

Conhecimento e Raciocínio

Aula 2

Sistemas Periciais

2.3 Motor de Inferência

Viriato A.P. Marinho Marques

DEIS - ISEC

2010 / 2011



4.1 Inferência com Regras

4.1 Inferência com Regras

A estratégia de inferência mais comum nos SP é o *modus ponens* (MP):

P	<i>modus ponens</i>	Está a chover
$\frac{P \rightarrow Q}{Q}$		<u>Se chover levo o guarda-chuva</u>
		Então, levo o guarda-chuva.

Como se pode verificar, o MP não é mais que uma formalização das regras IF...THEN de que temos vindo a falar.

As responsabilidades do Motor de Inferência são:

1. Determinar como iniciar o processo de inferência
2. Qual a regra a disparar a cada momento e escolher entre várias que se encontrem simultaneamente activadas
3. Conduzir todo o processo de inferência: como chegar à solução ?

4.2 Regras e Pesquisa da Solução

4.2 Regras e Modos de Pesquisa da Solução

A função de um SP é, em última análise, procurar a solução de um dado problema. Esta pesquisa pode ser realizada de duas formas básicas:

- **Forward Chaining.** Mais apropriada a tarefas tais como a selecção de um produto.
- Esta é a forma de inferência no CLIPS por defeito, mas pode ser alterada com programação adicional.
- CLIPS Selecção de Vinhos [ForwardBackward_Clips\DEMO_wine.txt](#)
- EXSYS Selecção de Câmara [ExSysCamCorder_Demo\camcorder.HTML](#)
- **Backward Chaining.** Mais apropriada a tarefas do tipo diagnóstico. Esta é a forma de inferência por defeito no EXSYS, mas pode ser alterada regra a regra ou globalmente, através de especificações feitas nos COMMAND BLOCKS.
- EXSYS Lâmpada avariada [ForwardBackward_Exys\bulb.HTML](#)
- CLIPS Classificação de animal [ForwardBackward_Clips\DEMO_animal.txt](#)

Em que diferem estas formas de inferência ?

4.2 Modos de Pesquisa da Solução

Forward Chaining ou *Data Driven*

A) Suponhamos que pretendemos ir de Lisboa para Tóquio. Uma hipótese consiste em começar por procurar todos os voos que partem de Lisboa. Depois seguir o seu percurso pelas várias cidades e finalmente seleccionar apenas os que chegam a Tóquio.

B) Suponhamos que pretendemos escolher um carro. Podemos fornecer inicialmente algumas características pretendidas. Estes dados, juntamente com outros a indicar, permitem estreitar o leque de opções e chegar a uma pequena lista de propostas finais.

Backward Chaining ou *Goal Driven*

A) Suponhamos que pretendemos ir de Lisboa para Tóquio de avião. Outra hipótese consiste em começar por procurar todos os voos que chegam a Tóquio. Depois seguir o percurso inverso e finalmente seleccionar apenas os que partem de Lisboa.

B) Suponhamos que um carro não pega. Quais as causas possíveis? Elas podem ser conhecidas progredindo "para trás", isto é, da consequência para as diversas causas possíveis. Exemplo de uma regra possível neste caso: **SE o carro não tem gasolina ENTÃO o carro não pega.** Comunica-se ao sistema que o carro não pega. Em *backward chaining* ele tentará saber se o carro tem ou não gasolina (perguntando-o ao operador) para provar que o objectivo *não pega* se deve a esse facto.

4.2 Modos de Pesquisa da Solução

Combinando *Forward* e *Backward Chaining*

Estas formas de inferência podem ser combinadas (no ExSys podem ser estabelecidas para uma regra ou grupos de regras). O interesse disto depende do tipo de problema:

Exemplo

Suponhamos que um perito de investigação criminal pretende encontrar um fugitivo Z:

- Da última vez que foi visto apanhava um avião em Buenos Aires
- Só fala inglês e alemão mas prefere alemão
- Sofre de uma doença que obriga a cuidados médicos regulares
- Detesta grandes cidades, multidões e tráfego
- É jogador habitual especialmente de corridas de cavalos

Baseado nestes factos, o investigador conclui que Z abandonará a Argentina e seguirá para uma pequena localidade onde exista uma comunidade alemã: → Paraguai. Até aqui usou *forward chaining* porque partiu de factos e foi restringindo as localidades (conclusões) possíveis.

Passa agora para *backward chaining*: o *goal* é "uma pequena localidade no Paraguai que tenha cuidados médicos e esteja perto de uma pista de cavalos". A localidade A tem? A localidade B tem? etc. Quando encontrar uma nestas condições, tem uma possível solução.

