1º OBJECTIVO:
LÓGICA PROPOSICIONAL E RACIOCÍNIO AUTOMÁTICO (Prova automática de teoremas)

INTRODUÇÃO

Consideremos as seguintes declarações

1. O João acordou;
2. O João foi buscar um pano de limpar o pó;
3. A mãe fica encantada se o João acorda e limpa o seu quarto;
4. Se o João for buscar um pano de limpar o pó, então ele limpa o seu quarto.

Podemos concluir por RACÍCIONIO INTUITIVO que

A mãe está encantada.

Deduzimos um facto que não estava explicito nas 4 declarações

E se tivéssemos centenas de declarações? Precisamos de AUTOMATIZAR O RACÍOCINIO.

Precisamos de procedimentos de PROVA DE TEOREMAS implementáveis em computador e cuja base são a LÓGICA PROPOSICIONAL e a LÓGICA DE PREDICADOS.

CONCEITOS DA LÓGICA PROPOSICIONAL

PROPOSIÇÕES
São declarações como

"A terra é redonda"
"IA é uma cadeira difícl"

que possui um valor de verdade ou veracidade, isto é: é FALSA ou VERDADIRA mas não as duas

ÁTOMOS
Símbolos que representam proposições

A - A terra é redonda
B - IA é uma cadeira difícil

FÓRMULAS
Combinação de ÁTOMOS, PARÊNTISIS e OPERADORES LÓGICOS

Uma fórmula em lógica proposicional tem a seguinte definição recursiva: um átomo é uma fórmula; se F é uma fórmula então ($\neg$F) é uma fórmula; se F e G são fórmulas então (F $\wedge$ G), (F $\vee$ G), (F $\Rightarrow$ G) e (F $\Leftrightarrow$ G) são fórmulas. Não existem quaisquer outras fórmulas para além das aqui definidas.

FÓRMULA	SIGNIFICADO EM PORTUGUÊS
($\neg$F)	"não F" "negação de F"
(F $\wedge$ G)	"F e G" "conjunção de F e G"
(F $\vee$ G)	"F ou G" "disjunção de F e G"
(F $\Rightarrow$ G) ou (G $\Leftarrow$ F)	"se F então G" "F se G" "F implica G" "F é suficiente para G" "G é necessário para F" onde F é o antecedente e G o consequente
(F $\Leftrightarrow$ G)	"F se, e só se, G" "F sse G" "F é necessário e suficiente para G" "G é necessário e suficiente para F"

As várias maneiras de traduzir os diferentes tipos de fórmulas em Português.

As fórmulas também têm um valor de verdade

F	G	($\neg$ F)	(F $\wedge$ G)	(F $\vee$ G)	(F $\Rightarrow$ G)	(F $\Leftrightarrow$ G)
verdadeiro	verdadeiro	falso	verdadeiro	verdadeiro	verdadeiro	verdadeiro
verdadeiro	falso	falso	falso	verdadeiro	falso	falso
falso	verdadeiro	verdadeiro	falso	verdadeiro	verdadeiro	falso
falso	falso	verdadeiro	falso	falso	verdadeiro	verdadeiro

Tabelas de verdade (veracidade) para as fórmulas construidas com os cinco operadores lógicos.

Precedência dos operadores: $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$

INTERPRETAÇÕES DE FÓRMULAS

A interpretação de uma fórmula F corresponde à atribuição de um valor de verdade a todos os átomos que ocorrem em F

Uma fórmula contendo n átomos distintos tem 2^n interpretações distintas

Vejamos as interpretações possíveis para a fórmula $(((B \vee C) \wedge \neg C) \vee D)$

B	C	D	(B∨C)	(¬C)	((B∨C)∧¬C)	(((B∨C)∧¬C)∨D)
verdadeiro	verdadeiro	verdadeiro	verdadeiro	falso	falso	verdadeiro
verdadeiro	verdadeiro	falso	verdadeiro	falso	falso	falso
verdadeiro	falso	verdadeiro	verdadeiro	verdadeiro	verdadeiro	verdadeiro
verdadeiro	falso	falso	verdadeiro	verdadeiro	verdadeiro	verdadeiro
falso	verdadeiro	verdadeiro	verdadeiro	falso	falso	verdadeiro
falso	verdadeiro	falso	verdadeiro	falso	falso	falso
falso	falso	verdadeiro	falso	verdadeiro	falso	verdadeiro
falso	falso	falso	falso	verdadeiro	falso	falso

Tabela de verdade para a fórmula $(((B \vee C) \wedge \neg C) \vee D)$. Cada linha corresponde a uma interpretação. Algumas interpretações da fórmula fazem-na verdadeira; outras fazem-na falsa. Por exemplo, a fórmula é falsa na segunda linha, mas é verdadeira na primeira.

De uma interpretação que torna uma fórmula verdadeira dizemos que **soluciona** a fórmula

TAUTOLOGIA (ou fórmula válida)
Fórmula que é verdadeira para todas as suas interpretações $(B \vee \neg B)$

INCONSITÊNCIA (ou fórmula não solucionável)
Fórmula que é falsa para todas as suas interpretações $(\neg B \wedge B)$

CONSISTENTE (fórmula solucionável)
Se não for inconsistente, isto é, é verdadeira em pelo menos uma das interpretações

EQUIVALÊNCIAS
Uma fórmula F é considerada equivalente a uma fórmula G se, e só se, a veracidade de F é igual à veracidade de G em todas as interpretações de F e G

A equivalência é escrita como $F \equiv G$. Como exemplo, a tabela de verdade mostra que $\neg(A \vee B) \equiv (\neg A \wedge \neg B)$.

A	B	(¬A)	(¬B)	(A ∨ B)	¬ (A ∨ B)	(¬A ∧ ¬B)
verdadeiro	verdadeiro	falso	falso	verdadeiro	falso	falso
verdadeiro	falso	falso	verdadeiro	verdadeiro	falso	falso
falso	verdadeiro	verdadeiro	falso	verdadeiro	falso	falso
falso	falso	verdadeiro	verdadeiro	falso	verdadeiro	verdadeiro

Tabela de verdade para demonstrar que $\neg(A \vee B) \equiv (\neg A \wedge \neg B)$.

