# Sistemas Baseados em Conhecimento

- Nesta aula serão vistos conceitos elementares sobre Sistemas Baseados em Conhecimento (SBC)
- SBC é um programa que comporta-se como um ser humano em um domínio específico do conhecimento
- Aplicações típicas de SBCs incluem diagnóstico médico, localização de falhas em equipamentos ou interpretação de dados experimentais

**Inteligência Artificial**

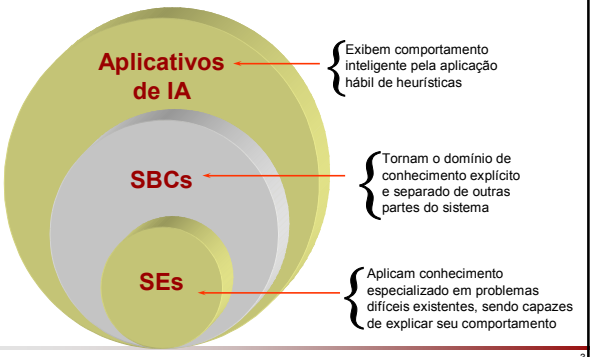
José Augusto Baranauskas
Departamento de Física e Matemática – FFCLRP-USP
E-mail: augusto@usp.br
URL: http://dfm.fmrp.usp.br/~augusto

# SBC

- Um Sistema Baseado em Conhecimento (SBC) é um programa de computador que utiliza conhecimento representado explicitamente para resolver problemas
- Ou seja, SBCs são desenvolvidos para serem usados em problemas que requerem uma quantidade considerável de conhecimento humano e de perícia para serem resolvidos
- "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



# IA, SBCs e SEs






# SBC

- 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
- Difere de sistemas convencionais na forma de incorporar o conhecimento



# Exemplo

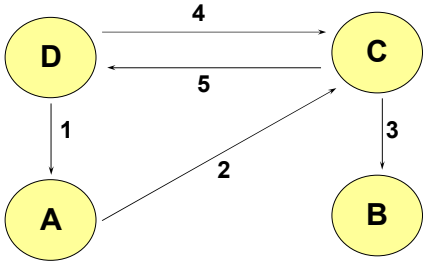
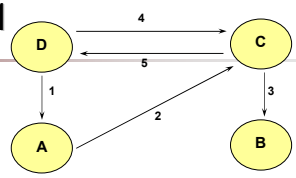



**Diagrama de Transição de Estado**

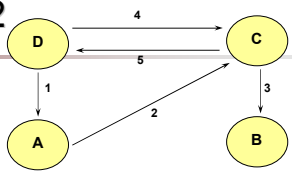


# Implementação 1




A: se 2 então desvie para estado C
B: fim
C: se 5 então desvie para estado D
  se 3 então desvie para estado B
D: se 4 então desvie para estado C
  se 1 então desvie para estado A

# Implementação 2




- Algoritmo
  - Estando o sistema no estado ***i*** e ocorrendo a entrada ***Wij***, o sistema deve ir para o estado ***j*** e repetir o processo
  - Este processo termina quando todas as entradas de ***W*** para o estado atual ***i*** estiverem vazias

- Matriz de Transição W

| Transição | A | B | C | D |
|---|---|---|---|---|
| A | | | 2 | |
| B | | | | |
| C | | 3 | | 5 |
| D | 1 | | 4 | |



# Implementação 1 x 2

- As duas implementações diferem quanto à forma de incorporar o conhecimento ao sistema
- Caso um novo evento ou um novo estado sejam adicionados, a implementação 1 precisa ser toda refeita, já na implementação 2 apenas a matriz de transição precisa ser alterada
- O mesmo é válido se um evento ou estado for removido

