regra é geral no sentido que é aplicável a quaisquer objetos X e Y; portanto ela pode também ser aplicada a objetos particulares tais como **ismael** e **abraão**
- Para aplicar a regra a **ismael** e **abraão**, Y tem que ser substituído por **ismael** e X por **abraão**
- Neste caso, dizemos que as variáveis X e Y estão instanciadas a:
  - X = abraão e Y = ismael
- Depois da instanciação, obtemos um caso especial da regra geral, que é:
  - filho_geral(ismael,abraão) :-
    progenitor(abraão,ismael).

Definindo Relações por Regras

- A condição da regra com as variáveis instanciadas
  - filho_geral(ismael,abraão) :-
    progenitor(abraão,ismael).
- é:
  - progenitor(abraão,ismael).
- Assim, Prolog tenta provar que a condição é verdadeira
- Para provar a condição, é trivial por Prolog encontra um fato correspondente no programa
- Isso implica que a conclusão da regra também é verdadeira e Prolog responde afirmativamente à pergunta:
  - `?- filho_geral(ismael,abraão).`
  - `yes`

Definindo Relações por Regras

- Vamos 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:
  - mãe(X,Y) :-
    progenitor(X,Y),
    mulher(X).
- Uma vírgula entre duas condições indica a conjunção das condições, significando que ambas condições têm que ser verdadeiras

Definindo Relações por Regras

- Exemplo: de Português para Prolog
  - Uma pessoa é mãe se tiver algum filho e essa pessoa for mulher
  - Uma pessoa é mãe se for progenitor de alguém e essa pessoa for mulher
  - Uma pessoa X é mãe de Y se X for progenitor de Y e X for mulher
  - X é mãe de Y se X é progenitor de Y e X é mulher
  - ```
    mãe(X,Y) :-
       progenitor(X,Y),
       mulher(X).
    ```
    ```

Definindo Relações por Regras

- Em Prolog, uma regra consiste de uma cabeça e uma corpo
- A cabeça e o corpo são conectados pelo símbolo **:-**, denominado *neck*, formado por dois pontos e hífen
- **:-** é pronunciado “se”

mãe(X,Y) **:-** progenitor(X,Y), mulher(X).

Exercício

- Identifique a cabeça e cauda de cada regra.
- Expressar cada regra em Português:

```
gosta(joao,X) :-
    gosta(X,vinho),
    gosta(X,comida).
gosta(joao,X) :-
    mulher(X),
    gosta(X,vinho).
```

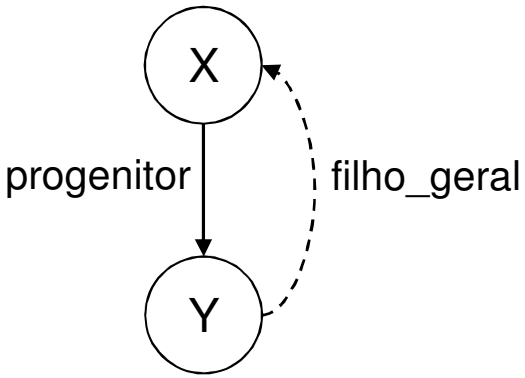
Exercício

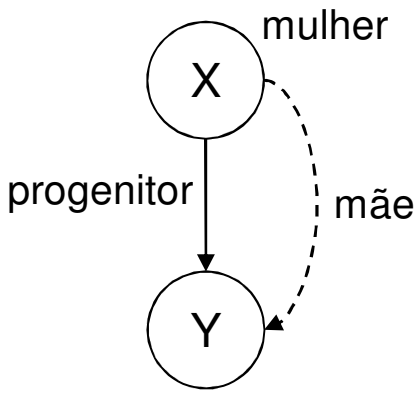
- Usando a base ao lado, defina a regra: Uma pessoa pode roubar algo se essa pessoa é um ladrão e ela gosta de um objeto
- Qual a resposta dada por Prolog a pergunta: João rouba o quê?

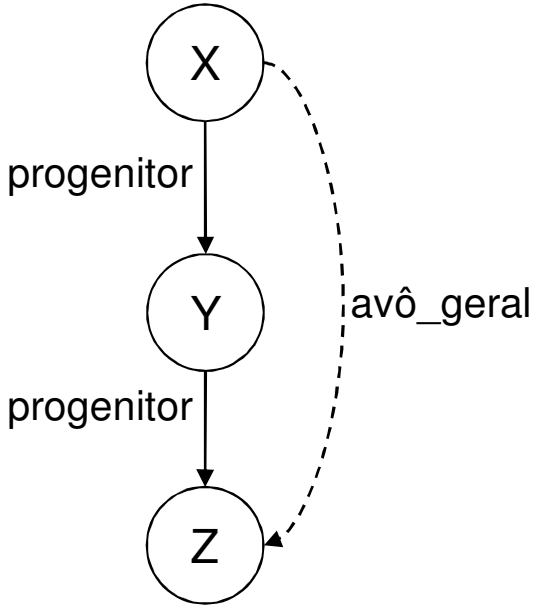
```
ladrao(joao).
ladrao(pedro).
gosta(maria,flor).
gosta(maria,queijo).
gosta(maria,vinho).
gosta(joao,rubi).
gosta(joao,X) :-
  gosta(X,vinho).
```

Grafos Definindo Relações

- Relações como ***progenitor***, ***filho_geral*** e ***mãe*** podem ser ilustradas por diagramas que seguem as seguintes convenções
  - Nós nos grafos correspondem a objetos (argumentos das relações)
  - Arcos entre nós correspondem a relações binárias (2 argumentos)
  - Arcos são orientados apontando do primeiro argumento da relação para o segundo argumento
  - Relações unárias são indicadas nos diagramas simplesmente marcando os objetos correspondentes com o nome da relação
  - As relações sendo definidas são representadas por arcos tracejados