[01]	$\neg(\neg F)$	$\equiv F$		[envolvimento]
[02]	$(F \Rightarrow G)$	$\equiv (\neg F \vee G)$		[modus ponnens]
[03]	$(F \Rightarrow G)$	$\equiv (\neg G \Rightarrow \neg F)$		[contrapositividade]
[04]	$(F \Rightarrow (G \Rightarrow H))$	$\equiv (G \Rightarrow (F \Rightarrow H))$		
[05]	$(F \Rightarrow (G \Rightarrow H))$	$\equiv ((F \wedge G) \Rightarrow H)$		
[06]	$(F \Leftrightarrow G)$	$\equiv (((F \wedge G) \vee (\neg F \wedge \neg G))$		
[07]	$(F \Leftrightarrow G)$	$\equiv ((F \Rightarrow G) \wedge (\neg G \vee F))$		
[08]	$(F \Leftrightarrow G)$	$\equiv ((\neg F \vee G) \wedge (\neg G \vee F))$		
[09]	$(G \wedge G)$	$\equiv G$		[idempotência]
[10]	$(G \wedge$ verdadeiro$)$	$\equiv G$		
[11]	$(G \wedge$ falso$)$	$\equiv$ falso		
[12]	$(G \wedge \neg G)$	$\equiv$ falso		[exclusão]
[13]	$(G \vee G)$	$\equiv G$		[fatorização]
[14]	$(G \vee$ verdadeiro$)$	$\equiv$ verdadeiro		
[15]	$(G \vee$ falso$)$	$\equiv G$		
[16]	$(G \vee \neg G)$	$\equiv G$		
[17]	$((F \wedge G) \wedge H)$	$\equiv (F \wedge (G \wedge H))$	⎤	
[18]	$((F \vee G) \vee H)$	$\equiv (F \vee (G \vee H))$	⎦	[associatividade]
[19]	$(F \wedge G)$	$\equiv (G \wedge F)$	⎤	
[20]	$(F \vee G)$	$\equiv (G \vee F)$	⎦	[comutatividade]
[21]	$(F \vee (G \wedge H))$	$\equiv ((F \vee G) \wedge (F \vee H))$	⎤	
[22]	$(F \wedge (G \vee H))$	$\equiv ((F \wedge G) \vee (F \wedge H))$	⎦	[distributividade]
[23]	$\neg(F \vee G) \equiv (\neg F \wedge \neg G)$		⎤	
[24]	$\neg(F \wedge G) \equiv (\neg F \vee \neg G)$		⎦	[leis de Morgan]
[25]	$(F \vee (F \wedge G))$	$\equiv F$	⎤	
[26]	$(F \wedge (F \vee G))$	$\equiv F$	⎥	
[27]	$(F \vee (\neg F \wedge G))$	$\equiv (F \vee G)$	⎥	
[28]	$(F \wedge (\neg F \vee G))$	$\equiv (F \wedge G)$	⎦	[absorção]

Lista das equivalências mais frequentemente utilizadas. Algumas destas equivalências são mencionadas, na literatura sobre lógica, por nomes específicos. Esses nomes estão colocados entre parentisis rectos. F, G e H correspondem a fórmulas quer na lógica proposicional quer na lógica de predicados. As fórmulas que são tautologias estão designadas por verdadeiro; as inconsistentes por falso.

PROCEDIMENTOS DE PROVA

PROVA

G é uma **consequência lógica** das fórmulas F_1, F_2, ..., F_n se, e só se, todas as interpretações que solucionam a fórmula $(F_1 \wedge F_2 \wedge ... \wedge F_n)$ também solucionam G

O mesmo é dizer que, quando F_1, F_2, ..., F_n são verdadeiras então, G também é

F_1, F_2, ..., F_n implica G (também conhecida por **fórmula objectivo**). A F_1, F_2, ..., F_n chamamos **premissas**.

Demonstrar que G é uma consequência lógica, é provar que $((F_1 \wedge F_2 \wedge ... \wedge F_n) \Rightarrow G)$ é um **teorema**; a demonstração é a prova.

Por simplicidade de escrita podemos dizer: provou-se que G é um teorema ou simplesmente, provou-se G

PROVA DIRECTA
Demonsta-se que $((F_1 \wedge F_2 \wedge ... \wedge F_n) \Rightarrow G)$ é uma tautologia

PROVA POR REFUTAÇÃO
Uma vez que:

$$\neg((F_1 \wedge F_2 \wedge ... \wedge F_n) \Rightarrow G)$$
$$\equiv \neg(\neg(F_1 \wedge F_2 \wedge ... \wedge F_n) \vee G)$$
$$\equiv (F_1 \wedge F_2 \wedge ... \wedge F_n \wedge \neg G).$$

demonstra-se que $(F_1 \wedge F_2 \wedge ... \wedge F_n \wedge \neg G)$ é uma inconsistência

Como exemplo vamos provar, utilizando os dois métodos que G é uma consequência lógica das premissas $F \wedge (F \Rightarrow G)$.

					Prova pelo método directo	Prova pelo método da refutação
F	G	$\neg G$	$(F \Rightarrow G)$	$(F \wedge (F \Rightarrow G))$	$((F \wedge (F \Rightarrow G)) \Rightarrow G)$	$(F \wedge (F \Rightarrow G) \wedge \neg G)$
verdadeiro	verdadeiro	falso	verdadeiro	verdadeiro	verdadeiro	falso
verdadeiro	falso	verdadeiro	falso	falso	verdadeiro	falso
falso	verdadeiro	falso	verdadeiro	falso	verdadeiro	falso
falso	falso	verdadeiro	verdadeiro	falso	verdadeiro	falso

Donde verificamos existir um procedimento finito na lógica proposicional para decidir se um dado objectivo é um teorema ou não. Daí dizer-se que, a lógica proposicional é *resolúvel*. Mas para um n muito grande, o número de linhas na tabela de verdade aumenta exponencialmente, bem como o esforço computacional, pelo que a construção de tabelas de verdade para prova de teoremas por qualquer dos dois métodos seja praticável, mas não um processo prático.

PROVA POR DEDUÇÃO NATURAL (prova directa)
Neste procedimento, aplicamos uma das regras de inferência ao conjunto de premissas de modo a deduzir uma fórmula consequente lógica, de tal maneira que, a fórmula deduzida contenha apenas átomos das premissas e do objectivo. Esta fórmula é então acrescentada ao conjunto das premissas. Este processo é repetido até que tenhamos deduzido uma fórmula idêntica ao objectivo (provou-se que o objectivo é um teorema) ou, até não conseguirmos deduzir mais nenhuma fórmula nova (provou-se que o objectivo não é um teorema).

[01] Se F e G então (F $\wedge$ G). [introdução da conjunção]

[02] Se (F $\wedge$ G) então F. ⎤
[03] Se (F $\wedge$ G) então G. ⎦ [eliminação da conjunção]

[04] Se F então (F $\vee$ G). ⎤
[05] Se G então (F $\vee$ G). ⎦ [introdução da disjunção]

[06] Se F e (F $\Rightarrow$ G) então G. [modus ponens] G é deduzido a partir de F e (F $\Rightarrow$ G)

[07] Se $\neg$G e (F $\Rightarrow$ G) então $\neg$F. [modus tollens]

[08] Se (F $\Rightarrow$ G) e (G $\Rightarrow$ H) então (F $\Rightarrow$H). [encadeamento]

[09] Se F e (F $\equiv$ G) então G.

[10] Se G e (F $\equiv$ G) então F.

Algumas regras de inferência. Os nomes das regras estão escritos entre parentisis rectos. F, G e H correspondem a qualquer fórmula, quer da lógica proposicional, quer da lógica de predicados. Por exemplo, as regras [09] e [10] dizem-nos que se, uma determinada fórmula é verdadeira , uma qualquer outra fórmula equivalente também é verdadeira.

Dadas as premissas:

[i] O João acorda.
[ii] O João traz o pano do pó.
[iii] A mãe fica encantada, se o João acorda e limpa o seu quarto.
[iv] Se o Jpão traz o pano do pó, então ele limpa o seu quarto.

Provar por dedução natural a meta: A mãe fica encantada.

Vamos fazer corresponder (atribuir) as diferentes proposições contidas nas premissas aos átomos:

A $\leftarrow$ O João acorda.
B $\leftarrow$ O João traz o pano do pó.
C $\leftarrow$ O João limpa o seu quarto.
D $\leftarrow$ A mãe fica encantada.