4.2.1 Backward Chaining

4.2.1 *Backward Chaining*

É uma abordagem *goal-driven* (guiada por objectivo):

- Inicia-se com um ou vários *goals* que se pretendem provar Verdadeiros ou Falsos
- Toma o primeiro *goal*
- Procura uma regra que tenha como conclusão o *goal* em consideração
- Observa a premissa desta regra e verifica se a pode satisfazer:
 - Para isso poderá ter de tomar como *goal* actual a premissa agora observada e procurar regras que a tenham como conclusão
 - Este processo continua até que consegue concluir, eventualmente através de factos novos fornecidos durante o processo de inferência, que o *goal* inicial pode ser provado ou não pode ser provado:
 - No primeiro caso passa ao *goal* seguinte (se existir mais do que um) e procura uma regra que tenha como conclusão este novo *goal*
 - No segundo caso procura outra regra que tenha como conclusão o *goal* inicial
- Repete o processo



4.2.1 Backward Chaining

Exemplo

Pretendemos decidir se vamos investir em acções da IBM:

Variáveis:

- A=Ter \$10.000
- B=22<Idade<30
- C=Educação nível médio
- D=Rendimento> \$40.000
- E=Investir em seguros
- F=Investir em acções
- G=Investir em acções da IBM

1. Todas as variáveis são booleanas, i.e. valor=T/F

2. A e B são conhecidas, *a priori*, como tendo o valor T

Regras:

- R1: If A and C then E
- R2: If D and C then F
- R3: If B and E then F
- R4: If B then C
- R5: If F then G

Backward Chaining

Condições Iniciais

- O *goal* é conhecer G como verdadeiro ou falso
- A única regra de conclusão G é R5 (se houvesse várias o motor escolheria uma)

4.2.1 Backward Chaining

Regras:

R1: If A and C then E

R2: If D and C then F

R3: If B and E then F

R4: If B then C

R5: If F then G

Backward Chaining

Passo 1

- Verificar se G está na lista de factos iniciais. Não está.

Passo 2

- Analisar a premissa de R5: é F que não é conhecido
- Tomar F como *goal* actual

Passo 3

- Procurar regras de conclusão F. Há duas: R2 e R3.
- Escolher uma arbitrariamente: R2

Passo 4

- Analisar a premissa de R2. A premissa é D e C que não são conhecidos
- Tomar arbitrariamente D como *goal* actual

Passo 5

- Procurar as regras de conclusão D. Não há nenhuma.
- Pedir D ao utilizador ?
- Não: como no Passo 3 se escolheu arbitrariamente R2, faz-se *backtracking* até este passo e escolhe-se agora R3 (normalmente os pedidos de informação ao utilizador só são feitos em última instância)

NOTA: não vale a pena considerar C em vez de D porque a premissa é um AND e portanto sem se saber D, C também não interessa...

4.2.1 Backward Chaining

Regras:

R1: If A and C then E

R2: If D and C then F

R3: If B and E then F

R4: If B then C

R5: If F then G

Passo 6

- Analisar a premissa de R3. A premissa é B (conhecido=*True* e E (*not yet known*))
- Tomar E como *goal* actual

Passo 7

- Procurar regras de conclusão E. Há apenas uma: R1.

Passo 8

- Analisar a premissa de R1: é A (conhecido=*True*) e C que não é conhecido
- Tomar C como *goal* actual

Passo 9

- Procurar regras de conclusão C. Há apenas uma: R4.

Passo 10

- Analisar a premissa de R4: é B (conhecido=*True*)

Passo 11

- **R4:** Como *B=True* então *C=True*; **R1:** Como *A* e *C=True* então *E=True*; **R3:** Como *B* e *E=True*, então *F=True*; **R5:** Como *F=True*, então *G=True*

Conclusão: Este investidor (\$10.000, $22 < \text{idade} < 30$) deve comprar acções da IBM

4.2.2 Forward Chaining

4.2.2 Forward Chaining

É uma abordagem *data-driven* (guiada por dados, factos "iniciais"):

- O SP analisa as premissas das regras procurando os factos que as tornariam verdadeiras
- Se uma premissa for verdadeira a conclusão da regra é verdadeira e o SP dispara as regras que (eventualmente) estavam na dependência desta conclusão

Regras:

R1: If A and C then E

R2: If D and C then F

R3: If B and E then F

R4: If B then C

R5: If F then G

A e B são
conhecidos, *a priori*,
como *True*

Forward Chaining

Passo 0

- Como se sabe que $A=B=True$, procurar regras que contenham um destes factos na sua premissa
- Escolher A aleatoriamente
- A regra que contém A na premissa é R1