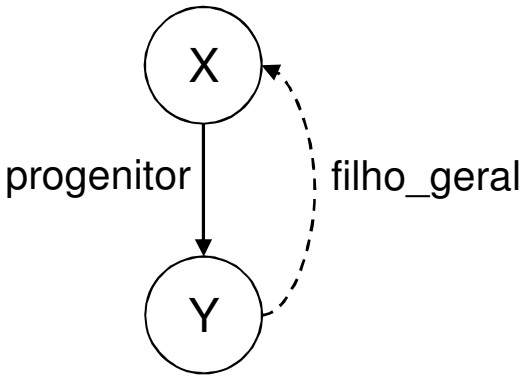


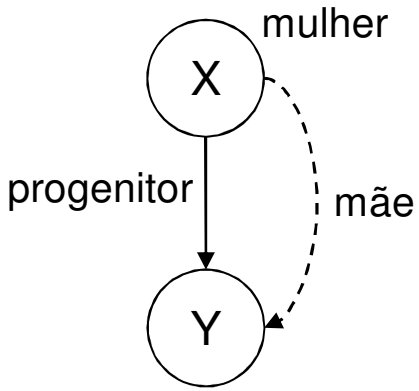


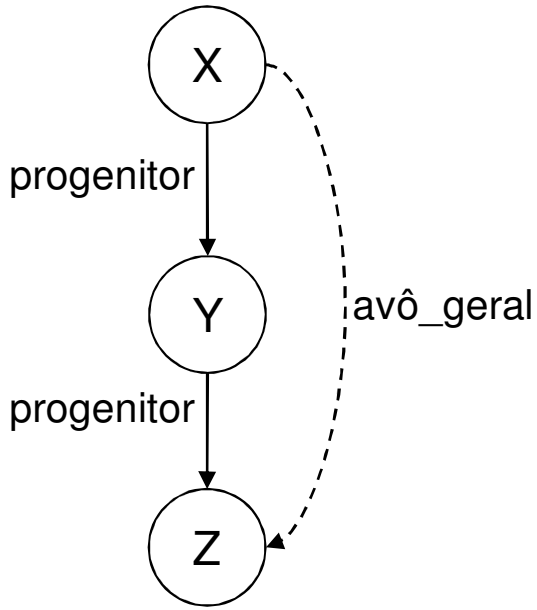


Grafos Definindo Relações

- Cada diagrama deve ser interpretado da seguinte forma: se as relações mostradas pelos arcos sólidos são verdadeiras então a relação mostrada pelo arco tracejado também é verdadeira
- Assim, a relação ***avô_geral*** pode ser imediatamente escrita como:
  - avô_geral(X,Z) :- progenitor(X,Y), progenitor(Y,Z).







Layout de um Programa Prolog

- Prolog fornece liberdade na escrita do *layout* do programa
- Entretanto, os programas devem ter um aspecto compacto e, acima de tudo, fácil de ler
- Assim, é um padrão escrever a cabeça de uma cláusula bem como cada condição em seu corpo em uma linha separada
- Além disso, as condições são deslocadas de modo a melhor separar a cabeça do corpo de uma regra

Layout de um Programa Prolog

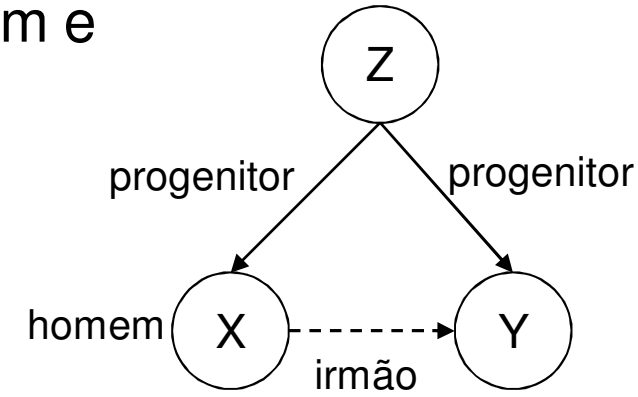
- Por exemplo, a relação
  - avô_geral(X,Z) :- progenitor(X,Y), progenitor(Y,Z).

deve ser escrita da seguinte forma:

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

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 e
    X é um homem.
- Em Prolog:
  - irmão(X,Y) :-
    progenitor(Z,X),
    progenitor(Z,Y),
    homem(X).



- Note a forma de expressar "ambos X e Y têm um progenitor em comum":
  - Algum Z deve ser o progenitor de X e este mesmo Z deve ser o progenitor de Y
- Uma forma alternativa, mas menos elegante seria: Z1 é progenitor de X e Z2 é progenitor de Y e Z1 é igual a Z2

Definindo Relações por Regras

- Podemos perguntar a Prolog:
  - `?- irmão(esaú,jacó).`
  - `yes`
- Portanto, poderíamos concluir que a relação ***irmão***, como definida, funciona corretamente
- Entretanto há uma falha em nosso programa que é revelada se perguntamos "Quem é o irmão de Jacó?"
  - `?- irmão(X,jacó).`
- Prolog fornecerá duas respostas
  - `X = esaú ;`
  - `X = jacó`
- Assim, Jacó é irmão dele mesmo? Provavelmente isso não era bem o que tínhamos em mente quando definimos a relação ***irmão***
- Entretanto, de acordo com nossa definição sobre irmãos, a resposta de Prolog é perfeitamente lógica

Definindo Relações por Regras

- Nossa regra sobre irmãos não menciona que X e Y não devem ser a mesma pessoa se X deve ser irmão de Y
- Como isso não foi definido, Prolog (corretamente) assume que X e Y podem ser a mesma pessoa e como conseqüência encontra que todo homem que tem um progenitor é irmão de si próprio