A meta a ser atingida (provada) é D.

Podemos então escrever as premissas sob a forma de fórmulas:
[1] A
[2] B
[3] A $\wedge$ C $\Rightarrow$ D
[4] B $\Rightarrow$ C

Destas premissas, podemos deduzir os seguintes consequentes lógicos. As regras de inferência utilizadas constam da fig. 2.7.

[5] C [modus ponens em 2 e 4]
[6] A $\wedge$ C [introdução da conjunção em 1 e 5]
[7] D [modus ponens em 3 e 6]

A meta foi deduzida como sendo uma consequência lógica das premissas. Assim sendo, foi provado que a meta é um teorema. Por outras palavras, a mãe fica encantada.

PROVA DE TEOREMAS POR RESOLUÇÃO
Através de uma regra de inferência chamada RESOLUÇÃO

Só se aplica a fórmulas na forma conjuntiva de cláusulas $C_1 \wedge C_2 \wedge ... \wedge C_n$

CLÁUSULA C_i
Disjunção de literais $A \vee B \vee ... \vee Z$

LITERAL
Átomo ou negação de um átomo A, B, C, D, $\neg A$, $\neg B$, $\neg C$, $\neg D$

A um átomo e sua negação chamamos **literais complementares**; por exemplo B e $\neg B$.

Vejamos alguns exemplos de cláusulas construidas a partir de literais:

$A \vee C \vee \neg E$	
$\neg A \vee D$	
B	Cláusula unitária
$\perp$	Cláusula vazia

CONVERSÃO DE FÓRMULAS NA FORMA NORMAL CONJUNTIVA
$C_1 \wedge C_2 \wedge ... \wedge C_n$

ETAPA	OBJECTIVO A ATINGIR COM A ETAPA	EQUIVALÊNCIA A UTILIZAR NA EXECUÇÃO DA ETAPA
1	Eliminar $\Leftrightarrow$	[1] $(F \Leftrightarrow G) \equiv ((\neg F \vee G) \wedge (\neg G \vee F))$
2	Eliminar $\Rightarrow$	[2] $(F \Rightarrow G) \equiv (\neg F \vee G)$
3	Reduzir o escopo do operador negação, aplicá-lo a pelo menos um átomo	Leis de Morgan: [3a] $\neg(F \vee G) \equiv (\neg F \wedge \neg G)$ [3b] $\neg(F \wedge G) \equiv (\neg F \vee \neg G)$
4	Transformar em conjunções de cláusulas	Distributividade: [4a] $(F \vee (G \wedge H)) \equiv ((F \vee G) \wedge (F \vee H))$ [4b] $((G \wedge H) \vee F) \equiv ((G \vee F) \wedge (H \vee F))$

Exercício:

Converter a fórmula $((C \vee D) \Rightarrow (\neg A \Leftrightarrow B))$ numa conjunção de cláusulas utilizando

A REGRA DE RESOLUÇÃO
A regra de *resolução* pode ser aplicada a um par de cláusulas I1 e I2 quando um literal em I1 e um literal em I2 forem complementares.

Por exemplo, consideremos as seguintes cláusulas como sendo I1 e I2:

(I1) $B \vee F$
(I2) $\neg B \vee H$

B é um átomo
F e H são disjunções de zero ou mais literais; e o número de literais em F é independente do número de literais em H.

Através do princípio da *resolução* podemos deduzir a cláusula I3.

(I3) $F \vee H$

À cláusula I3 chamamos **resolvente**, e diz-se que foi deduzida a partir da **resolução** das cláusulas **pais** I1 e I2, através da **resolução sobre** os literais complementares B e $\neg B$
Isto é, I1 e I2 são as premissas a partir das quais I3 foi deduzido. Os literais no resolvente são a união dos literais das cláusulas pais menos os literais sobre os quais se fez a resolução.
Se ambos os pais forem cláusulas unitarias então o resolvente é a cláusula vazia NIL.

A regra da resolução é uma generalização das regras de inferência modus ponnens e modus tollens

RESOLUÇÃO POR DEDUÇÃO
Suponhamos que nos é dado um conjunto *U* de cláusulas.
Então, através de um procedimento chamado **resolução por dedução**, podemos deduzir a partir de *U* uma sequência de cláusulas resolventes do seguinte modo: deduzir resolventes das cláusulas mutuamente resolúveis de *U*; seguidamente, resolver este resolventes entre si ou com cláusulas de *U* de modo a deduzir mais resolventes; continuar este processo até que se chegue a um critério de paragem previamente estabelecido

Considere-se o conjunto *U* formado pelas seguintes cinco cláusulas:

[I1] $\neg A \vee \neg C$
[I2] $\neg A \vee C \vee D$
[I3] $A \vee D \vee E$
[I4] $\neg D$
[I5] $\neg E$

Alguns dos resolventes que podemos deduzir são:

[I6] $\neg A \vee C$ [deduzido da resolução de I2 com I4]
[I7] $A \vee E$ [resolvendo I3 com I4]
[I8] $D \vee E \vee C$ [resolvendo I3 com I6]
[I9] $C \vee E$ [resolvendo I6 com I7]

... e por aí adiante...
mais resolventes podiam ser deduzidos.

RESOLUÇÃO POR REFUTAÇÃO

G é uma consequência lógica de F_1, F_2, ..., F_n se, e só se, por resolução por dedução podemos deduzir a cláusula vazia $\perp$, a partir do conjunto de cláusulas (chamado conjunto de entrada) derivado de F_1, F_2, ..., F_n, $\neg G$.

Deste modo a resolução é dita **completa**: se G é um teorema, então é garantido que a resolução provará a sua veracidade

PROCEDIMENTO DE RESOLUÇÃO POR REFUTAÇÃO (prova de teoremas por resolução)

Provar que uma fórmula objectivo G é uma consequência lógica das premissas F_1, F_2, ..., F_n.

1.Converter F_1, F_2, ..., F_n e $\neg G$ num conjunto de cláusulas equivalente (disjunção de literais). Este conjunto forma o conjunto de entrada *U* .

2.Selecionar do conjunto duas cláusulas que sejam resoluveis; resolvê-las; e acrescentar o resolvente ao conjunto. Repetir este processo de resolução de cláusulas e acrescentar o resolvente ao conjunto, parando quando o resolvente for a cláusula vazia $\perp$ (nesta situação provou-se que G é um teorema, uma vez que se mostrou que o conjunto de entrada *U* era inconsistente), ou não se conseguir deduzir nenhum novo resolvente (provou-se que G não é um teorema). Uma vez que o conjunto *U* tem um número finito de átomos uma das duas condições de paragem é garantida (recordemos que, a lógica proposicional é resolúvel).

Exercício:

Dadas as premissas:

[i] A
[ii] B
[iii] $A \wedge C \Rightarrow D$
[iv] $B \Rightarrow C$

Provar, utilizando a regra de resolução por refutação, que D é um consequente lógico destas premissas. D é a nossa meta.

OBJECTIVO 2:
Raciocínio Automático Utilizando Lógica de Predicados

Representar os factos

Sócrates é um homem
Platão é um homem
Todo o homem é mortal

Através da lógica proposicional não podemos extrair qualquer relação

entre Sócrates e Platão!
entre ser homem e ser mortal!

