Sistemas Baseados em Conhecimento

Inteligência Artificial

Prof. Leandro C. Fernanfdes

O início das coisas ...

- “A inteligência requer Conhecimento”

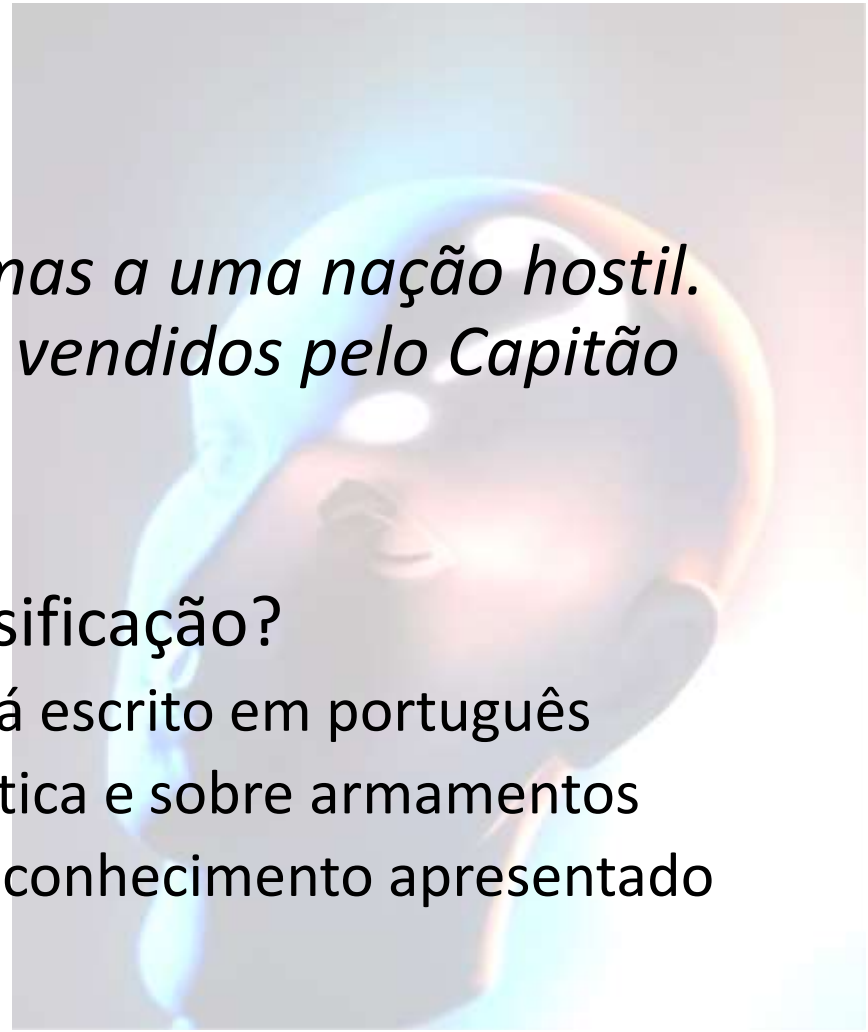
- Características do conhecimento humano
  - Volumoso
  - Impreciso
  - Dinâmico
  - Organizado por conteúdo

- Um sistema artificial deve ter:
  - Capacidade de generalização
  - Compreensão pelas pessoas que o fornecem
  - Facilmente modificado
  - Vastamente utilizado (impreciso)

Reflita sobre esse caso

- Problema: West é criminoso ou não?

“A lei americana diz que é proibido vender armas a uma nação hostil. Cuba possui alguns mísseis e todos eles foram vendidos pelo Capitão West, que é americano”



- Como você resolveria este problema de classificação?
  - **Linguagem**: você é capaz de entender o que está escrito em português
  - **Conhecimento**: você sabe um pouco de geopolítica e sobre armamentos
  - **Inferência**: você é capaz de raciocinar usando o conhecimento apresentado

O que possivelmente se passou na sua cabeça

Conhecimento prévio

A) Todo americano que vende uma arma a uma nação hostil é criminoso
B) Todo país em guerra com uma nação X é hostil a X
C) Todo país inimigo político de uma nação X é hostil a X
D) Todo míssil é um arma
E) Toda bomba é um arma
F) Cuba é uma nação
G) USA é uma nação
H) Cuba é inimigo político dos USA
I) Irã é inimigo político dos USA

Novo Conhecimento

M) Cuba possui um míssil M1	- *a partir de* K
N) M1 é um míssil	- *a partir de* K
O) M1 é uma arma	- *a partir de* D e N
P) Cuba é hostil aos USA	- *a partir de* F, G, H e C
Q) M1 foi vendido a Cuba por West	- *a partir de* L, M e N
R) West é criminoso	- *a partir de* A, J, O, P e Q

Conhecimento do problema

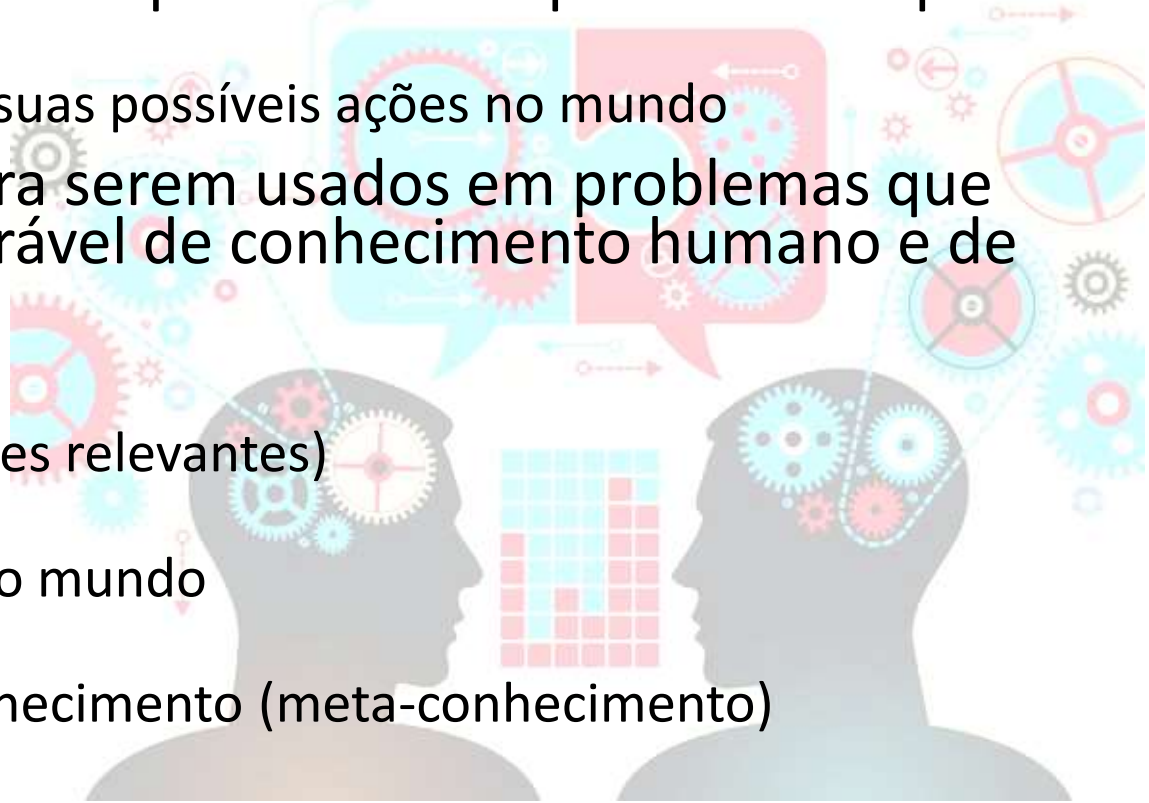
J) West é americano
K) Existem mísseis em cuba
L) Os mísseis de cuba foram vendidos por West