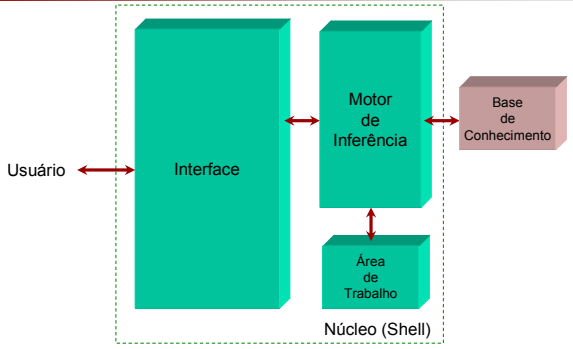


# Arquitetura

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

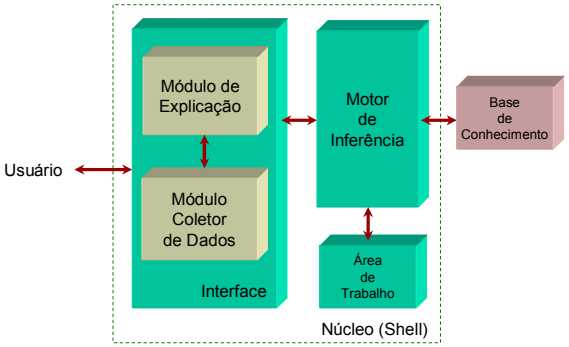


# Arquitetura






# Arquitetura






# Arquitetura

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



# Arquitetura

- 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

- Representação por relações entre objetos
- Relações mais comuns
  - is-a (é-um)
  - ako (a-kind-of) ou um-tipo-de ou faz-parte

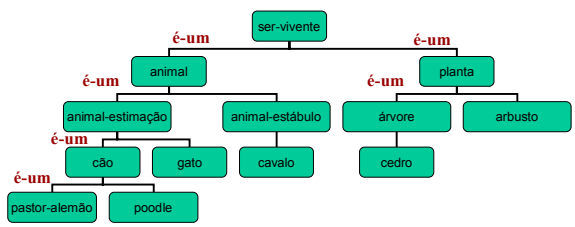


# Exemplo

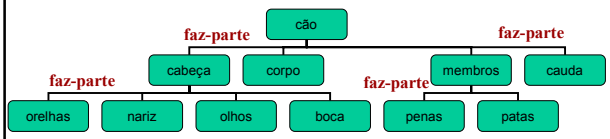
- pastor-alemão é-um cão
- poodle é-um cão
- cavalo é-um animal-estábulo
- cão é-um animal-estimação
- animal-estimação é-um animal
- animal-estábulo é-um animal
- animal é-um ser-vivente
- planta é-um ser-vivente
- árvore é-uma planta
- arbusto é-uma planta



# Hierarquia é-um (is-a)

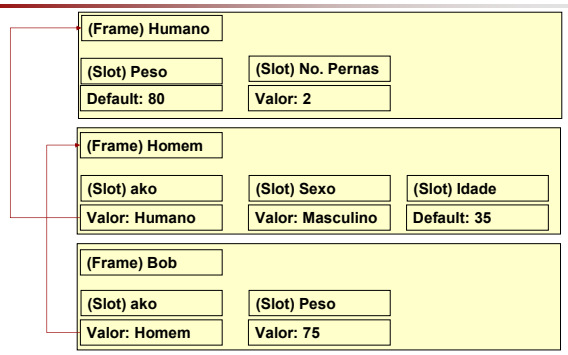


# Hierarquia faz-parte (ako)






# Frames



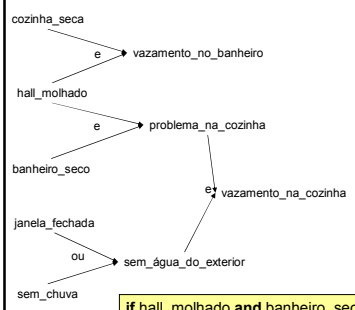


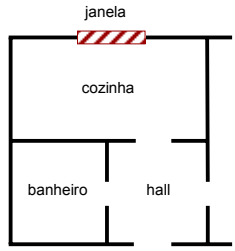

# Linhas de Raciocínio

- Uma vez que o conhecimento está representando de alguma forma, é necessário escolher um procedimento para tirar conclusões a partir da BC
- Utilizando regras if-then há duas formas
  - *backward chaining* ou encadeamento regressivo
  - *forward chaining* ou encadeamento progressivo
- Veremos, em Prolog, estes dois tipos de linhas de raciocínio