- Para corrigir a regra sobre irmãos, devemos adicionar a restrição que X e Y devem ser diferentes
- Veremos nas próximas aulas como isso pode ser efetuado de diversas maneiras, mas para o momento, vamos assumir que a relação ***diferente*** já é conhecida de Prolog e que diferente(X,Y) é satisfeita se e somente se X e Y não são iguais
- Isso nos leva à seguinte regra sobre irmãos:
  - irmão(X,Y) :-
    progenitor(Z,X),
    progenitor(Z,Y),
    homem(X),
    diferente(X,Y).

Pontos Importantes

- 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
- Cláusulas Prolog consistem em uma cabeça e o corpo; o corpo é uma lista de condições separadas por vírgulas (que significam conjunções)
- Fatos são cláusulas que têm uma cabeça e o corpo vazio; perguntas têm apenas o corpo; regras têm uma cabeça e um corpo (não vazio)

Pontos Importantes

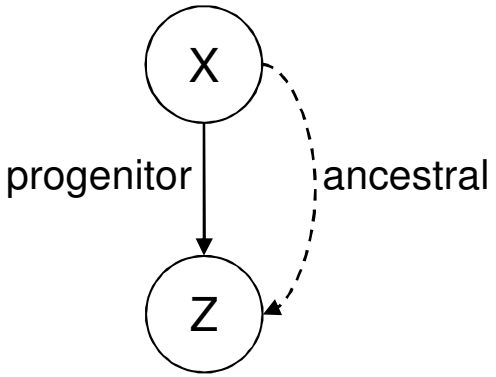
- Durante a computação, uma variável pode ser substituída por um objeto: dizemos que a variável está *instanciada*
- As variáveis são universalmente quantificadas e são lidas como “Para todo”