Como uma máquina solucionaria o problema?

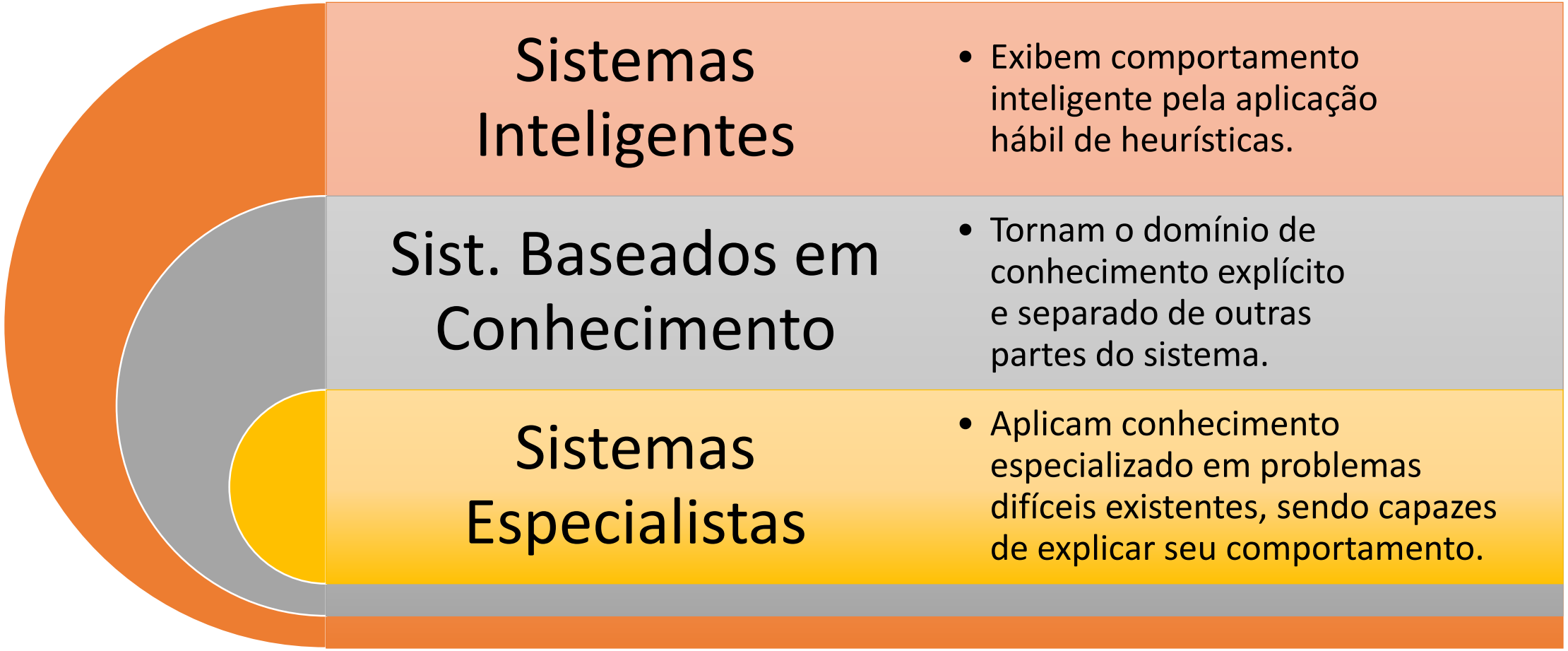
- Segundo a IA...
  - Identificar o **conhecimento** do domínio
  - Representá-lo em uma **linguagem** formal
  - Implementar um mecanismo de **inferência** para utilizá-lo
- The Knowledge Principle (Lenat & Feigenbaum)
  - *Se um programa é capaz de realizar bem uma tarefa complexa, ele deve conhecer bastante sobre o mundo no qual opera.*
- Questões-chave:
  - Como adquirir esse conhecimento?
  - Como representá-lo adequadamente?
  - Como raciocinar com ele correta e eficientemente?

Sistemas Baseados em Conhecimento

- Um Sistema Baseado em Conhecimento (SBC) é um programa de computador que utiliza conhecimento representado explicitamente para resolver problemas.
  - São sistemas que “raciocinam” sobre suas possíveis ações no mundo
- Ou seja, SBCs são desenvolvidos para serem usados em problemas que requerem uma quantidade considerável de conhecimento humano e de perícia para serem resolvidos
- Conhecem:
  - O estado atual do mundo (propriedades relevantes)
  - Como o mundo evolui
  - Como identificar estados desejáveis do mundo
  - Como avaliar o resultado das ações
  - Apresentam conhecimento sobre conhecimento (meta-conhecimento)



IA, SBCs e SEs


Sistemas Inteligentes
• Exibem comportamento inteligente pela aplicação hábil de heurísticas.
Sist. Baseados em Conhecimento
• Tornam o domínio de conhecimento explícito e separado de outras partes do sistema.
Sistemas Especialistas
• Aplicam conhecimento especializado em problemas difíceis existentes, sendo capazes de explicar seu comportamento.