Precisámos de VARIÁVEIS e QUANTIFICADORES.

homem(Sócrates)
homem(Platão)
$\forall x$: homem(x) $\rightarrow$ mortal(x)

LÓGICA DE PREDICADOS PRIMEIRA ORDEM

EXTENDE O RACIOCÍNIO AUTOMÁTICO (dedução matemática - possibilidade de deduzir novo conhecimento a partir do actual) PARA ALÉM DA LÓGICA PROPOSICIONAL

É SEMI-DECISÓRIA

_____ // _____

NOMENCLATURA:

DOMÍNIO conjunto de elementos sobre os quais vamos raciocínar identificados por nomes chamados CONSTANTES
ex:

números inteiros	1,2,3...
nomes próprios	António, Maria...
pesos	56.4, 78.0

VARIÁVEIS entidades como u, v, w,z, y às quais podemos atribuir o valor das constantes

FUNÇÔES fazem corresponder um mais elementos (os ARGUMENTOS) do domínio a um elemento desse domínio
ex:
soma(2,3) transforma os argumentos 2 e 3 em 5
mult(3,4) transforma 3 e 4 em 12
sqrt(9) transforma 9 em 3

PREDICADOS transformam um ou mais elementos do domínio em VERDADEIRO ou FALSO
ex:
ÍMPAR(2) é falso
PAR(2) é verdadeiro
MAIOR(soma(2,3), 3) é verdadeiro

os predicados têm a ver com as PROPRIEDADES dos elementos individuais do domínio ou com as RELAÇÔES entre estes

TERMO uma constante é um termo
uma variável é um termo
f(t1, t2, ..., tn) é um termo sse f é uma função de n argumentos e t1,t2,...,tn são termos
ex:
2, 3 e soma(2,3) são termos

ÁTOMO P(t1,t2,...,tn) é um átomo sse P é um predicado de n argumentos e t1,t2,...,tn são termos
ex:
MAIOR(soma(2,3),3)

FÓRMULA um átomo é uma fórmula
se F é uma fórmula, então $\neg F$ é uma fórmula
se F e G são fórmulas, então $(F \wedge G)$, $(F \vee G)$, $(F \rightarrow G)$, $(F \Leftrightarrow G)$ são fórmulas
se F é uma fórmula e x uma variável livre, então $\forall_x F$ e $\exists_x F$ são fórmulas

OPERADORES LÓGICOS (ordem de precedência)

$\neg \ \wedge \ \vee \ \rightarrow \ \Leftrightarrow$

QUANTIFICADORES

$\forall$	universal	"para todo"
$\exists$	existêncial	"existe pelo menos um"

os quantificadores têm prioridade em relação os op. lógicos

//

LÓGICA DE PRIMEIRA ORDEM

só as variáveis podem ser quantificadas, as funções e predicados não podem ser quantificados

//

EXEMPLO: representar os seguintes factos através de fórmulas da lógica de predicados.

1. Marcos era um homem.
2. Marco era Pompeu
3. Todos os Pompeus eram Romanos
4. César era um governador.
5. Todos os Romanos eram leais a César ou odiavam-no.
6. Todos são leais a alguém.
7. As pessoas só tentam assassinar governadores aos quais não são leais.
8. Marcos tentou assassinar César.

1. homem(Marco)
2. Pompeu(Marco)
3. $\forall$x: Pompeu(x) $\rightarrow$ Romano(x)
4. governador(César)
5. $\forall$x: romano(x) $\rightarrow$ leal(x,César) $\vee$ odeia(x, César)
6. $\forall$x,$\exists$y: leal(x,y)

7. $\forall x, \exists y$: pessoa(x) $\wedge$ governador(y) $\wedge$ tentaassassinar(x,y) $\rightarrow \neg$leal(x,y)
8. tentaassassinar(Marco,César)

Vamos tentar provar o teorema raciocinando 'para trás'
BACKWARD CHAINING ou REASONING BACKWARD

Marcos era leal a César?

ou seja provar que

$\neg$leal(Marco,César)

por substituição em 7 passamo a provar

pessoa(Marcos) $\wedge$
governador(César) $\wedge$
tentaassassinar(Marco, César)

por 4

pessoa(Marco) $\wedge$
tentaassassinar(Marco, César)

por 8

pessoa(Marco)

Não conseguimos provar a menos que Marcos seja uma pessoa
se acrescentarmos outro facto:

9. $\forall x$: homem(x) $\rightarrow$ pessoa(x)

Então podemos concluir que Marco não era leal a César!

FUNÇÕES E PREDICADOS COMPUTÁVEIS

gt(t1,t2) Predicado que dá verdadeiro se o termo t1 é superior a t2
evita a escrita de todas as possibilidades!

gt(2,3) = falso
gt(4,1) = verdadeiro

soma(t1,t2) Função que transforma os argumentos t1 e t2 na soma de t1 com t2

gt(soma(2,3),1) = verdadeiro

EXEMPLOS:

1. Marco era um homem homem(Marco)
2. Marco era Pompeu Pompeu(Marco)
3. Marco nasceu no ano 40 nasceu(Marco,40)
4. Todo o homem é mortal $\forall x$: homem(x)$\rightarrow$mortal(x)
5. Todos os Pompeus morreram com a erupção do vulcão no ano 79
   erupção(vulcão,79) $\wedge$ $\forall x$: [pompeu(x)$\rightarrow$morreu(x,79)]
6. Nenhum mortal vive mais do que 150 anos
   $\forall x:\forall t1:\forall t2$: mortal(x) $\wedge$ nasceu(x,t1) $\wedge$ gt(t2-t1,150) $\rightarrow$morreu(x,t2)
7. Agora estamos em 1996
   agora=1996 (equivalência)
8. Homens vivos não morrem
   $\forall x:\forall t$: [vivo(x,t)$\rightarrow\neg$morreu(x,t)]$\wedge$[morreu(x,t)$\rightarrow\neg$ vivo(x,t)]
9. Se alguém morre, então estará morto no tempo seguinte
   $\forall x:\forall t1:\forall t2$: morreu(x,t1) $\wedge$ gt(t2,t1) $\rightarrow$morreu(x,t2)

Provar: Marco está vivo?