4.2.2 Forward Chaining

Regras:

R1: If A and C then E

R2: If D and C then F

R3: If B and E then F

R4: If B then C

R5: If F then G

Passo 1

- Provar a conclusão de R1: implica conhecer C
- C não figura na lista de factos iniciais

Passo 2

- Procurar uma regra que tenha como conclusão C
- É a regra R4

Passo 3

- Verificar a premissa de R4: é B
- Procurar B nos factos iniciais: é conhecido a *True*
- Disparar R4: C é asserido como *True*

Passo 4

- R1 dispara e E é asserido como *True*

Passo 5

- Procurar regras em que E figure na premissa
- Existe a regra R3

Passo 6

- Ver premissa de R3: como B=E=*True*, R3 dispara
- F é asserido a *True*

Passo 7

- Procurar regras com F na premissa
- É a regra R5

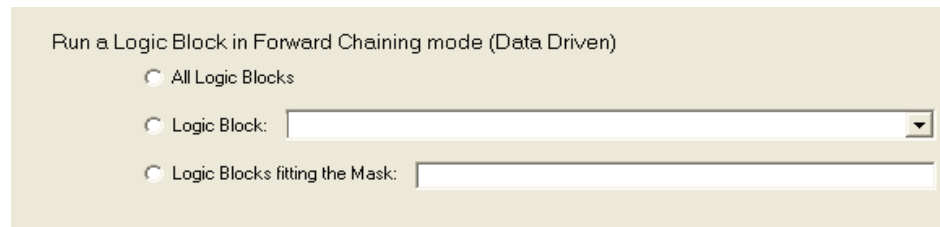
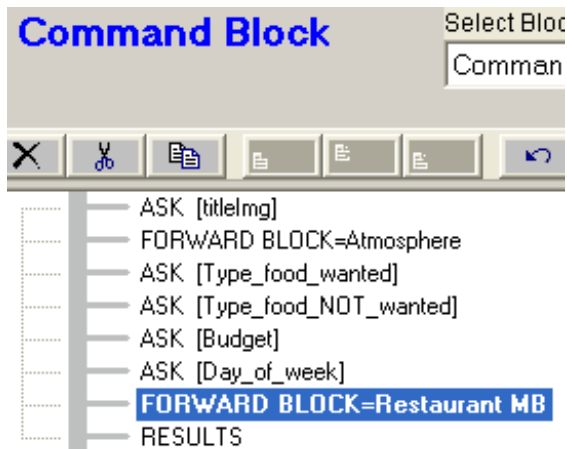
Passo 8

- Ver premissa de R5: contém apenas F
- Como F=*True*, R5 dispara e G é asserido a *True*

4.2.3 Mixed (forward+backward)

4.2.3 Mixed Chaining

É possível misturar os dois tipos de inferência (p.e. o ExSys permite especificar o tipo a utilizar regra a regra ou por bloco). Neste caso o disparo das regras faz-se conjugando os métodos anteriores.



The CORVID Inference Engine also supports another way to run the Inference Engine - forward chaining. Forward chaining is data driven, rather than goal driven.

Running the Inference Engine in this mode is done when there is a body of data already available and you just want to use the logic in the rules to analyze it. In this case the rules are tested sequentially to see what conclusions result. Forward chaining is somewhat faster for some problems, but the questions are not as focused and it is not as good an emulation of a session with a human expert.

4.2.3 Mixed (forward+backward)

4.2.3.1 Mixed Chaining - Prioridade ao *Backward*

Usar *Backward Mode*

R1: If F and H then K
R2: If E and A then K
R3: If E and B then H

Usar em *Forward Mode*

R4: If A and G then B
R5: If B and D then H
R6: If G and D then E
R7: If A and B then D
R8: If A and C then G

Condições

Goal = K

A e C são inicialmente conhecidos

Factos / Goals

0. Condições Iniciais
1. Procurar uma regra backward de conclusão K. A primeira é R1
2. Ver premissa: é F e H
3. Procurar uma regra backward de conclusão F ou H. Apenas R3
4. Ver premissa: é E e B
5. Procurar regra backward de conclusão E ou B. **Não há.** Passar às regras de Forward Chaining
6. Factos actuais: apenas A e C. Procurar regra forward de premissa A ou C ou ambos: é R8
7. Disparar R8: conclui G
8. Procurar regra forward de premissa A, C, G ou combinações destes. É R4

AC / K

AC / F H

AC / F EB

ACG / FEB

4.2.3 Mixed (forward+backward)

Usar *Backward Mode*

R1: If F and H then K
R2: If E and A then K
R3: If E and B then H


Usar em *Forward Mode*

~~R4~~: If A and G then B
~~R5~~: If B and D then H
~~R6~~: If G and D then E
~~R7~~: If A and B then D
~~R8~~: If A and C then G