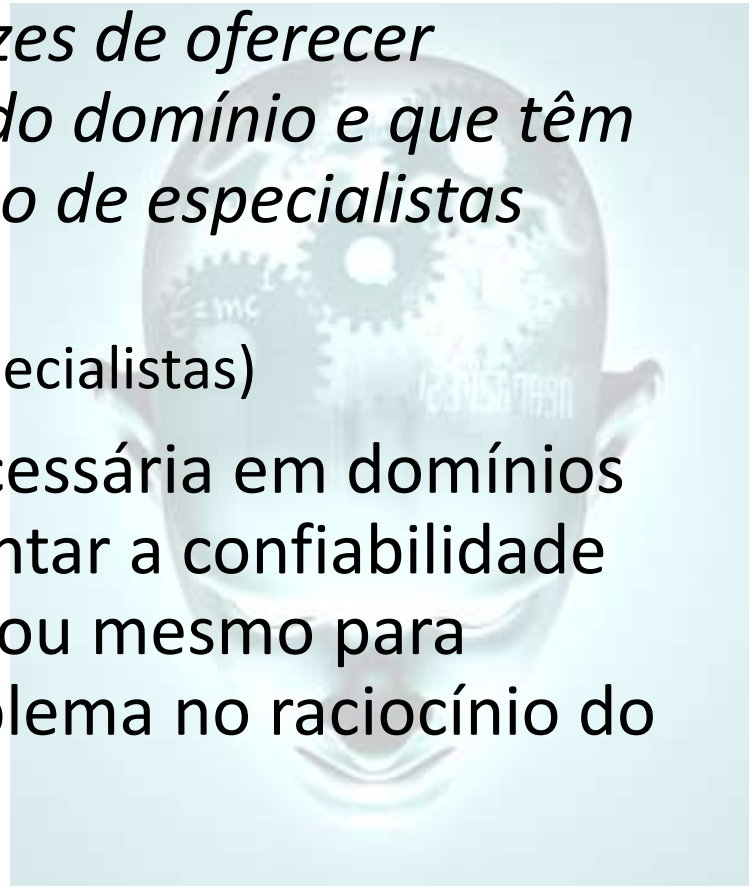
Sistemas Baseados em Conhecimento

- Para fazer com que um Sistema Baseado em Conhecimento chegue perto do desempenho de um especialista humano, o sistema deve:
  - ter grande quantidade de conhecimento disponível
  - conseguir ter acesso a este conhecimento rapidamente
  - ser capaz de raciocinar adequadamente com este conhecimento
  - um SE, adicionalmente, devem possuir uma capacidade amigável de interação usuário-computador que torna o raciocínio do sistema transparente ao usuário
- Os SBCs diferem-se dos sistemas convencionais na forma de incorporar o conhecimento

Sistemas Especialistas

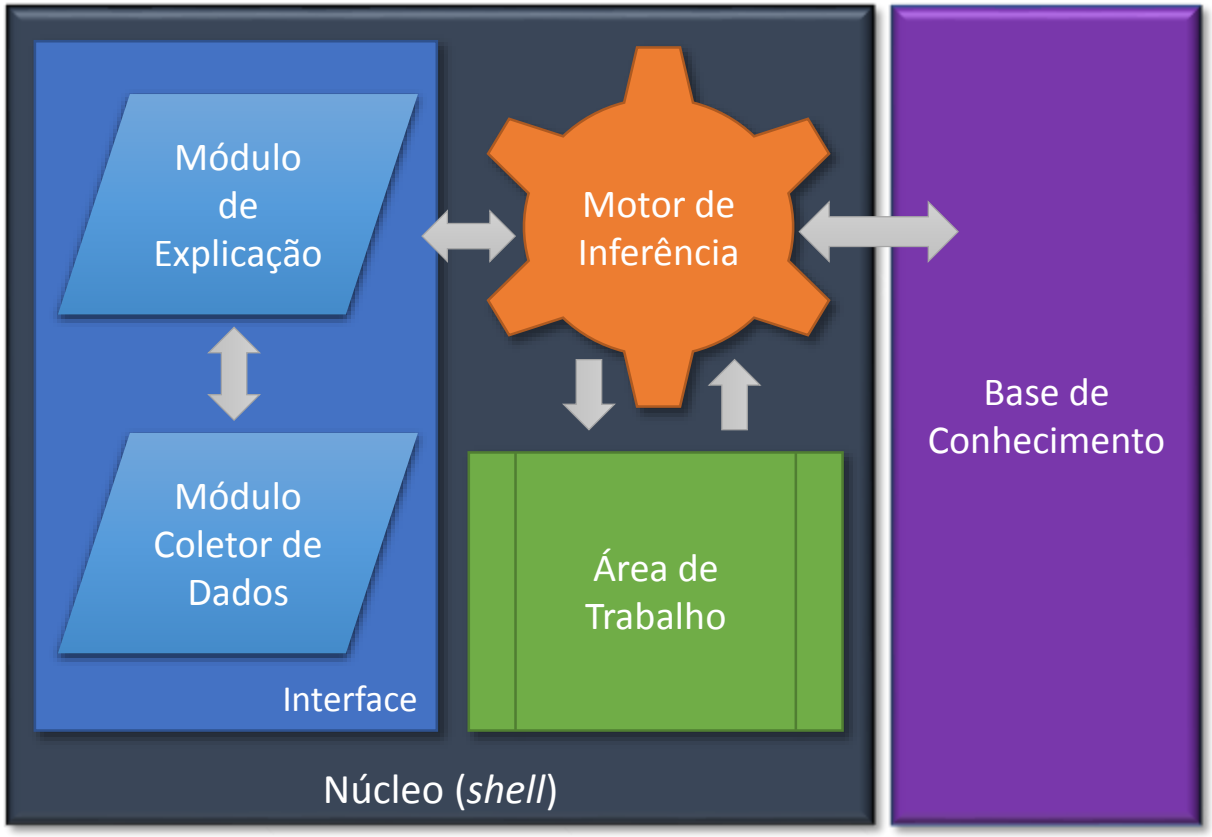
Sistemas Especialistas

- *“Sistemas Especialistas (SEs) são sistemas capazes de oferecer soluções para problemas específicos em um dado domínio e que têm habilidade de aconselhar no nível comparável ao de especialistas naquela área”*
  - (Lucas and van der Gaag, Princípios de Sistemas Especialistas)
- A habilidade de explicação é especialmente necessária em domínios incertos (como diagnóstico médico) para aumentar a confiabilidade do usuário no conselho fornecido pelo sistema ou mesmo para permitir o usuário detectar algum possível problema no raciocínio do sistema



Arquitetura

- Base de Conhecimento (BC)
- Área de Trabalho (AT)
- Motor de Inferência (MI)
- Interface com usuário
  - Módulo Coletor de Dados (MCD)
  - Módulo de Explicação (ME)



Arquitetura

- Interface:
  - É um processador de linguagem projetado para processar e produzir comunicação orientada a problemas entre o usuário e o sistema
    - Usualmente ocorre utilizando linguagem natural, sendo complementada por menus e elementos gráficos
- Módulo Coletor de Dados
  - Acionado pelo MI quando este necessita dados específicos
  - Pergunta ao usuário e obtém as respostas, enviando-as ao MI

Arquitetura

- Módulo de Explicação
  - Módulo que facilita a explicação, justificando as conclusões e explicando o comportamento do sistema
  - Isto é feito por meio de questões interativas:
    - Porque o sistema faz uma pergunta em particular?
    - Como o sistema alcança a conclusão correta?
    - Porque uma certa alternativa é rejeitada?
    - Qual é a tática atual do sistema para alcançar a conclusão?