$\neg$vivo(Marcos,agora)

9 por substituição

morreu(Marco, agora)

10 por substituição

morreu(Marco,t1) ∧ gt(agora,t1)

5 por substituição

Pompeu(Marco) ∧ gt(agora,79)

2

gt(agora,79)

8 substituição

gt(1996,79)

por computação de gt

nil

RESOLUÇÃO NA LÓGICA DE PREDICADOS

UNIFICAÇÃO DE 2 PREDICADOS

Suponhamos que queremos unificar:

P(x,x) com
P(y,z)

poderão unificar se substituirmos x por y y/x
ficando

P(y,y)
P(y,z)

e substituindo y por z z/y
ou seja efectuamos as substituições

(z/y)(y/x)

Exemplo:

odeia(x,y)
odeia(Marco,z)

podem ser unificados através das seguintes substituições

(Marco/x,z/y)
(Marco/x,y/z)
(Marco/x,César/y,César/z)

ALGORITMO DA UNIFICAÇÃO

Entre L1 e L2
Retorna NIL se unificam sem substituições
FAIL se não unificam
Lista de substituições para unificação

Algoritmo UNIFICAÇÃO(L1,L2)

1. Se L1 e L2 são constantes ou variáveis então
   Se L1 e L2 são idênticos então
      RETURN NIL
   Senão se L1 é variável então
      Se L1 ocorre em L2 então
         RETURN FAIL
      Senão
         RETURN (L2/L1)
   Senão se L2 é variável então
      Se L2 ocorre em L1 então
         RETURN FAIL
      Senão
         RETURN (L1/L2)
   Senão
      RETURN FAIL
2. Se os símbolos predicativos iniciais de L1 e L2 não são idênticos então
      RETURN FAIL
3. Se L1 e L2 têm um número de argumentos diferente então
      RETURN FAIL
4. SUBST = {} Conjunto das substituições
5. Para I=1 até nº de argumentos em L1
   chamar o algoritmo UNIFICAÇÃO com os i-nésimos argumentos de L1 e L2, colocar o resultado em S
   Se S contém FAIL então
      RETURN FAIL
   Se S não é vazio então
      Aplicar S ao restante de L1 e L2
      SUBST=SUBST $\cup$ S
6. RETURN SUBST

CONVERSÃO NA FORMA DE CLÁUSULAS
Forma normal conjuntiva

1. Eliminar $\rightarrow$
   por modus ponens: $a \rightarrow b \equiv \neg a \vee b$
2. Reduzir o escopo do operador $\neg$
   $\neg(\neg a) \equiv a$
   leis de Morgan: $\neg(a \wedge b) \equiv \neg a \vee \neg b$
   $\neg(a \vee b) \equiv \neg a \wedge \neg b$
   $\neg \forall x: P(x) \equiv \exists x: \neg P(x)$
   $\neg \exists x: P(x) \equiv \forall x: \neg P(x)$
3. Cada quantificador deve corresponder a uma única variável
   $\forall x: P(x) \vee \forall x: Q(x)$ converter em $\forall x: P(x) \vee \forall y: Q(y)$
4. Mover os quantificadores para a esquerda da fórmula
   $\forall x: P(x) \vee \forall y: Q(y)$ converter em $\forall x: \forall y: P(x) \vee Q(y)$
5. Eleminar os quantificadores existenciais
   substituir a variável quantificada por $\exists$ por uma função que produza o resultado desejado
   $\exists y:$ Presidente(y) converter em Presidente(S1)
   S1 é uma função sem argumentos que satisfaz Presidente (chamada função de Skolem)
   $\forall x: \exists y:$ pai(y,x) converter em pai(S2(x),x))
   o valor de y que satisfaz Pai depende de x
   deve-se gerar funções com igual número de argumentos que o número de quantificadores universais em cujo escopo ocorram as expressões
6. Suprimir os quantificadores universais, assumindo que todas as variáveis da fórmula estão implicitamente quentificadas universalmente.
7. Converter a fórmula numa conjunção de disjunções
   Prop. associativa: $a \vee (b \vee c) \equiv (a \vee b) \vee c$
   Prop. distributiva: $(a \wedge b) \vee c \equiv (a \vee c) \wedge (b \vee c)$

8. Criar uma cláusula separada para cada conjunção. Para que a fórmula original seja verdadeira todas as cláusulas derivadas têm de o ser.
9. Alterar o nome das variáveis por forma a que duas cláusulas não façam referência à mesma variável.

Exercício: converter na forma normal conjuntiva

1. homem(Marco)
2. Pompeu(Marco)
3. $\forall$x: Pompeu(x) $\rightarrow$ Romano(x)
4. governador(César)
5. $\forall$x: romano(x) $\rightarrow$ leal(x,César) $\vee$ odeia(x, César)
6. $\forall$x,$\exists$y: leal(x,y)
7. $\forall$x,$\forall$y: pessoa(x) $\wedge$ governador(y) $\wedge$ tentaassassinar(x,y) $\rightarrow$ $\neg$leal(x,y)
8. tentaassassinar(Marco,César)

Resultado após aplicar os 9 passos anteriores:
1. homem(Marco)
2. Pompeu(Marco)
3. $\neg$Pompeu(x1) $\vee$ Romano(x1)
4. governador(César)
5. $\neg$romano(x2) $\vee$ leal(x2,César) $\vee$ odeia(x2, César)
6. leal(x3,S1(x3))
7. $\neg$pessoa(x4)$\vee\neg$governador(y1)$\vee\neg$tentaassassinar(x4,y1)$\vee\neg$leal(x4,y1)
8. tentaassassinar(Marco,César)

ALGORITMO DA RESOLUÇÃO (Por refutação)
Dado um conjunto de declarações F (premissas)
provar a declaração P (objectivo)

1. Converter todas as declarações F para a forma normal conjuntiva (cláusulas)
2. Negar P e converter o resultado na forma de cláusula. Adicionar esta às cláusulas obtidas em 1.
3.

Repetir

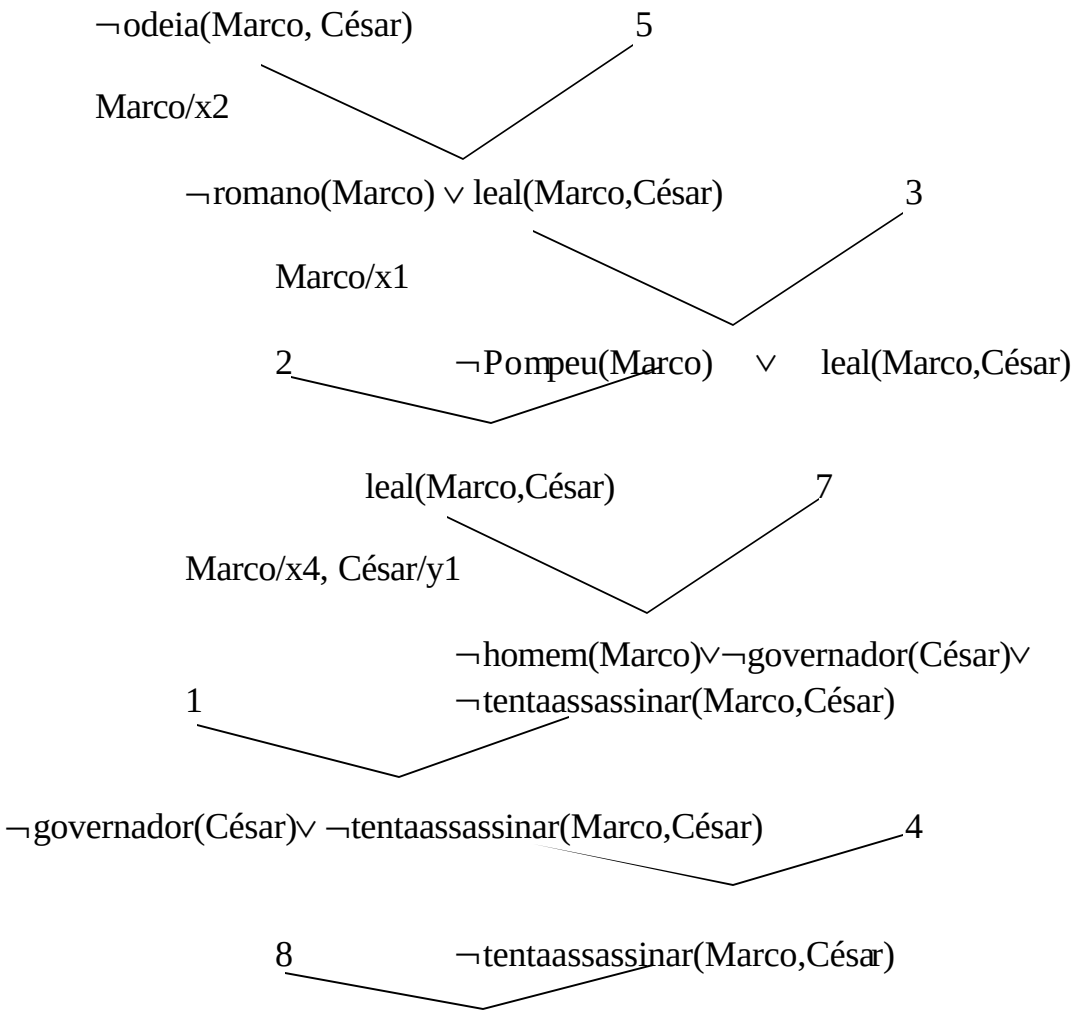
- Seleccionar 2 cláusulas - chamadas pais
- Resolvê-las (aplicar o principio da resolução). O resolvente será a disjunção de todos os literais presentes nas cláusulas pais com as devidas substiuições feitas e com as seguintes excepções:
  - Se uma das cláusulas pai contém um literal T1 e a outra contém um literal $\neg$T2 e se T2 é unificável com T1, então nem T1 nem T2 devem aparecer no resolvente. T2 e T1 chamam-se literais complementares.
  - Utilizar a substituição produzida pela unificação para criar o resolvente.
  - Se existir mais do que um par de literais complementares só um par deve ser omitido no resolvente.
- Se o resolvente é a cláusula vazia
  então
    foi encontrada uma contradição
  senão
    adicionar o resolvente ao conjunto de cláusulas