# Exemplo BC: Detectar Vazamento

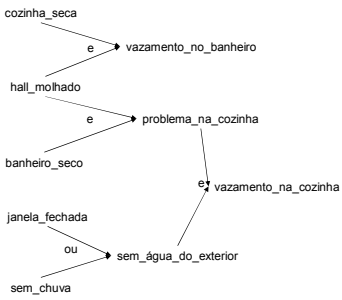



**if** hall_molhado **and** banheiro_seco **then** problema_na_cozinha



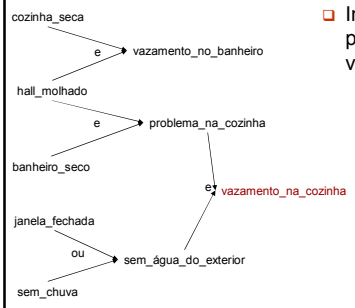
# Exemplo BC




```
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.
```

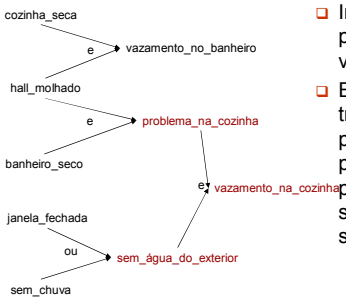


# Exemplo BC: Backward Chaining




- Iniciar com uma hipótese, por exemplo, vazamento_na_cozinha

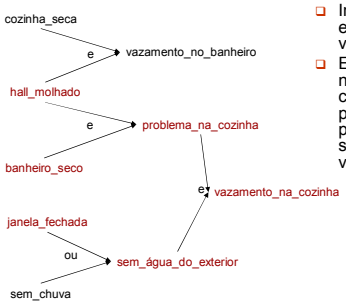
# Exemplo BC: Backward Chaining




- 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



# Exemplo BC: 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



# Exemplo BC: Backward Chaining

- 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

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

- A hipótese pode ser verificada por

```
?- vazamento_na_cozinha.
yes
```



# Linhas de Raciocínio

- A utilização da sintaxe Prolog, como no exemplo anterior, tem algumas desvantagens
  - A sintaxe pode não ser adequada para um usuário não familiarizado com Prolog; por exemplo o especialista do domínio deve ser capaz de ler as regras, especificar novas ou mesmo alterá-las
  - A BC não é sintaticamente distinguível do resto do programa; uma distinção explícita entre a BC e o restante do programa Prolog é desejável
- Para tanto, vamos usar a notação de operadores Prolog, escolhendo 'if', 'then', 'and' e 'or' como operadores de tal forma que possamos escrever
  - if hall_molhado and banheiro_seco then problema_na_cozinha.
- ao invés de
  - problema_na_cozinha :- hall_molhado, banheiro_seco.
- Além disso, vamos declarar as evidências observadas por meio da relação **fato/1**, por exemplo:
  - fato(hall_molhado).



# Linhas de Raciocínio

- 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
- No slide seguinte temos esse novo interpretador bem como a BC sobre detecção de vazamento
- O interpretador pode ser chamado agora da forma
  - e_verdade(vazamento_na_cozinha).



# 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).
```

```
%% BC
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

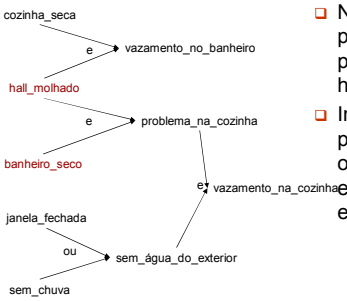
```
?- e_verdade(vazamento_na_cozinha).
yes
```

# Linhas de Raciocínio

- A desvantagem prática desse mecanismo simples de inferência é que o usuário deve fornecer antecipadamente todas as informações relevantes como fatos, antes que o processo de raciocínio tenha início
- Portanto seria mais interessante se a informação fosse fornecida pelo usuário, de forma interativa, somente quando necessária
- Esta abordagem será vista mais adiante nesta apresentação



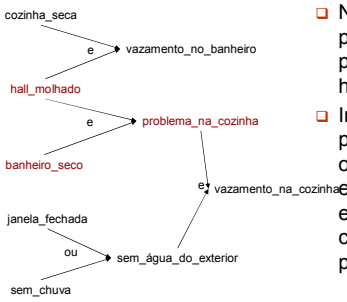
# Exemplo BC: Forward Chaining




- No encadeamento progressivo, iniciamos a partir de evidências (e não hipóteses)
- Iniciar com alguns dados, por exemplo, se observarmos que o hall está molhado e o banheiro está seco...