Arquitetura

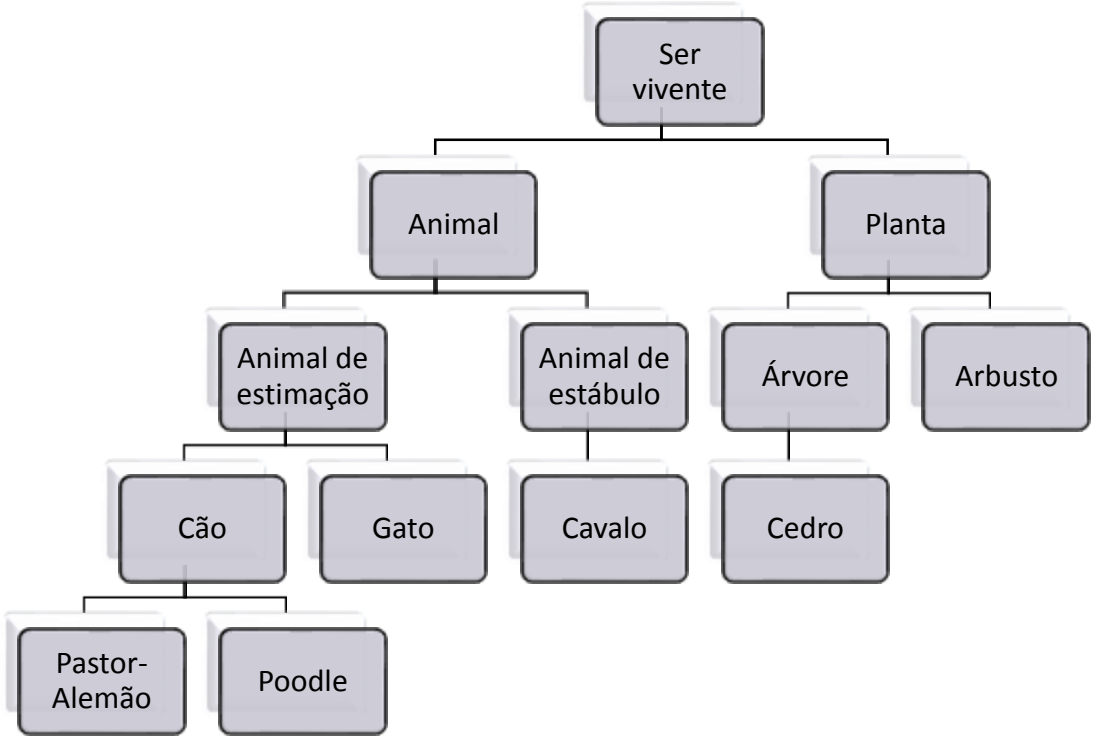
- Base de Conhecimento
  - Contém informações necessárias, no nível de um especialista, para solucionar problemas em um domínio específico.
- Área de Trabalho
  - Armazena fatos deduzidos a respeito do problema corrente.
  - Atualizada sempre que novas informações tornam-se disponíveis.
  - Conteúdo geralmente descartado após execução.
- Motor de Inferência
  - Responsável em aplicar as estratégias de inferência e controle
  - Usa algum tipo de raciocínio
  - Processa informações contidas na BC e AT, tentando encontrar uma solução para o problema no qual está trabalhando

Representação do Conhecimento

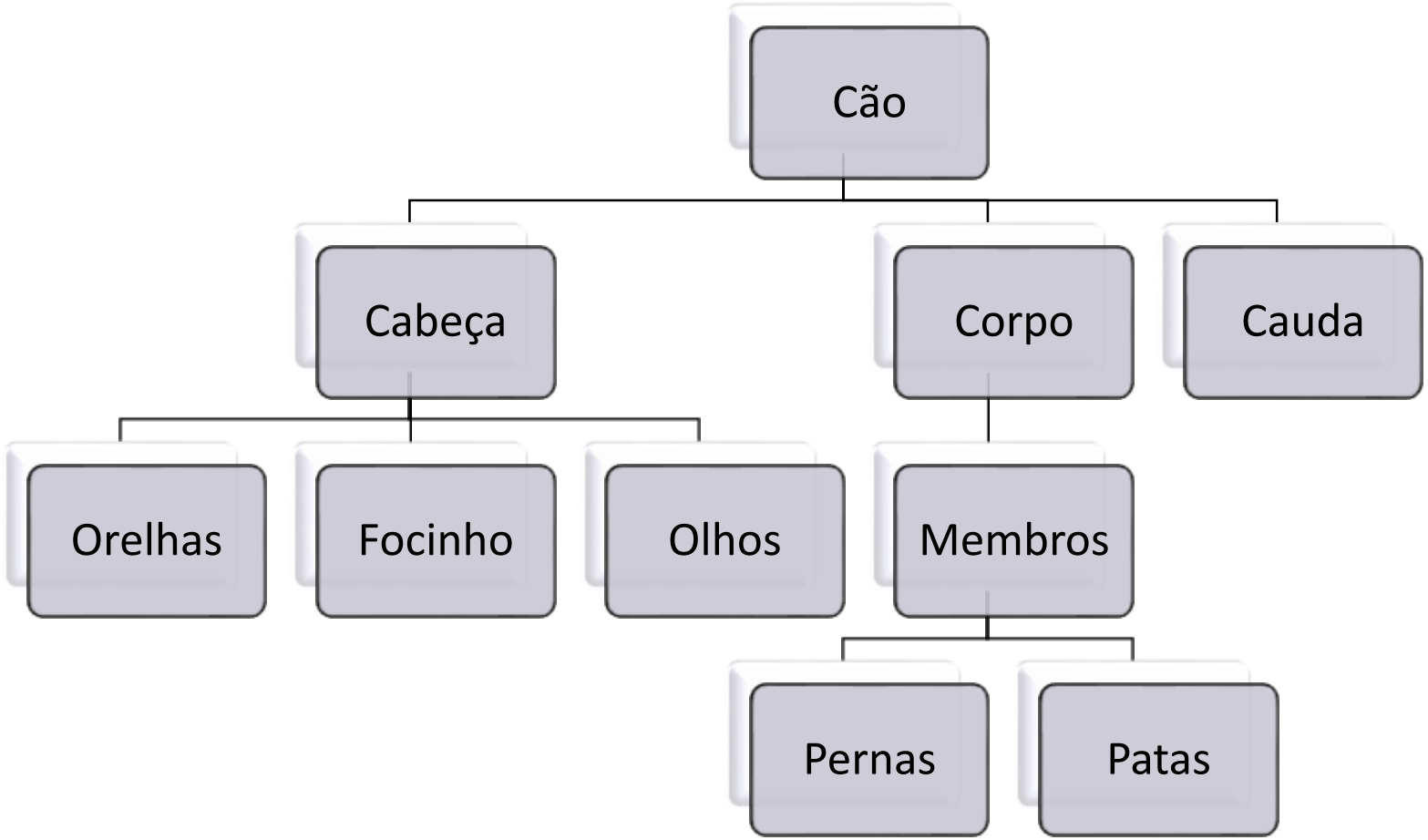
- Regras *if-then*
  - *if* condição P *then* conclusão C
  - *if* situação S *then* ação A
  - *if* condições C1 e C2 são verdadeiras *then* condição C não é verdadeira
- Lógica de predicados
  - numero_pernas(humano,2).
  - homem(bob).
  - gosta(X,Y) :- inteligente(Y).
- Redes semânticas
  - Representação por relações entre objetos
  - Relações mais comuns
    - is-a (é-um)
    - ako (a-kind-of) ou um-tipo-de ou faz-parte
- Frames