Até encontrar uma contradição ou progressão impossível ou até ter gasto um volume de esforço pré-determinado (tempo, memória etc.)

ESTRATÉGIAS DE ESCOLHA DE CLÁUSULAS
Para acelarar o processo de prova por resolução
1. Resolver apenas pares de cláusulas que contêm literais complementares
2. Eliminar primeiro as tautologias e as cláusulas facilmente satisfeitas por outras: P $\vee$ Q é satisfeita por P (tautologia).
3. Preferir cláusulas que estejam contidas ou contenham a declaração a provar.
4. Preferir as cláusulas que tenham apenas um literal.
Exemplo: Axiomas na forma de cláusulas (premissas)

1. homem(Marco)
2. Pompeu(Marco)
3. ¬Pompeu(x1) ∨ Romano(x1)
4. governador(César)
5. ¬romano(x2) ∨ leal(x2,César) ∨ odeia(x2, César)
6. leal(x3,S1(x3))
7. ¬homem(x4)∨¬governador(y1)∨¬tentaassassinar(x4,y1)∨¬leal(x4,y1)
8. tentaassassinar(Marco,César)

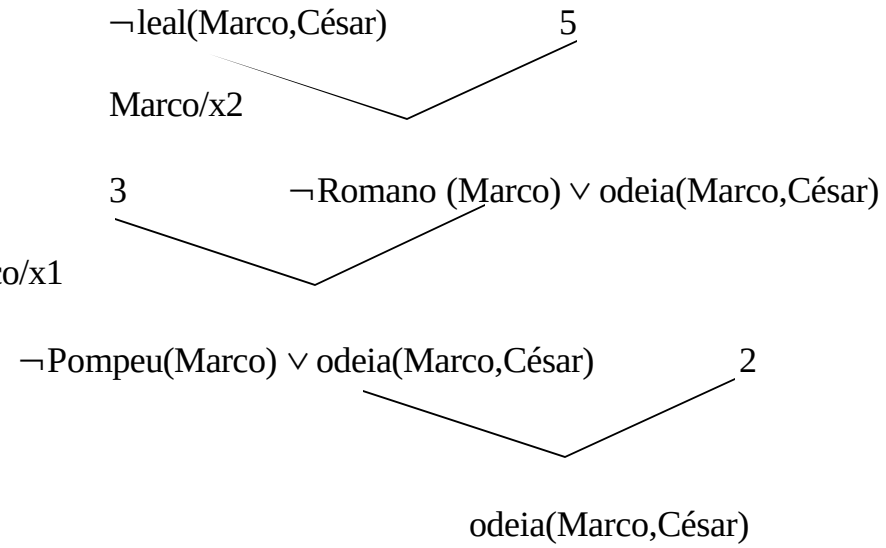
Provar por resolução: odeia(Marco, César)
Devemos acrescentar a negação desta declaração: (prova por refutação)
9. ¬odeia(Marco, César)



(ÁRVORE DE PROVA POR RESOLUÇÃO)
INSUCESSO NA RESOLUÇÃO

Provar: leal(Marco,César)

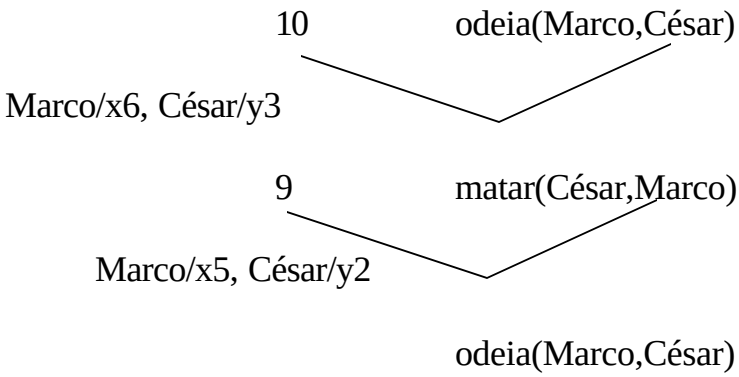
Corresponde a acrescentar a cláusula ¬leal(Marco,César)



Neste ponto não existe nenhuma cláusula com um literal complementar!

Suponhamos que acrescentávamos as seguintes cláusulas:
9. ¬matar(x5,y2) ∨ odiar(y2,x5)
10. ¬odiar(x6,y3) ∨ matar(y3,x6)

podiamos continuar a prova:



Entrámos em ciclo!