- Todavia, leituras alternativas são possíveis para variáveis que aparecem apenas no corpo
- Por exemplo:
  - temfilhos(X) :- progenitor(X,Y).
- Pode ser lida de duas formas:
  - *Para todo* X e Y,
    se X é um progenitor de Y então
    X tem filhos
  - *Para todo* X,
    X tem filhos se
    existe algum Y tal que X é um progenitor de Y

Exercícios

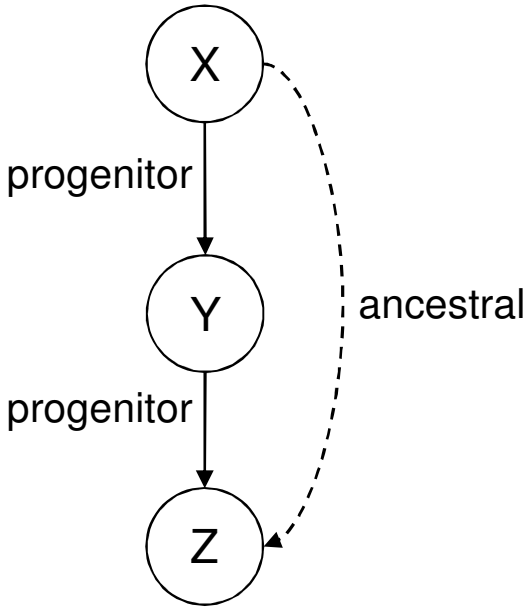
- Traduza para Prolog: Todo mundo que tem filho é feliz (defina a relação unária ***feliz***)
- Defina as relações ***irmã*** e ***irmão_geral***
- Defina a relação ***neto_geral*** usando a relação progenitor
- Defina a relação ***tio(X,Y)*** em termos das relações ***progenitor*** e ***irmão***

Regras Recursivas

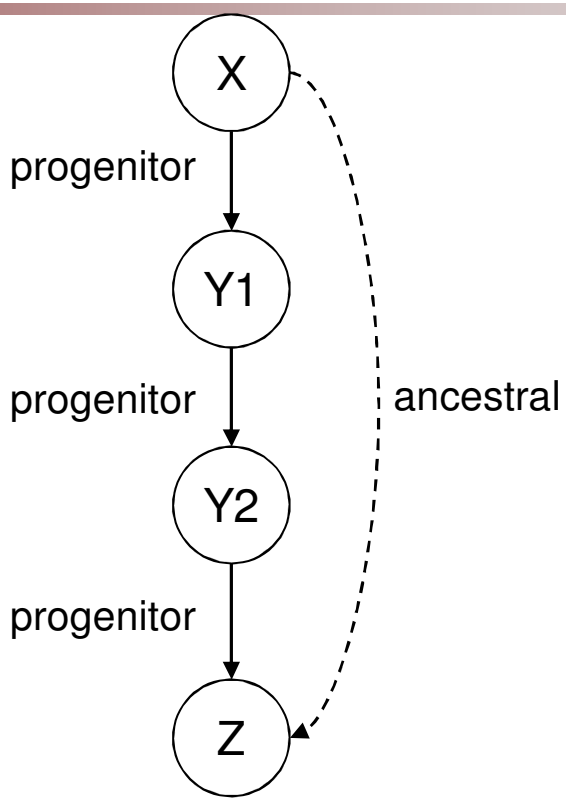
- Vamos adicionar a relação ***ancestral***
- Esta relação será definida por duas regras: a primeira será o caso base (não recursivo) e a segunda será o caso recursivo
  - Para todo X e Z,
    X é um ancestral de Z se
    X é um progenitor de Z.
  - ancestral(X,Z) :-
    progenitor(X,Z).



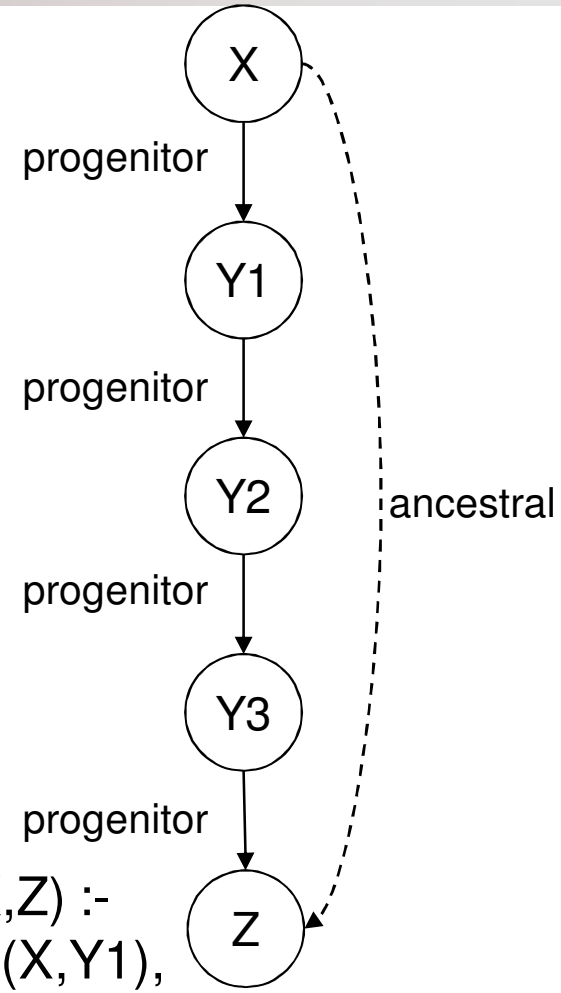
Regras Recursivas




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



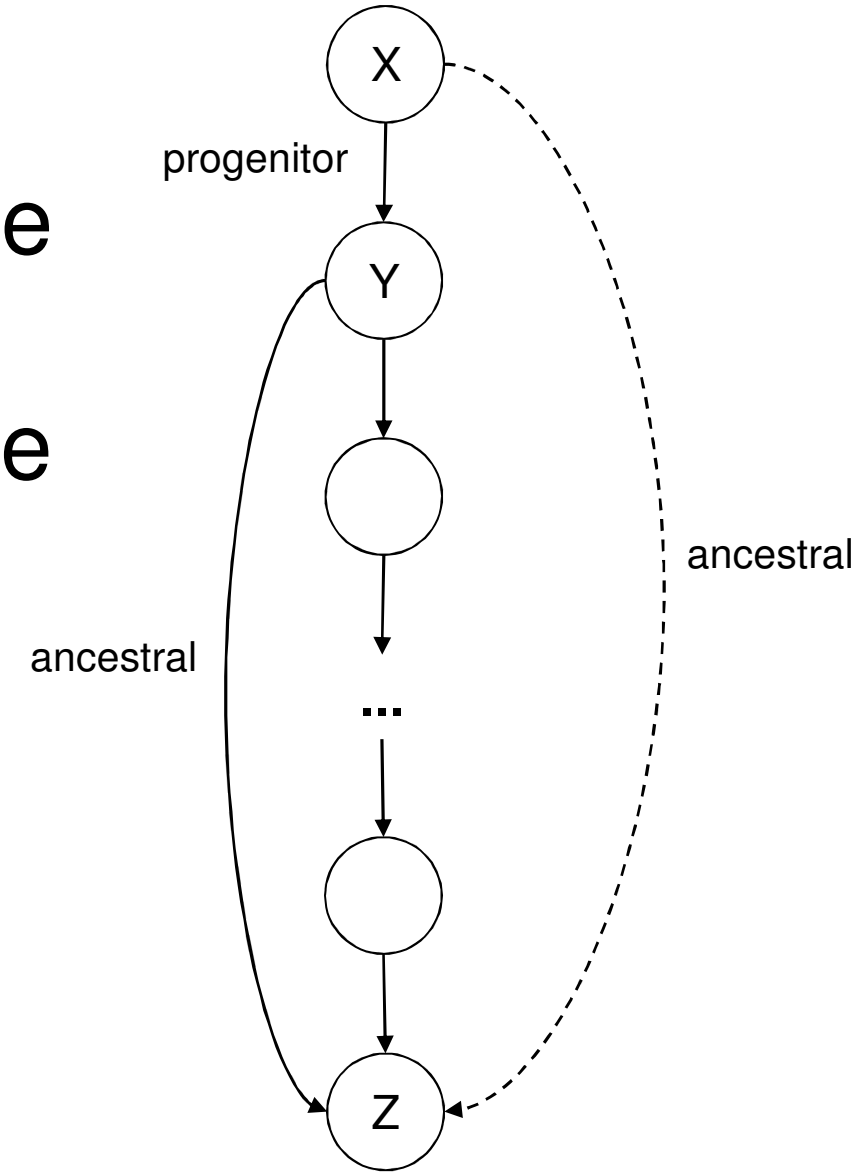