Redes Semânticas: Hierarquia **é-um** (is-a)

- pastor-alemão *é-um* cão
- poodle *é-um* cão
- cavalo *é-um* animal-estábulo
- cão *é-um* animal-estimação
- animal-estimação *é-um* animal
- animal-estábulo *é-um* animal
- animal *é-um* ser-vivente
- planta *é-um* ser-vivente
- árvore *é-uma* planta
- arbusto *é-uma* planta



Redes Semânticas: Hierarquia **faz-parte** (*ako*)

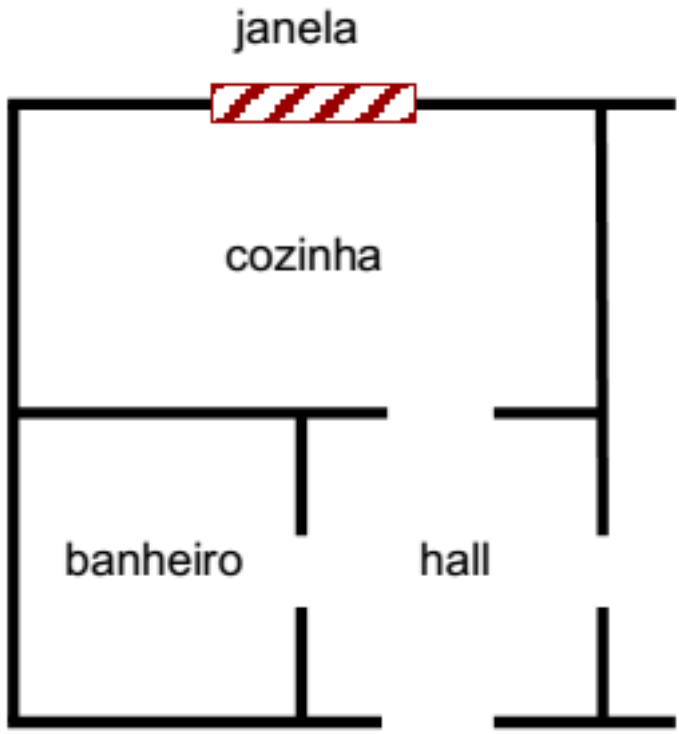


Linhas de Raciocínio

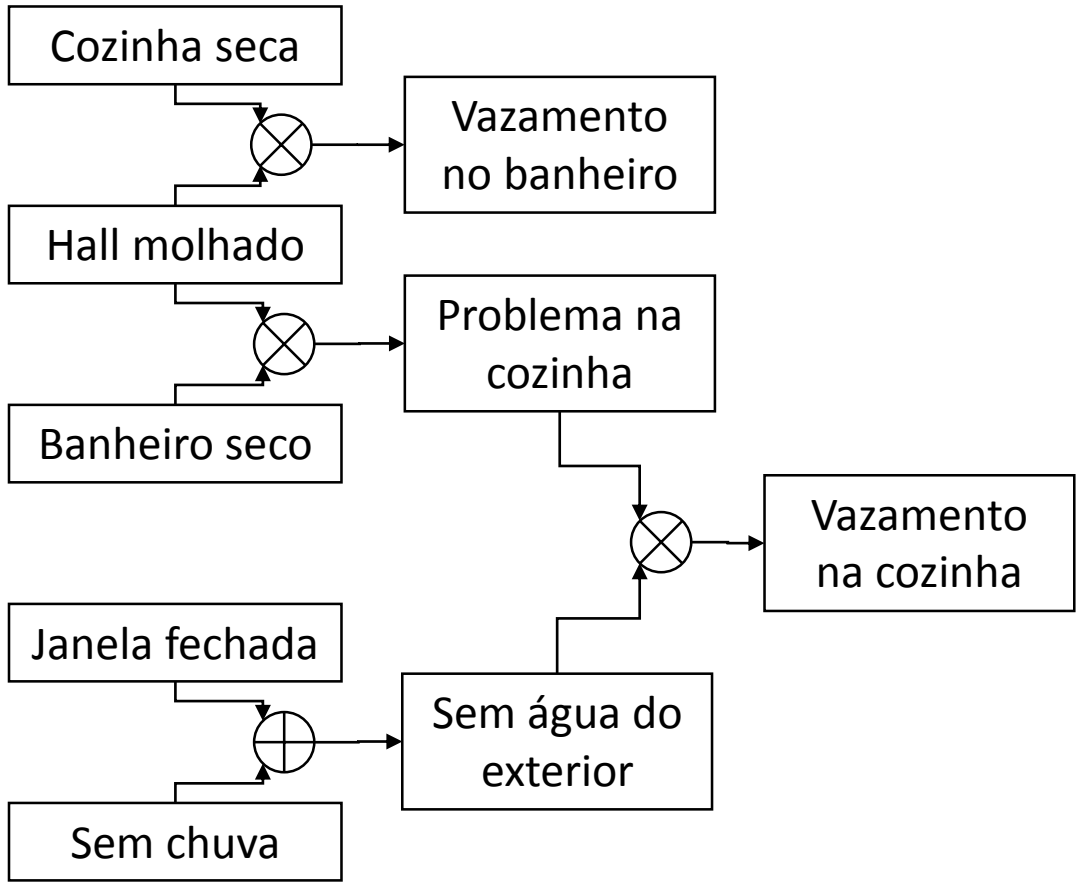
- Uma vez que o conhecimento está representando de alguma forma, é necessário escolher um procedimento para tirar conclusões a partir da Base de Conhecimento (BC)