RESOLUÇÃO COM SUBSTITUIÇÃO E REDUÇÃO

- Substituição de um valor por outro igual (IGUALDADE)
- Redução de predicados COMPUTÁVEIS quando o seu valor é falso podem ser desprezados (FALSO é o elemento neutro da disjunção)

Exemplo:

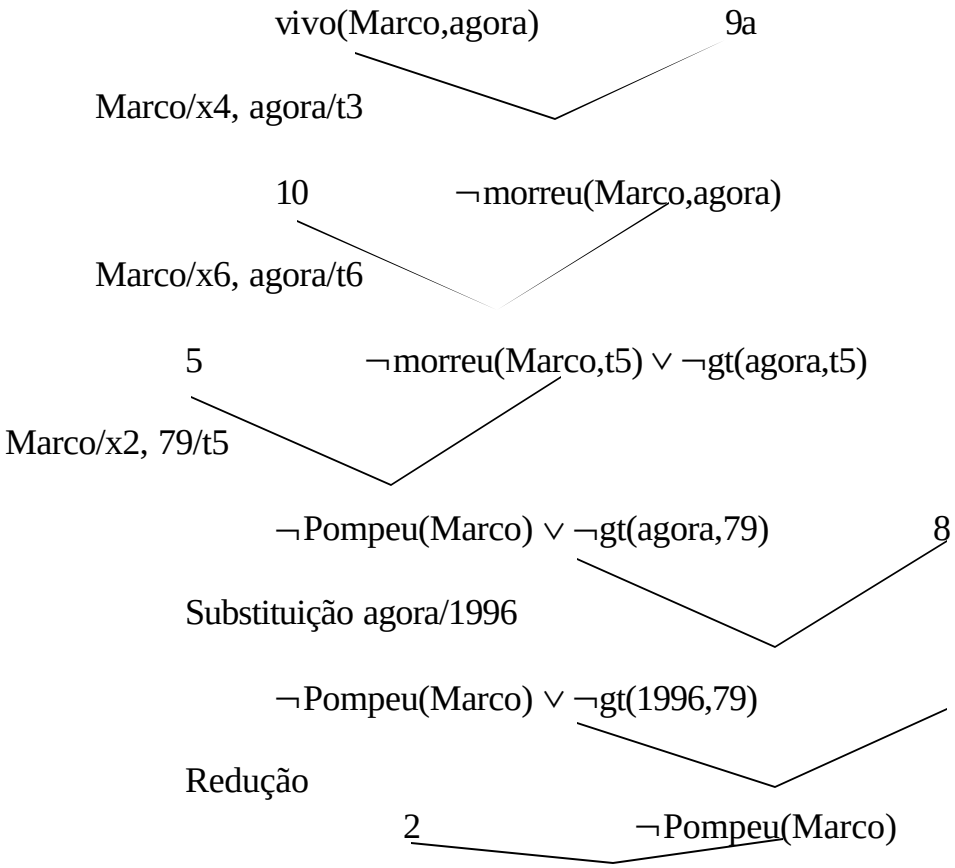
1. homem(Marco)
2. Pompeu(Marco)
3. nasceu(Marco,40)
4. ¬homem(x1) ∨ mortal(x1)
5. ¬pompeu(x2) ∨ morreu(x2,79)
6. erupção(vulcão,79)
7. ¬mortal(x3) ∨ ¬nasceu(x3,t1) ∨ ¬gt(t2-t1,150) ∨ morreu(x3,t2)
8. agora=1996

9a. ¬vivo(x4,t3)∨¬morreu(x4,t3)

9b. morreu(x5,t4) ∨ vivo(x5,t4)

10. ¬morreu(x6,t5) ∨ ¬gt(t6,t5) ∨ morreu(x6,t6)

Provar: ¬vivo(Marco,agora)



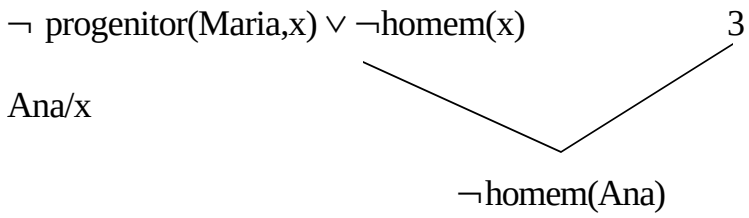
TENTANDO VÁRIAS SUBSTITUIÇÕES

Por vezes pode-se voltar atrás no processo de prova por forma a tentar outras substiruições possíveis.

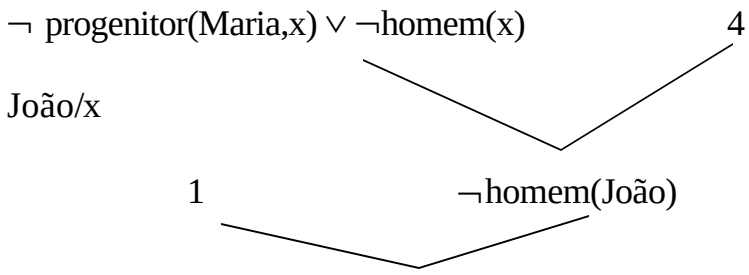
Exemplo:

1. homem(Antonio)
2. homem(João)
3. progenitor(Maria,Ana)
4. progenitor(Maria,João)

Provar: progenitor(Maria,x) $\wedge$ homem(x)



Falha! Então retrocedemos e tentamos outra substituição



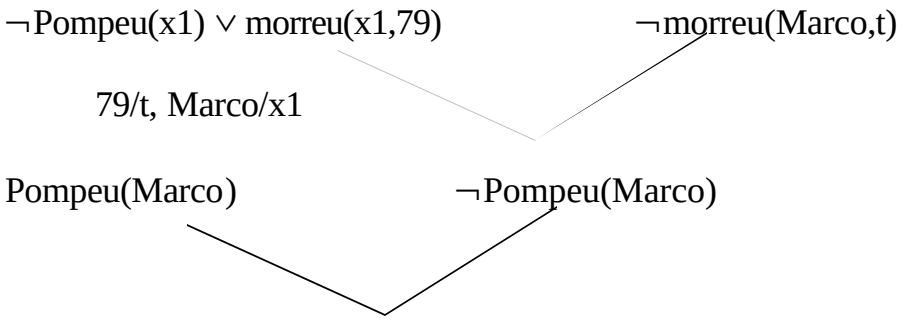
RESPOSTA A QUESTÕES

Em vez de procurarmos a cláusula vazia no processo de resolução por refutação podemos:

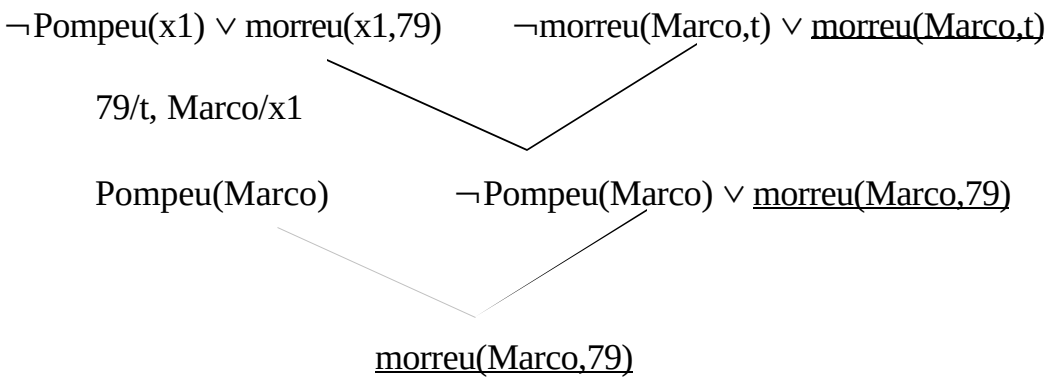
- Acrescentar à expressão objectivo a disjunção da negação da parte da expressão que contém variáveis e cujo valor queremos determinar - expressão espelho
- Procurar chegar à expressão espelho na qual as variáveis estarão instanciadas