Condições

Goal = K

A e C são inicialmente conhecidos

 = já disparou

Factos / Goals

9. Disparar R4: conclui B (a conclusão B permite retirá-lo da lista de *goals* e passá-lo para a lista de factos)

ACGB / FEB

10. Procurar regra forward de premissa A, C, G, B ou combinações destes factos. É R7

11. Disparar R7: conclui D

ACGBD / FE

12. Procurar regra forward de premissa A, C, G, B, D ou combinações destes factos. Há R5 e R6.

13. Disparar R5 e R6: conclui H e E (remover E da lista de *goals*)

ACGBDHE / FE

14. Todas as regras forward foram disparadas e o *goal* F ainda não foi atingido: retornar às regras backward e dispará-las em forward


4.2.3 Mixed (forward+backward)

Usar *Backward Mode*

R1: If F and H then K

R2: If E and A then K

R3: If E and B then H

 = já disparou

Usar em *Forward Mode*

~~R4~~: If A and G then B

~~R5~~: If B and D then H

~~R6~~: If G and D then E

~~R7~~: If A and B then D

~~R8~~: If A and C then G

Condições

Goal = K

A e C são inicialmente conhecidos

Factos / Goals

15. Procurar regra backward de premissa A, C, G, B, D, H, E ou combinações destes factos: há duas, R2 e R3

16. Disparar R2: conclui K que é o *goal*: objectivo atingido.

ACGBDHE / F

ACGBDHE**K** / ~~F~~

4.2.3.1 Mixed Chaining - Prioridade ao *Forward*

O exemplo anterior também pode ser resolvido dando prioridade ao *forward chaining*. Vamos considerar as mesmas regras e condições iniciais.

4.2.3 Mixed (forward+backward)

Usar *Backward Mode*

R1: If F and H then K
R2: If E and A then K
R3: If E and B then H

Usar em *Forward Mode*

R4: If A and G then B
R5: If B and D then H
R6: If G and D then E
R7: If A and B then D
R8: If A and C then G

Condições

Goal = K

A e C são inicialmente conhecidos

Factos / Goals

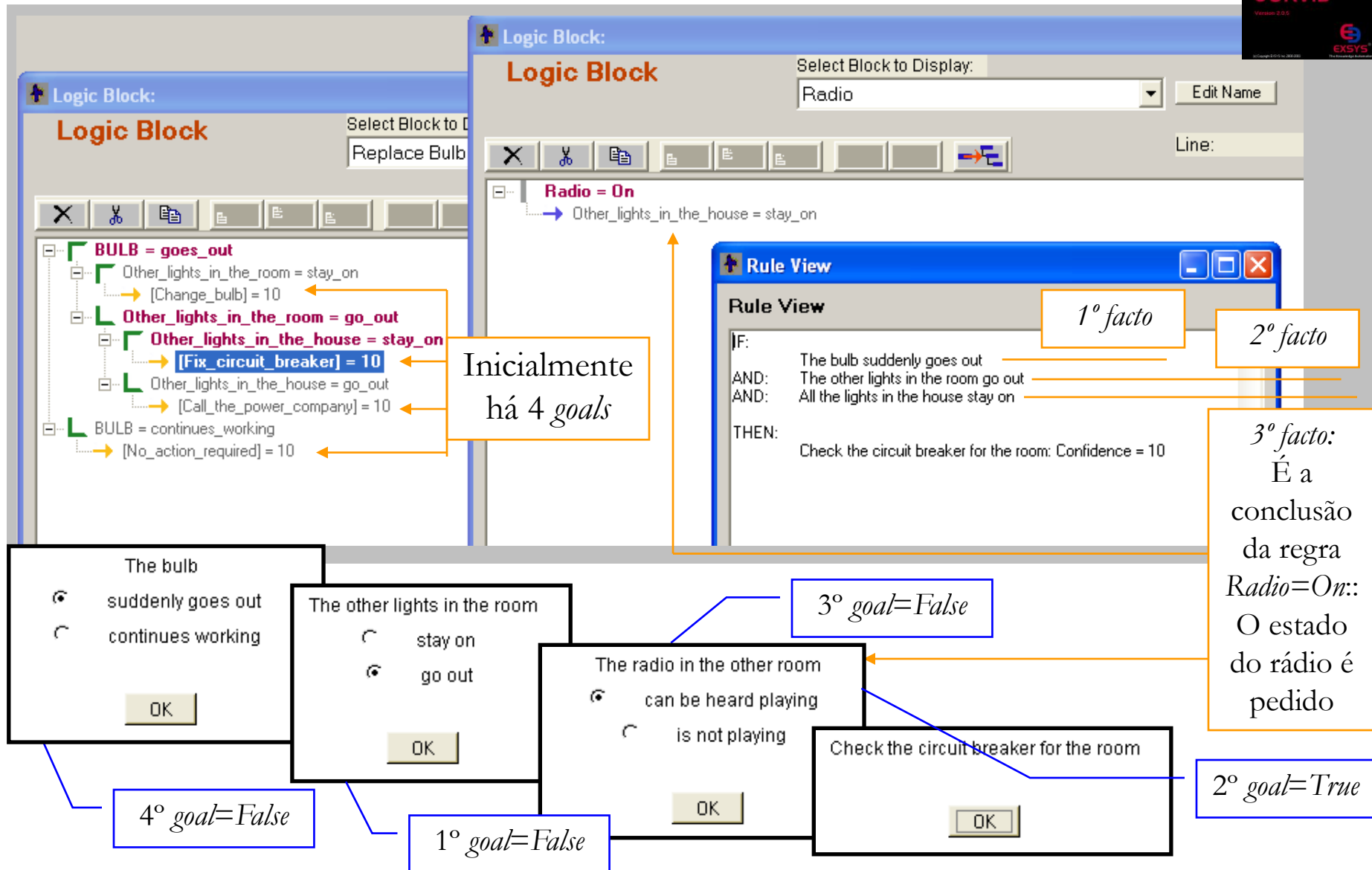
0. Condições Iniciais
1. De R8 infere-se G
2. De R4 infere-se B
3. De R7 infere-se D
4. De R5 infere-se H
5. De R6 infere-se E
6. Não há mais regras forward e K não foi atingido. Tentar backward chaining
7. Procurar regra de conclusão K: há duas, R1 e R2
8. Ver premissa de R1: é F e H, nada se pode concluir
9. Ver premissa de R2: é E e A. Disparar R2, infere-se K