- Utilizando regras *if-then* há duas formas
  - *Backward Chaining* ou Encadeamento Regressivo
  - *Forward Chaining* ou Encadeamento Progressivo

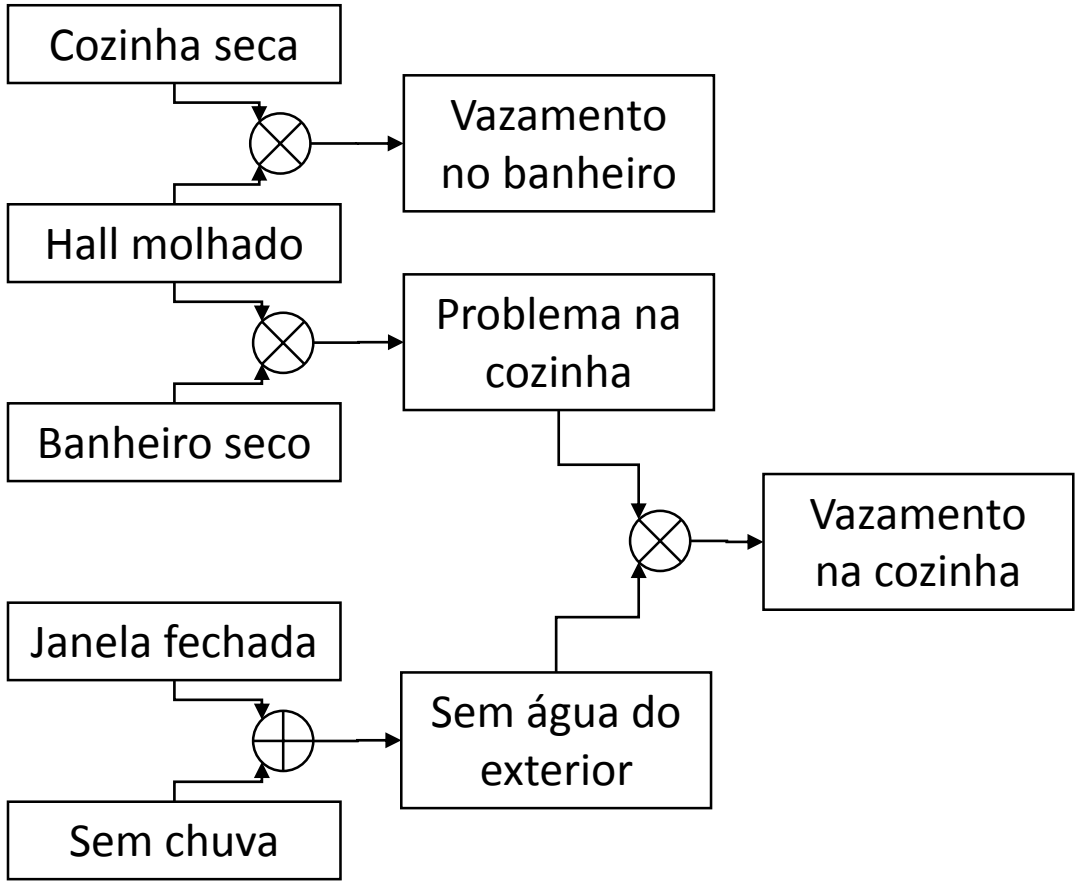
Exemplo: Detectar vazamento



if hall_molhado **and** banheiro_seco ***then***
problema_na_cozinha



Raciocínio: *Backward Chaining*



- Iniciar com uma hipótese
  - por exemplo, *vazamento na cozinha*
- Então raciocinamos para trás na rede de inferência
  - para confirmar a hipótese precisamos que *problema na cozinha* e *sem água do exterior* sejam verdadeiros
    - *Problema na cozinha* pode ser confirmado se encontramos que o hall está molhado e o banheiro seco
    - *Sem água do exterior* pode ser confirmado, por exemplo, se encontramos que a janela está fechada

Implementando a solução em Prolog!

- Base de Conhecimento:

```
vazamento_no_banheiro :-
     hall_molhado,
     cozinha_seca.
problema_na_cozinha :-
     hall_molhado,
     banheiro_seco.
sem_água_do_exterior :-
     janela_fechada ;
     sem_chuva.
vazamento_na_cozinha :-
     problema_na_cozinha,
     sem_água_do_exterior.
```

- As evidências encontradas pelo mecanismos de inferência:

```
hall_molhado.
banheiro_seco.
janela_fechada.
```

- A hipótese pode ser verificada por:

```
?- vazamento_na_cozinha.
yes
```

Então, já temos um Sistema Especialista? Considerações e ajustes

- A utilização da sintaxe típica em Prolog tem algumas desvantagens
  - A sintaxe pode não ser adequada se considerarmos um usuário que não conhece bem Prolog;
  - O especialista do domínio deve ser capaz de: ler as regras, alterá-las ou especificar novas.
  - A BC não é sintaticamente distinguível do resto do programa.
    - A distinção explícita entre a BC e o restante do programa Prolog é desejável
- Usando a notação de operadores em Prolog, podemos especificar os operadores '*if*', '*then*', '*and*' e '*or*' de tal forma que seja possível escrever

```
if hall_molhado and banheiro_seco then problema_na_cozinha.
```

  ao invés de

```
problema_na_cozinha :- hall_molhado, banheiro_seco.
```

- Além disso, as evidências observadas podem ser informadas pela relação fato/1.

```
fato(hall_molhado).
```

Então, já temos um Sistema Especialista? Considerações e ajustes

- Essas alterações significam que agora precisamos de um novo interpretador para as regras nessa nova sintaxe.
- O interpretador será definido como a relação **e_verdade(P)** onde**:**
  - P é um fato fornecido; ou
  - P pode ser derivado utilizado as regras.