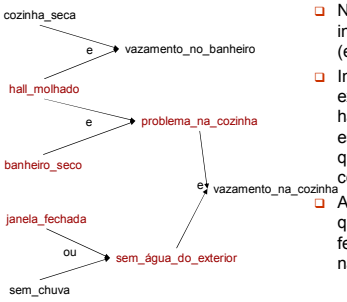
# Exemplo BC: Forward Chaining




- No encadeamento progressivo, iniciamos a partir de evidências (e não hipóteses)
- Iniciar com alguns dados, por exemplo, se observarmos que o hall está molhado e o banheiro está seco podemos concluir que há um problema na cozinha



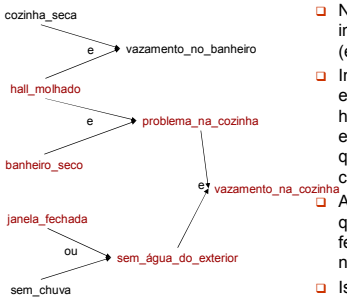
# Exemplo BC: Forward Chaining




- No encadeamento progressivo, iniciamos a partir de evidências (e não hipóteses)
- Iniciar com alguns dados, por exemplo, se observarmos 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



# Exemplo BC: Forward Chaining




- No encadeamento progressivo, iniciamos a partir de evidências (e não hipóteses)
- Iniciar com alguns dados, por exemplo, se observarmos 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 que há um vazamento na cozinha



# Forward Chaining

```
%% Interpretador Forward Chaining
:-op(800,fx,if).
:-op(700,xfx,then).
:-op(300,xfy,or).
:-op(200,xfy,and).
:- dynamic fato_derivado/1.

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(P1 and P2) :-
  e_verdade(P1),
  e_verdade(P2).
e_verdade(P1 or P2) :-
  e_verdade(P1)
  ;
  e_verdade(P2).
```

```
%% BC
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

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

# Raciocínio Progressivo x 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
- Informação de entrada → ... → informação derivada
- 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
- Raciocínios progressivo e regressivo diferem na direção da busca
  - Progressivo: parte dos dados em direção às metas
    - Também denominado raciocínio orientado a metas
  - Regressivo: parte das metas em direção aos dados
    - Também denominado raciocínio orientado a dados



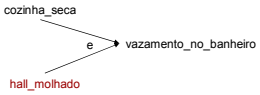
# Raciocínio Progressivo x Regressivo

- A escolha entre raciocínio 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 há muitas hipóteses disponíveis e não há nenhuma razão para começar por uma ou outra, em geral é melhor utilizar raciocínio progressivo
- 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 nos dados de entrada pode ser propagada no encadeamento progressivo para verificar se esta mudança indica alguma falha no processo sendo monitorado ou um alteração de desempenho
- Além disso, em geral, 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



# Raciocínio Progressivo x Regressivo

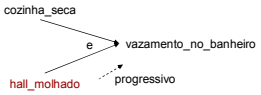
- Tarefas especialistas são usualmente mais complicadas e uma combinação de ambos raciocínios pode ser utilizada
- Em medicina, por exemplo, algumas observações iniciais do paciente disparam o raciocínio do médico na direção progressiva para gerar alguma hipótese inicial
- Esta hipótese inicial deve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






# Raciocínio Progressivo x Regressivo

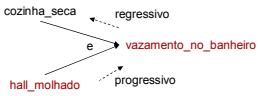
- Tarefas especialistas são usualmente mais complicadas e uma combinação de ambos raciocínios pode ser utilizada
- Em medicina, por exemplo, algumas observações iniciais do paciente disparam o raciocínio do médico na direção progressiva para gerar alguma hipótese inicial
- Esta hipótese inicial deve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






# Raciocínio Progressivo x Regressivo

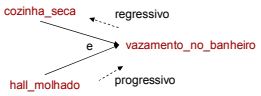
- Tarefas especialistas são usualmente mais complicadas e uma combinação de ambos raciocínios pode ser utilizada
- Em medicina, por exemplo, algumas observações iniciais do paciente disparam o raciocínio do médico na direção progressiva para gerar alguma hipótese inicial
- Esta hipótese inicial deve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






# Raciocínio Progressivo x Regressivo

- 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

- 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)
- Vamos analisar primeiramente a explicação 'como'
  - Quando o sistema encontra uma solução, o usuário pode perguntar: Como você encontrou esta solução?
  - A explicação típica consiste em apresentar ao usuário o caminho (rastro) de *como* a solução foi derivada
  - 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 BC
- Vamos representar a árvore de prova de uma proposição **P** da seguinte forma (usando operador <=)
  - 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**



# Explicação 'Como?'