AC / K
ACG / K
ACGB / K
ACGBD / K
ACGBDH / K
ACGBDHE / K

ACGBDHE**K**

[illegible]

4.2.4 Árvores de Inferência



4.2.5 Why e How

4.2.5 Explicação e Justificação: *Why* e *How*

As árvores de inferência proporcionam um suporte às questões *Why* e *How* que os utilizadores podem pôr aos SP:

- **How:** Usa-se quando o SP chega a uma conclusão e se pretende saber como ele a atingiu; portanto, basta seguir os caminhos da árvore que conduzem àquela conclusão e ir mostrando as conclusões intermédias
- **Why:** Usa-se quando o utilizador quer saber porque é que o SP pede uma determinada informação (em *backward chaining*); portanto, basta identificar o objectivo (*goal*) para o qual esta informação é necessária, e mostrar as conclusões intermédias até à conclusão em consideração.



Exemplo:

The .HOW property will output a full explanation of how the value for a variable was set. This is the same explanation that is set to the end of the Trace file. This text can be used in result screen to explain how conclusions were reached.

4.4 Outros tipos de Inferência

4.4.1 Inferência com *Frames*

A inferência com *frames* baseia-se essencialmente em verificações dos valores contidos nos *slots* com consequente selecção ou exclusão de instâncias que verificam ou não condições expressas por regras.

Mais informação: A Guide to Expert Systems, *Watterman*
Dec Support Systems and Int.Systems, Efraim & Aronson

4.4.2 Inferência com Modelos (*MBR - Model Based Reasoning*)

1. Os modelos representam conhecimento acerca da estrutura e comportamento de um dado domínio. p.e. equipamento complexo.
2. São uma forma de representação de conhecimento profundo (*deep knowledge*)
3. As inferências são feitas de raiz, como se um técnico consultasse os manuais para compreender o que se passa em vez de simplesmente associar sintomas e causas através da sua experiência
4. Normalmente incorporam **simulação** do domínio

4.4.2 Inferência com Modelos

Problemas

- Exigem conhecimento detalhado e complexo nem sempre disponível
- Implementação mais difícil (justificável consoante o caso)
- Teoricamente, um SP com conhecimento base de electrónica capaz de solucionar problemas de um dado computador, será capaz de os solucionar em qualquer um...mas na prática isto é obviamente difícil

Exemplos

Na NASA um SP de (Fulton & Pepe, 1990) está ligado a uma série de sensores. Os sinais são processados por um programa de simulação que prevê as saídas. Os resultados são comparados com os de outros sensores. Se dentro da tolerância, nada é feito; senão são tomadas medidas para prevenir situações potencialmente perigosas.