- O interpretador pode ser chamado agora da forma:

```
e_verdade(vazamento_na_cozinha).
```

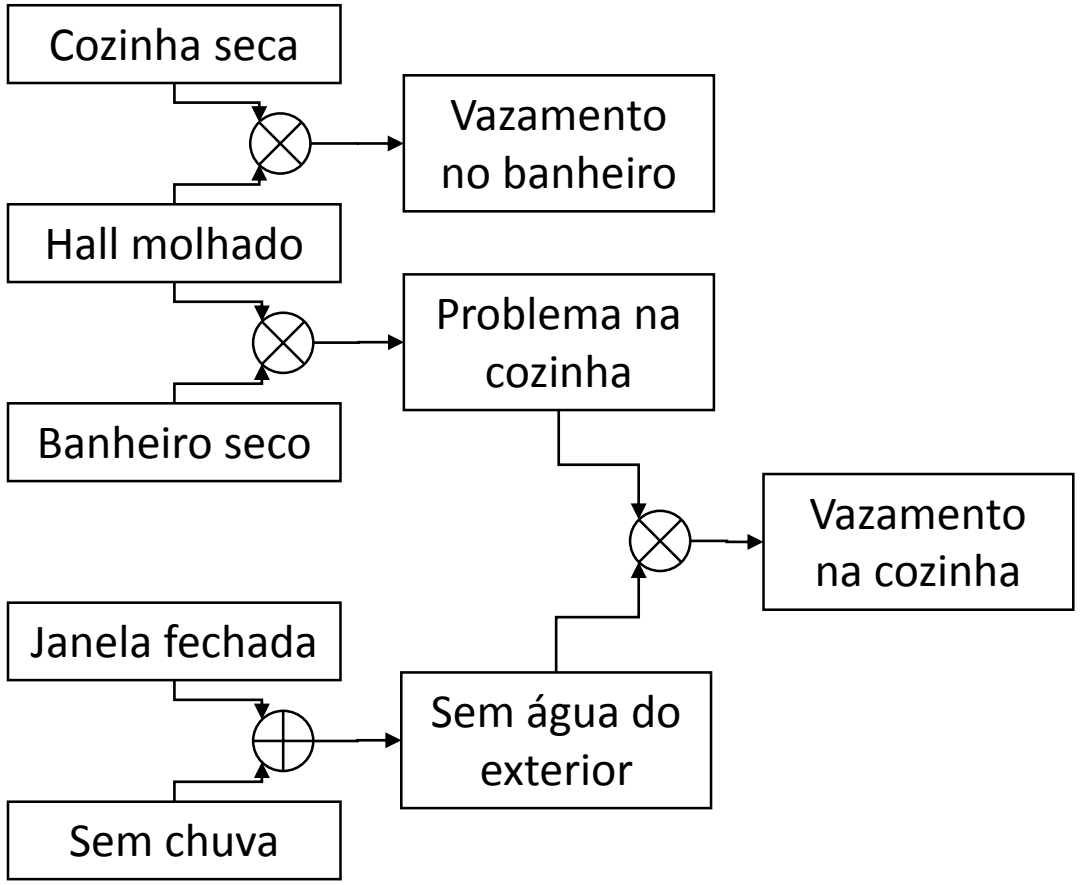
Implementação do Sist.Especialista em Prolog: versão *Backward Chaining*

```
%% Interpretador Backward Chaining
:-op(800,fx,if).
:-op(700,xfx,then).
:-op(300,xfy,or).
:-op(200,xfy,and).

e_verdade(P) :- fato(P).
e_verdade(P) :- if Cond then P,
                e_verdade(Cond).
e_verdade(P1 and P2) :- e_verdade(P1),
                        e_verdade(P2).
e_verdade(P1 or P2) :-  e_verdade(P1);
                        e_verdade(P2).
```

```
%% Base de Conhecimento
if hall_molhado and cozinha_seca
   then vazamento_no_banheiro.
if hall_molhado and banheiro_seco
   then problema_na_cozinha.
if janela_fechada or sem_chuva
   then sem_água_do_exterior.
if problema_na_cozinha and sem_água_do_exterior
   then vazamento_na_cozinha.

%% Evidencias
fato(hall_molhado).
fato(banheiro_seco).
fato(janela_fechada).
```

A hipótese pode ser verificada por: `?- e_verdade(vazamento_na_cozinha). yes`

Raciocínio: *Forward Chaining*



- No encadeamento progressivo, iniciamos a partir de evidências.
- Começamos com alguns dados:
  - Por exemplo, observando que o hall está molhado e o banheiro está seco, podemos concluir que há um problema na cozinha.
  - Além disso, se observarmos que a janela da cozinha está fechada ... podemos inferir que não há água vindo do exterior
- Isto nos leva à conclusão final
  - De que há um vazamento na cozinha

Implementação do Sist.Especialista em Prolog: versão *Forward Chaining*

```
%% Interpretador Forward Chaining
:-op(800,fx,if).
:-op(700,xfx,then).
:-op(300,xfy,or).
:-op(200,xfy,and).

forward :-
    novo_fato_derivado(P), !,
    write('Derivado: '), writeln(P),
    assert(fato_derivado(P)),
    forward
      ;
    writeln('Sem mais fatos').
```

```
novo_fato_derivado(P) :-
     if Cond then P,
     \+ fato(P),
     \+ fato_derivado(P),
     e_verdade(Cond).

e_verdade(P) :- fato(P) ;
                fato_derivado(P).
e_verdade(P) :- if Cond then P,
                e_verdade(Cond).
e_verdade(P1 and P2) :- e_verdade(P1),
                        e_verdade(P2).
e_verdade(P1 or P2) :-  e_verdade(P1);
                        e_verdade(P2).
```

Implementação do Sist.Especialista em Prolog: versão *Forward Chaining* (cont.)

```
%% Base de Conhecimento
if hall_molhado and cozinha_seca
   then vazamento_no_banheiro.
if hall_molhado and banheiro_seco
   then problema_na_cozinha.
if janela_fechada or sem_chuva
   then sem_água_do_exterior.
if problema_na_cozinha and sem_água_do_exterior
   then vazamento_na_cozinha.

%% Evidencias
fato(hall_molhado).
fato(banheiro_seco).
fato(janela_fechada).
```

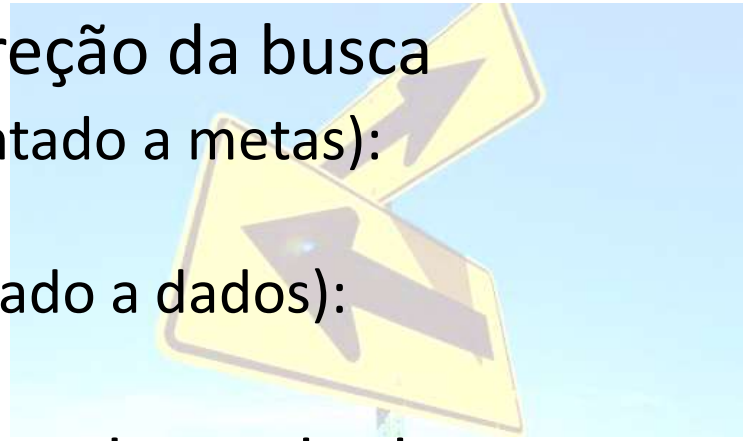
- A hipótese pode ser verificada através da consulta:

```
?- forward.
Derivado: problema_na_cozinha
Derivado: sem_água_do_exterior
Derivado: vazamento_na_cozinha
Sem mais fatos
```