```
%% Arvore 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).
```



# Explicação 'Como?'

```
?- 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)
```

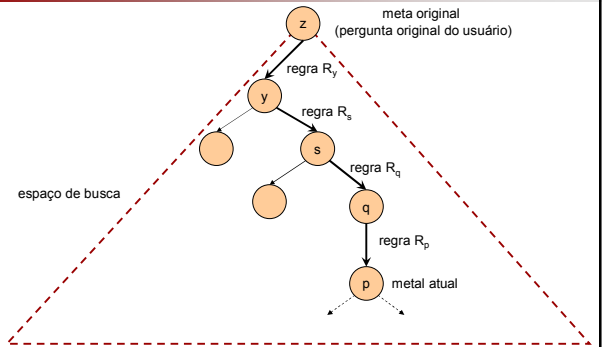


# Explicação 'Por quê?'

- 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 $\mathbf{R_p}$ e
  - Eu posso usar **q** para investigar **s** pela regra $\mathbf{R_s}$ e
  - ...
  - Eu posso usar **y** para investigar **z** pela regra $\mathbf{R_y}$ e
  - **z** foi sua pergunta original



# Explicação 'Por quê?'



# Incerteza

- Existem domínios nos quais as respostas vão além de **verdadeiro** e **falso**, por exemplo, **altamente provável**, **provável**, **improvável**, **impossível**
- Alternativamente o grau de crença pode ser expresso por um número real que varia em um intervalo, por exemplo [0,1], [-1,+1], [-5,+5]
- Estes número são conhecidos como **fatores de certeza**, **grau de crença** ou **probabilidade subjetiva**



# Incerteza

- Vamos assumir que às proposições ou regras possam ser adicionados um número no intervalo [0,1]
  - Proposição : FatorCerteza
  - if Condição then Conclusão : FatorCerteza
- Se P1 e P2 são proposições com certezas c(P1) e c(P2):
  - c(P1 e P2) ≡ mín{c(P1), c(P2)}
  - c(P1 ou P2) ≡ máx{c(P1), c(P2)}
  - c(if P1 then P2 : C) ≡ c(P2) = c(P1) * C

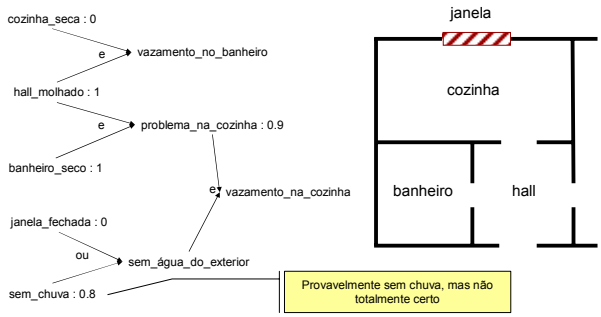


# Incerteza

- O interpretador seguinte assume que as estimativas de certeza para as evidências (região mais à esquerda da rede de regras) sejam especificadas pela relação certeza/2
  - certeza(Proposição,FatorCerteza)
- Por exemplo, a situação em que o hall está molhado, banheiro seco, cozinha não seca, janela não fechada e o usuário pensa que não há chuva mas não está totalmente certo pode ser especificado como:
  - certeza(cozinha_seca,0).
  - certeza(hall_molhado,1).
  - certeza(banheiro_seco,1).
  - certeza(janela_fechada,0).
  - certeza(sem_chuva,0.8).

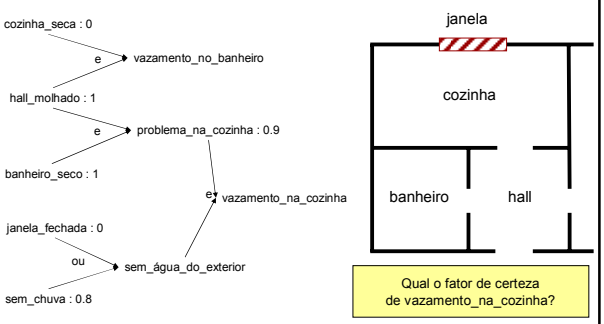


# Incerteza



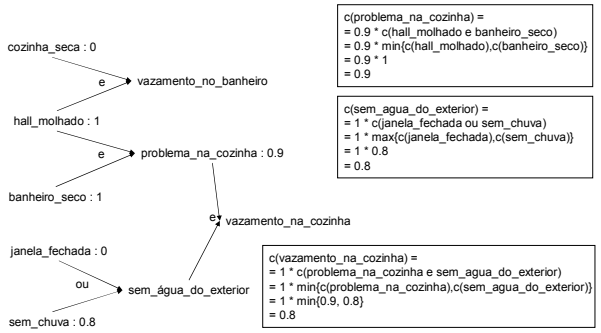



# Incerteza






# Incerteza



# Incerteza