```
ancestral(X,Z) :-
  progenitor(X,Y1),
  progenitor(Y1,Y2),
  progenitor(Y2,Z).
```




```
ancestral(X,Z) :-
  progenitor(X,Y1),
  progenitor(Y1,Y2),
  progenitor(Y2,Y3),
  progenitor(Y3,Z).
```

Regras Recursivas

- Para todo X e Z,
  X é um ancestral de Z se
  há algum Y tal que
  X é um progenitor de Y e
  Y é um ancestral de Z.
- ancestral(X,Z) :-
  progenitor(X,Y),
  ancestral(Y,Z).



Regras Recursivas

- ancestral(X,Z) :- progenitor(X,Z). % caso base
- ancestral(X,Z) :- progenitor(X,Y), ancestral(Y,Z). % caso recursivo
- Podemos perguntar: quais os descendentes de Sara?
  - `?- ancestral(sara,X).`
  - **`X = isaque;`**
  - **`X = esaú;`**
  - **`X = jacó;`**
  - **`X = josé`**

Programa da Família Bíblica

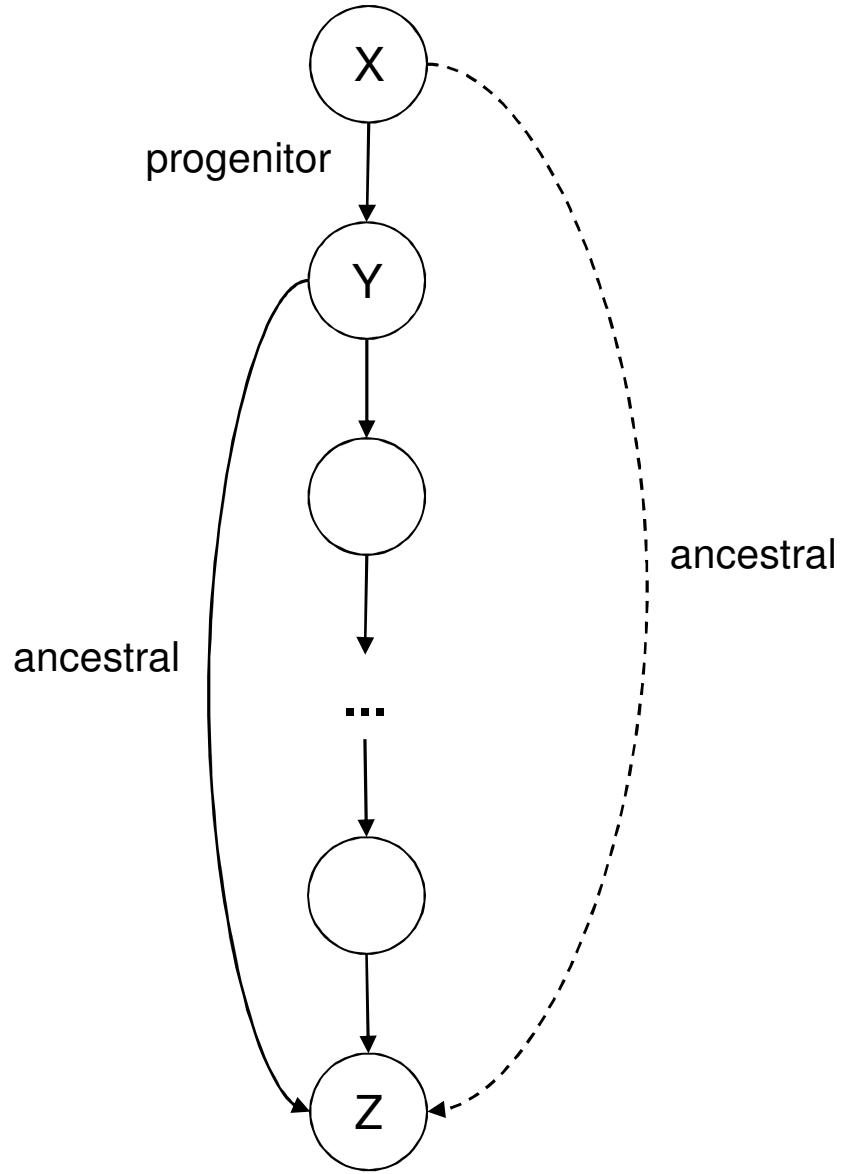
```
progenitor(sara,isaque).
progenitor(abraão,isaque).
progenitor(abraão,ismael).
progenitor(isaque,esaú).
progenitor(isaque,jacó).
progenitor(jacó,josé).

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

```
filho_geral(Y,X) :-
  progenitor(X,Y).
mãe(X,Y) :-
  progenitor(X,Y),
  mulher(X).
avô_geral(X,Z) :-
  progenitor(X,Y),
  progenitor(Y,Z).
irmão(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).
```

Exercício

- Vimos a seguinte definição da relação ancestral
  - ancestral(X,Z) :- progenitor(X,Z).
  - ancestral(X,Z) :- progenitor(X,Y), ancestral(Y,Z).
- Considere a seguinte definição alternativa:
  - ancestral(X,Z) :- progenitor(X,Z).
  - ancestral(X,Z) :- progenitor(Y,Z), ancestral(X,Y).
- Esta é uma definição correta de ancestrais?
- É possível modificar o diagrama de forma a corresponder a esta nova definição?



Slides baseados nos livros:

Bratko, I.;
Prolog Programming for Artificial Intelligence,
3rd Edition, Pearson Education, 2001.

Clocksin, W.F.; Mellish, C.S.;
Programming in Prolog,
5th Edition, Springer-Verlag, 2003.

Material baseado em slides de
José Augusto Baranauskas
USP-Ribeirão Preto