O CASEY é um SP CBR de diagnóstico de doenças do coração. Incorpora o *Heart Failure Program* que modeliza relações causa → sintomas de modo a adaptar (se possível) síndromas "novos" a conjuntos de evidências já conhecidas do SP.

O PATDEX é um SP CBR de diagnóstico de falhas em sistemas complexos (Richter, 1991). Encontra-se integrado no sistema MOLTKE que lhe proporciona conhecimento causal.



4.4.2 Inferência com Modelos

Tipos de Modelos

Basicamente usam-se 2 tipos de modelos:

- Modelos Matemáticos: um modelo puramente matemático do domínio. P.e. simular o andamento de um relógio com base nas características do maquinismo
- Modelos de componentes: compostos por elementos interligados entre si que simulam o comportamento do sistema. Quando um componente altera o seu estado interno, propaga-o aos que dele dependem.

Naturalmente que existem outras hipóteses: veja-se p.e. o *Heart Failure Program* usado no CASEY.

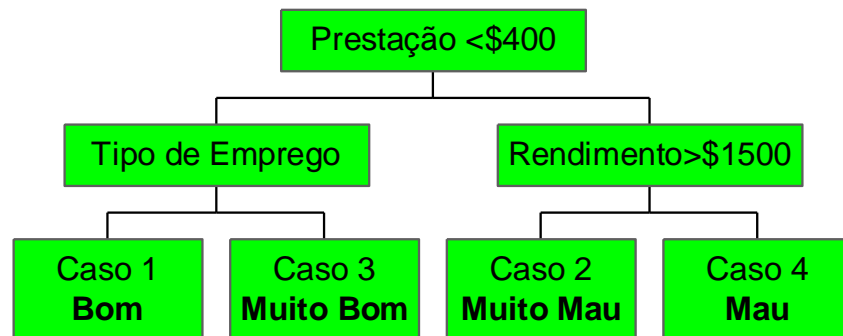


4.5 Árvores de Decisão

4.5 Árvore de Decisão

(Introduzimos aqui as Árvores de Decisão essencialmente para mostrar a representação das inferências utilizada pela Attar Software no seu xPertRule KB. Este assunto será de novo abordado nas secções referentes a Indução)

- Uma Árvore de Decisão (*Decision Tree*) recebe como entrada um objecto ou situação descrito por um conjunto de propriedades e produz uma saída do tipo Yes / No.
- Cada nó da árvore corresponde a um teste acerca do valor de uma propriedade
- Cada ramo é etiquetado com o valor desse teste
- As folhas representam um valor Booleano



Empréstimo Bancário:

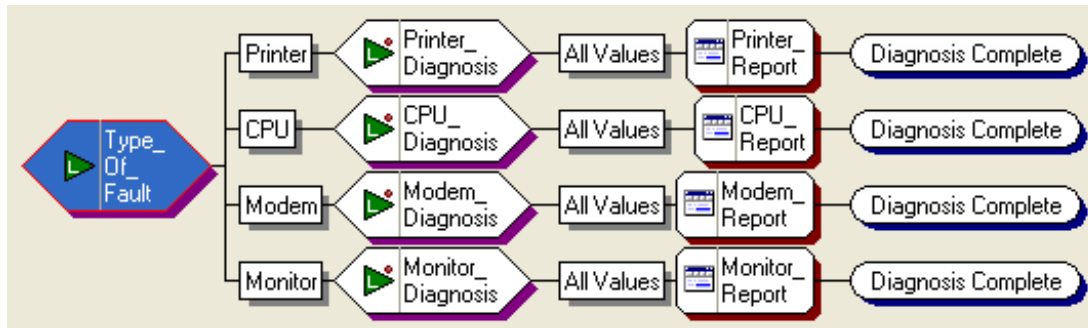
- **Atributos:** prestação, tipo de emprego, rendimento
- **Resultado:** Empréstimo Bom...Mau de valor Yes / No

4.5 Árvores de Decisão



As árvores de decisão podem ser automaticamente geradas por algoritmos próprios (p.e. ID3) a partir de exemplos. Esses exemplos não são mais do que relatos de ocorrências, descrições de factos passados, ou seja, casos.