```
fatorCerteza(P,Cert) :-
  certeza(P,Cert).
fatorCerteza(P1 and P2, Cert) :-
  fatorCerteza(P1,Cert1),
  fatorCerteza(P2,Cert2),
  Cert is min(Cert1, Cert2).
fatorCerteza(P1 or P2, Cert) :-
  fatorCerteza(P1, Cert1),
  fatorCerteza(P2, Cert2),
  Cert is max(Cert1, Cert2).
fatorCerteza(P, Cert) :-
  if Cond then P : C1,
  fatorCerteza(Cond, C2),
  Cert is C1 * C2.
fatorCerteza(P, Cert) :-
  if Cond then P,
  fatorCerteza(Cond, Cert).
```

```
%% BC
if hall_molhado and cozinha_seca
   then vazamento_no_banheiro.
if hall_molhado and banheiro_seco
   then problema_na_cozinha : 0.9.
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
certeza(cozinha_seca,0).
certeza(hall_molhado,1).
certeza(banheiro_seco,1).
certeza(janela_fechada,0).
certeza(sem_chuva,0.8).

?- fatorCerteza(
   vazamento_na_cozinha,F).
F = 0.8
```



# Exemplo Núcleo SE (Bratko)

- No NSE é utilizado o formato de regras if-then com uma informação adicional, o nome da regra, no formato
  - NomeRegra :: if Condição then Conclusão.
- Fatos são representados como
  - fato :: Fato.
- Aquilo que pode ser perguntado ao usuário é definido pela relação perguntável/2



# Exemplo Núcleo SE (Bratko)

- Por exemplo, a regra
  - regra1 :: if Animal tem pelo or Animal da leite then Animal eum mamifero.
- O nome da regra é regra1
- A regra significa que se um animal tem pelo ou dá leite então esse animal é um mamífero



# Exemplo Núcleo SE (Bratko)

- Para encontrar uma resposta **R** à pergunta **P**, o NSE funciona de acordo com os seguinte princípios
  - se **P** é um fato então **R** é '**P** é verdade'
  - se há uma regra da forma
    - if **Condição** then **P**
    - então explore a **Condição** e utilize o resultado para construir uma resposta **R** à pergunta **P**
  - se **P** é *perguntável* então pergunte ao usuário sobre **P**
  - se **P** é da forma **P1 and P2** então explore **P1** e então
    - se **P1** é falso então **R** é '**P** é falso' senão explore **P2** e combine apropriadamente ambas respostas à **P1** e **P2** em **R**
  - se **P** é da forma **P1 or P2** então explore **P1** e então
    - se **P1** é verdade então **R** é '**P** é verdade' ou alternativamente explore **P2** e combine apropriadamente ambas respostas à **P1** e **P2** em **R**



# Exemplo Núcleo SE (Bratko)

```
regra1 :: if Animal tem pelo
            or
            Animal da leite
          then Animal eum mamifero.

regra2 :: if Animal tem penas
            or
            Animal voa and Animal poe ovos
          then Animal eum passaro.

regra3 :: if Animal eum mamifero and
            (Animal come carne
             or
             Animal tem 'dentes pontudos' and
             Animal tem garras and
             Animal tem 'olhos frontais'
            )
         then Animal eum carnivoro.
```



# Exemplo Núcleo SE (Bratko)

```
regra4 :: if Animal eum carnivoro and
             Animal tem 'cor amarelo tostado' and
             Animal tem 'manchas pretas'
          then Animal eum leopardo.
regra5 :: if Animal eum carnivoro and
             Animal tem 'cor amarelo tostado' and
             Animal tem 'listras escuras'
          then Animal eum tigre.
regra6 :: if Animal eum passaro and
             Animal nao voa and
             Animal nada
          then Animal eum pinguim.
regra7 :: if Animal eum passaro and
             Animal eum 'bom voador'
          then Animal eum albatroz.

fato :: X eum animal :-
  pertence(X, [leopardo, tigre, pinguim, albatroz]).
```

## Exemplo Núcleo SE (Bratko)

```
?- main.
Pergunta por favor?
luke eum tigre.