Comparação entre tipos de raciocínio: Progressivo vs Regressivo

- Regras *if-then* formam uma cadeia de inferência (encadeamento) da esquerda para a direita
  - Os elementos no lado esquerdo são informações de entrada
  - Os elementos no lado direito são informações derivadas
- Estes dois tipos de informação têm uma variedade de nomes dependendo do contexto em que são utilizadas
  - dados → ... → metas,
  - evidências → ... → hipóteses
  - observações → ... → explicações, diagnósticos
  - manifestações → ... → diagnósticos, causas

Comparação entre tipos de raciocínio: Progressivo vs Regressivo



- Os tipos de raciocínios diferem-se quanto a direção da busca
  - Progressivo (também denominado raciocínio orientado a metas):
    - Parte dos dados em direção às metas
  - Regressivo (chamado também de raciocínio orientado a dados):
    - Parte das metas em direção aos dados
- A escolha entre o tipo progressivo ou regressivo depende do problema
  - Se desejamos verificar se uma hipótese particular é verdadeira então é mais natural utilizar raciocínio regressivo, iniciando com a hipótese em questão.
  - Se temos muitas hipóteses disponíveis e não há razões para começar por uma ou outra, em geral é melhor utilizar raciocínio progressivo.

Quando utilizar cada tipo? Progressivo vs Regressivo

- Em geral, o raciocínio progressivo é mais natural em tarefas de monitoramento nas quais os dados são adquiridos de forma contínua e o sistema tem que detectar a ocorrência de situação anômala.
  - Uma mudança de entrada pode ser propagada no encadeamento progressivo de se verificar, por exemplo, se esta mudança indica uma falha no processo sendo monitorado ou um alteração de desempenho.
- Adicionalmente, uma recomendação mais geral a ser usada é:
  - Se há poucos nós de dados (lado esquerdo da rede de inferência) e muitos nós metas (lado direito) então raciocínio progressivo é mais apropriado;
  - Se há poucos nós metas e muitos nós de dados então raciocínio regressivo é mais apropriado.

Quando utilizar cada tipo?
Casos especiais: Progressivo e regressivo juntos!

- Tarefas especialistas são usualmente mais complicadas e uma combinação de ambos raciocínios pode ser utilizada.
  - Em medicina, por exemplo, algumas observações iniciais do paciente disparam o raciocínio do médico na direção progressiva para gerar alguma hipótese inicial.
  - Esta hipótese inicial deve ser confirmada ou rejeitada por evidências adicionais, que podem ser obtidas utilizando raciocínio regressivo.

Explicação em um SE

- Há dois tipos usuais de explicação em um SE
  - Como? (Como o sistema chegou a uma conclusão?)
  - Por quê? (Por quê o sistema está fazendo uma determinada pergunta)

- Analisemos primeiramente a explicação ‘como’
  - Quando o sistema encontra uma solução e o usuário pergunta: Como você encontrou esta solução?
  - A explicação típica consiste em apresentar ao usuário o caminho (rastro) de como a solução foi derivada.
    - Por exemplo, suponha que os sistema encontrou que há um vazamento na cozinha e o usuário pergunta ‘Como?’
    - A explicação pode ser da seguinte forma:
      - Há um problema na cozinha, que foi concluído a partir do hall estar molhado e o banheiro seco e
      - Não há água vindo do exterior, que foi concluído a partir da janela estar fechada

Explicação: “Como?”

- Esta explicação é, de fato, uma árvore de prova em como a solução final segue a partir das regras e fatos na Base de Conhecimento.
- A árvore de prova de uma proposição **P** obedece a lógica a seguir
  - se **P** é um fato, então sua árvore de prova é **P**
  - se **P** foi derivada usando a regra
    - *if* **Cond** *then* **P**
    - então sua árvore de prova é **P <= Prova**, onde **Prova** é a árvore de prova de **Cond**
  - sejam **P1** e **P2** proposições cujas árvores de provas são **Prova1** e **Prova2**
    - A árvore de prova de **P1 and P2** é **Prova1 and Prova2**
    - A árvore de prova de **P1 or P2** é **Prova1 or Prova2**
- Utilizando o operador ‘<=‘ podemos codificar o módulo de explicação em Prolog do nosso sistema (da seguinte maneira ...)

Módulo de Explicação: Implementação

```
%% Árvore de Prova
:-op(800,fx,if).
:-op(800,xfx,<=).
:-op(700,xfx,then).
:-op(300,xfy,or).
:-op(200,xfy,and).

e_verdade(P,P) :- fato(P).
e_verdade(P, P <= ProvaCond) :- if Cond then P,
       e_verdade(Cond,ProvaCond).
e_verdade(P1 and P2, Prova1 and Prova2) :-
      e_verdade(P1,Prova1),
      e_verdade(P2,Prova2).
e_verdade(P1 or P2,Prova) :-
      e_verdade(P1,Prova) ;
      e_verdade(P2,Prova).
```

```
explique(P) :- explique(P,0).

explique(P1 and P2,T) :- !,
      explique(P1,T),
      tab(T), writeln('and'),
      explique(P2,T).

explique(P <= Cond,T) :- !,
      tab(T), write(P),
      writeln(' foi derivado a partir de'),
      T1 is T + 5,
      explique(Cond,T1).

explique(P,T) :-
    tab(T), writeln(P).
```

Módulo de Explicação: Teste de execução

```
?- e_verdade(vazamento_na_cozinha,P), explique(P).
vazamento_na_cozinha foi derivado a partir de
      problema_na_cozinha foi derivado a partir de
            hall_molhado
            and
            banheiro_seco
      and
      sem_água_do_exterior foi derivado a partir de
            janela_fechada
P = vazamento_na_cozinha <=
      (problema_na_cozinha <= hall_molhado and banheiro_seco) and
      (sem_água_do_exterior <= janela_fechada)
```

Explicação: “Porquê?”

- Um ‘por quê?’ ocorre quando o sistema solicita alguma informação e o usuário quer saber o porquê da necessidade daquela informação.
- Por exemplo, assuma que o sistema perguntou: **p** é verdade?
- O usuário pode responder:
  - Por quê? (por quê você precisa saber se p é verdade?)
- Uma explicação apropriada seria:

  Porque:
  - Eu posso usar **p** para investigar **q** pela regra **Rp** e
  - Eu posso usar **q** para investigar **s** pela regra **Rs** e
  - ...
  - Eu posso usar **y** para investigar **z** pela regra **Ry** e
  - **z** foi sua pergunta original