O Attar xPertRule Knowledge Builder pode fazer exactamente isto (a Attar possui também o Miner, uma ferramenta mais completa para Data-Mining) apresentando o resultado numa árvore de decisão que pode ser lida, também, como nada mais que uma sequência de regras (semelhante a uma Árvore de inferência). Veja-se o seguinte exemplo de um SP para diagnóstico de falhas em PCs:

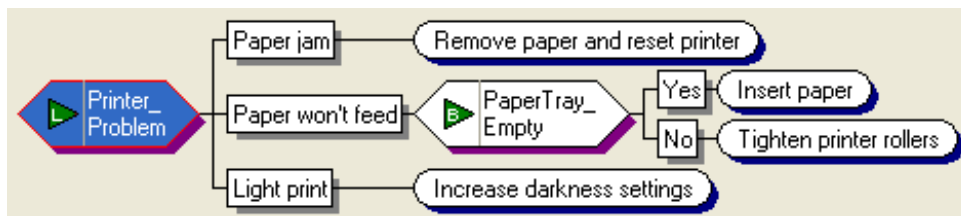


1

2

3

4

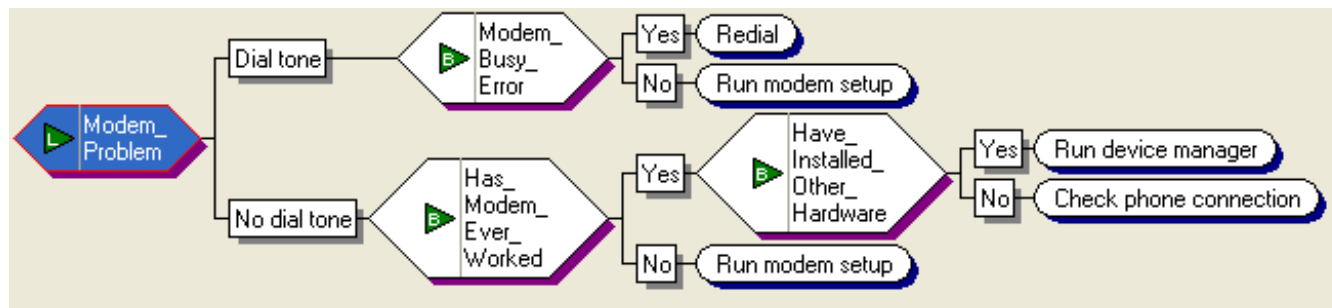
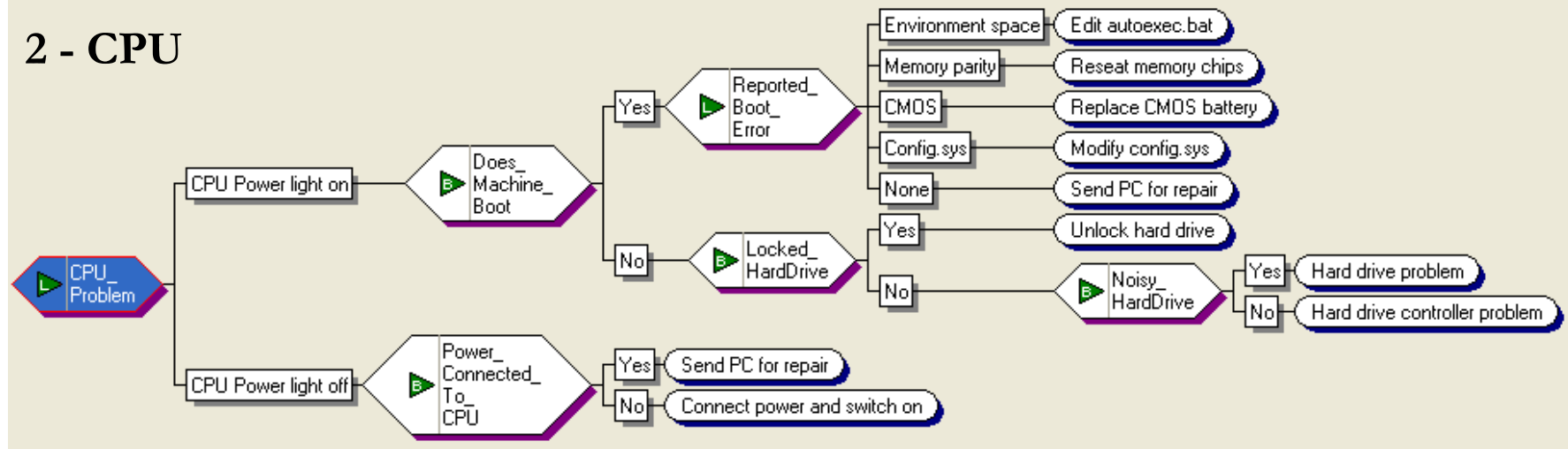