Eh verdade: luke tem pelo? s.
Eh verdade: luke come carne? s.
Eh verdade: luke tem cor amarelo tostado? porque.
Para investigar, pela regra5, luke eum tigre
Esta foi a sua pergunta
Eh verdade: luke tem cor amarelo tostado? n.
Eh verdade: luke tem dentes pontudos? s.
Eh verdade: luke tem garras? s.
Eh verdade: luke tem olhos frontais? s.
Eh verdade: luke da leite? n.
luke eum tigre eh false
Gostaria de saber como? s.

luke eum tigre eh false
  foi derivado pela regra5 a partir
    luke tem cor amarelo tostado eh false
      foi informado
```



## Implementações - Exterior

- Reconhecimento de Voz
  - Hearsay I, II e III
- Aplicações Médicas
  - Mycin, Emycin, Puff, Expert, Internist
- Reconhecimento de Linguagem Natural
  - Mergie, Shrdlu, Gus
- Matemática
  - Macsyma, Mathlab



## Implementações - Brasil

- Aplicações Médicas
  - SIMIME (IME-USP)
- Previsão
  - SE p/ análise financeira
- Controle
  - Apoio à operação do metrô de São Paulo
- Reconhecimento de Linguagem Natural
  - Interface LN p/ SQL



## Mais Alguns Exemplos

- Alguns exemplos mais recentes incluem:
  - PROSPECTOR
  - CONSELHEIRO DIPMETER
  - FOSSIL
  - SPAM
  - ACE
  - RESEDA
  - PUFF
  - CENTAUR



## MYCIN

- MYCIN: um dos primeiros, e talvez o melhor Sistema Especialista conhecido
- É um Sistema Especialista médico, projetado para ser capaz de auxiliar médicos no diagnóstico de doenças infecciosas, e então sugerir o possível tratamento
- Como testes de laboratório podem tomar um certo número de dias e o tratamento de muitas doenças infecciosas necessita de um rápido início, é necessário diagnosticar o paciente o mais rápido possível
- MYCIN pode dar diagnósticos antes que os resultados de laboratórios estejam disponíveis



## MYCIN

- MYCIN também inclui:
  - Uma base de conhecimento consistindo de 500 regras de produção (regras if-then)
  - Regras probabilísticas, que permitem traçar conclusões plausíveis a partir de evidências incertas
  - *Backward chaining* exaustivo para encontrar um diagnóstico
  - Facilidade de explicações que pode explicar seu raciocínio e
  - Uma interface para linguagem natural, utilizável por qualquer um

## Uma Sessão Exemplo com MYCIN

- MYCIN começa com a inclusão de informações dos sintomas do paciente e sua ficha médica. Então ele pergunta:
  - MYCIN: Qual é a infecção?
  - Médico: Endarteritis.
  - MYCIN: Is Organism-1 a rod or a coccus?
  - Médico: Rod.
- MYCIN então pergunta questões adicionais...



## Uma Sessão Exemplo com MYCIN

- Para garantir que o médico possa confiar no diagnóstico do MYCIN, o programa fornece as evidências:
  - Médico: Como você decidiu que o Organism-1 é E. coli?
  - MYCIN: Eu usei a regra 084 para concluir que Organism-1 é E. coli. Isto segundo um fator de certeza de 0.51.
- Depois de obter um diagnóstico, MYCIN recomendará um tratamento adequado



## Pontos Importantes no Desenvolvimento de um SE

- Aquisição do Conhecimento:
  - Grande gargalo na construção de um SE
- Ferramentas de suporte à construção de SEs
  - Linguagens de programação
  - Ferramentas de apoio
  - Shells



Slides baseados nos livros:

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

Russell, Stuart J. & Norvig, Peter. *Inteligência Artificial*, 2ª edição, Campus, 2004

Material elaborado por
José Augusto Baranauskas
Revisão 2005