1 - Printer

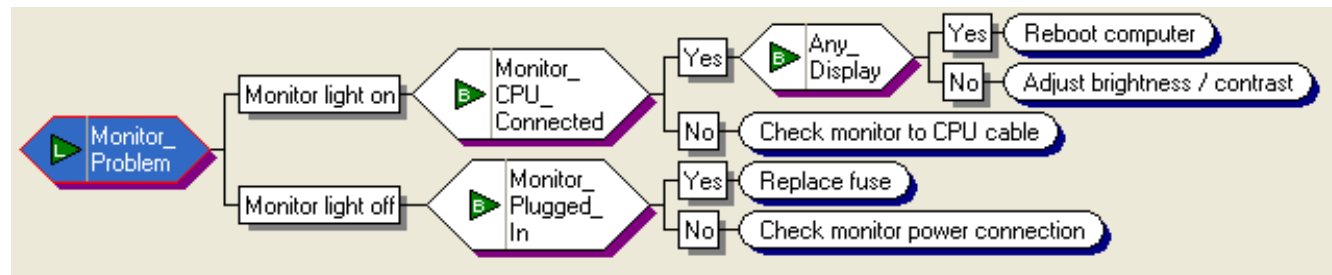
4.5 Árvores de Decisão



2 - CPU



3 - Modem



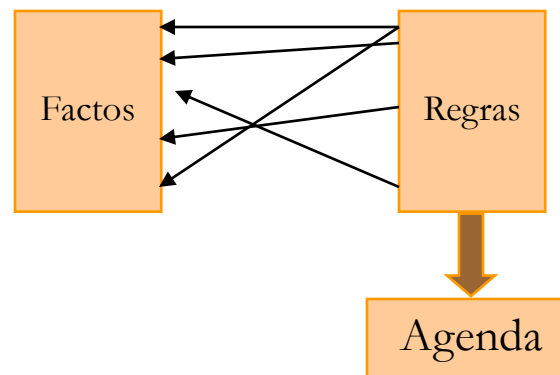
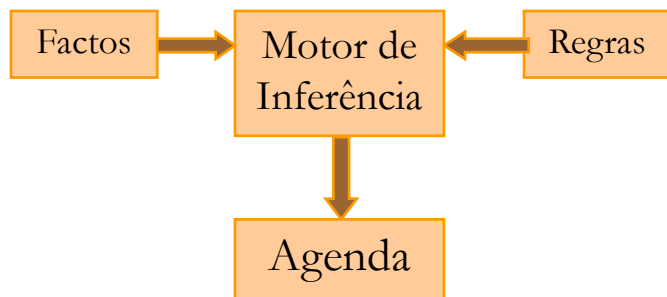
4 - Monitor

4.6 Algoritmo RETE

4.6 RETE *Pattern-Matching Algorithm*

Os *shells* CLIPS, Art*Enterprise e Eclipse (pelo menos) baseiam-se no algoritmo RETE para testar o *match* entre as cláusulas das premissas das regras e os factos conhecidos em cada "iteração".

- Sem este algoritmo, em cada iteração o motor de inferência tem de comparar **todas as cláusulas** com **todos os factos** asseridos, para saber quais as regras que pode activar



"Regras procurando Factos"

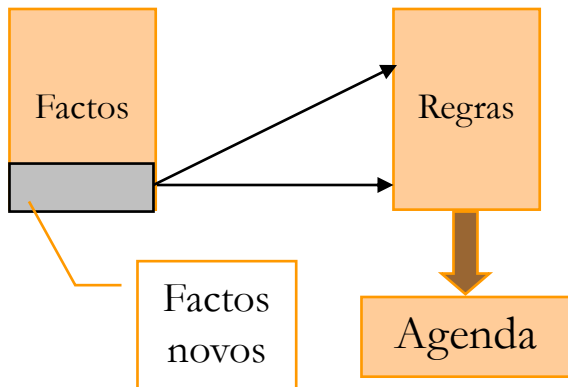
4.6 Algoritmo RETE

Geralmente em cada iteração apenas alguns factos (poucos) mudam:

- **Redundância Temporal:** consiste em testar todas as regras contra todos os factos sabendo-se que certamente apenas uma muito pequena percentagem de regras poderão ser activada "de novo" uma vez que poucos factos mudaram

O algoritmo RETE:

- Guarda o estado dos *matches* em cada iteração (i.e. os *matches parciais*: premissas que só verificavam algumas das suas cláusulas)
- Controla quais os factos novos que a iteração anterior gerou
- Como consequência testa apenas os factos novos contra as premissas de *match parcial*



"Factos procurando Regras"



- Authorete: Knowledge management and automation of business rules - *in plain English*
- Rete++: Embeddable rules engine C++
 - Cafe Rete: Business rule automation
- Eclipse: Rule-based programming language and inference rule engines