Exemplo:

Quando morreu Marco? morreu(Marco,t)



É equivalente a:



t foi instaciada com o valor 79

€

Exercícios Pág 166-169

Artificial Intelligence Second Edition, Elaine Rich e Kevin Knight

OBJECTIVO 3:
UTILIZAÇÃO DA LÓGICA NA ESCRITA DE PROGRAMAS

ESTRATÉGIA SLD
Prova de teoremas por resolução
Combinação de vários refinamentos à resolução (ordenação, supressão e restrição de cláusulas)
Condiciona:

- o tipo de cláusulas
- o modo como as cláusula são construidas
- os literais da cláusula sobre os quais se resolve
- a ordem pela qual se resolve as cláusulas
- o tipo de resolução a utilizar

CLÁUSULAS DE HORN
Cláusula que tem no máximo um literal positivo.
Exemplos:

$\perp$	cláusula vazia
$\neg F1 \vee \neg F2 \vee \neg F3 \vee ... \neg Fn$	cláusula negativa
F	cláusula positiva
$F1 \vee \neg F2 \vee \neg F3 \vee ... \neg Fn$	cláusula mista

Para utilizar SLD deve-se condicionar o conjunto de entrada de forma a conter os seguintes tipos de cláusluas de Horn:

pelo menos uma cláusula positiva	das premissas (pode ser omitida se existirem predicados computáveis)
zero ou mais cláusulas mistas	das premissas
uma única cláusula negativa	o objectivo

COMPONENTES DA SLD:

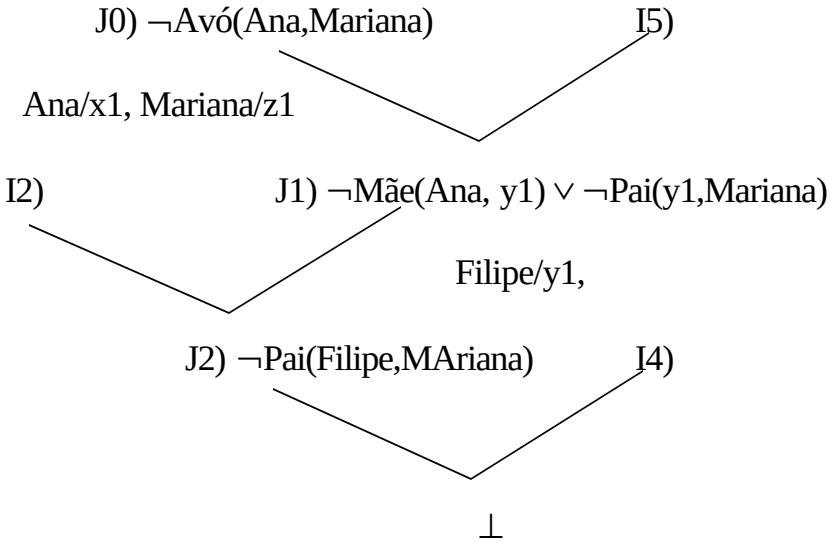
1 Resolver cláusulas em que os literais mais à esquerda são complementarmente unificáveis
2 Seleccionar apenas cláusulas positivas e mistas para a resolução e retroceder (backtracking) sempre que for detectada uma má seleccção

Exemplo: (Primeira componente SLD)
Resolver cláusulas em que os literais mais à esquerda são complementarmente unificáveis

Dadas as premissas
1 Ester é a mãe de Isabel, Ana é mãe do Filipe, e Isabel é mãe da Mariana
2 Filipe é o pai da Mariana
3 Para todo o x, y, z: se x é a mãe de y, e y é o pai ou a mãe de z, então x é avó de z.

Podemos derivar as seguintes CLÁUSULAS DE HORN:
I1) Mãe(Ester, Isabel)
I2) Mãe(Ana, Filipe)
I3) Mãe(Isabel, Mariana)
I4) Pai(Filipe, Mariana)
I5) Avó(x1, z1) $\vee$ $\neg$Mãe(x1,y1) $\vee$ $\neg$Pai(y1, z1)
I6) Avó(x2, z2) $\vee$ $\neg$Mãe(x2,y2) $\vee$ $\neg$Mãe(y2, z2)

Provar que Ana é avó de Mariana (por refutação):



I1, I2... Correspondem às PREMISSAS OU AXIOMAS
J0 Corresponde à negação do OBJECTIVO (TEOREMA A PROVAR)
J1, J2... Correspondem aos RESOLVENTES de ordem 1, 2 ...

SEGUNDA COMPONENTE SLD

- Cláusula definitiva/exacta/regular (definite) é uma cláusula com um único literal positivo
- Todas as cláusulas de Horn mistas e positivas são cláusulas exactas
- As únicas cláusulas exactas no conjunto de entrada são aquelas que derivaram das premissas
- CONSORTE de Jm é uma cláusula exacta que é resoluvel com Jm
- Quando Jm tem apenas uma consorte não é necessário fazer selecção, resolve-se Jm com essa consorte. Mas quando Jm tem mais que uma consorte torna-se necessário fazer uma selecção.
- Uma boa selecção leva-nos para o fim da prova, uma má selecção não.
- Uma vez detectada uma má selecção selecção deve-se fazer o BACKTRACKING e seleccionar uma consorte diferente para resolver com Jm.
- Ao processo de selecção seguido de backtracking chama-se ITERAÇÃO RAÍZ-Jm. Estas iterações repetem-se até que uma das selecções nos encaminhe para o fim da prova, ou por esgotamento das selecções sem terminar a prova.
- O número de iterações raíz-Jm será no máximo igual ao número de consorte de Jm.
- A cada Jm associamos um inteiro chamado c-índice (índice de consorte) que corresponde em cada instante ao número de consortes que falta resolver com Jm. Inicialmente o seu valor é igual ao número de consortes de Jm e vai-se decrementando à medida que fazemos backtracking.
- Diz-se que Jm está num estado activo quando o seu c-índice é superior a zero.

ESTRATÉGIA SLD

1 Resolver Jm com o seu consorte I´k para deduzir Jm+1

2 Decrementar de uma unidade o c-índice de Jm

3 Continuar com a resolução dos literais mais à esquerda das cláusulas começando com as iterações raíz em Jm+1, Jm+2,... até deduzir o resolvente Jm+i (I≥1) tal que:

a) Jm+i é a cláusula vazia (o objectivo foi provado)

b) Jm+i não se pode resolver com outra cláusula exacta ou o literal mais à esquerda de Jm+i é computável e o seu valor igual a verdadeiro. Então percorremos as cláusulas Jm+i-1, Jm+i-2,..., até encontrarmos um estado activo e retrocedemos (BACKTRACKING) para esse ponto. Se não encontrarmos nenhum estado activo até J0 então damos a prova por impossível.

Exemplo: Dadas as premissas:
I1) Mãe(Ester, Isabel)
I2) Mãe(Ana, Filipe)
I3) Mãe(Isabel, Mariana)
I4) Pai(Filipe, Mariana)
I5) Avó(x1, z1) $\vee$ $\neg$Mãe(x1,y1) $\vee$ $\neg$Pai(y1, z1)
I6) Avó(x2, z2) $\vee$ $\neg$Mãe(x2,y2) $\vee$ $\neg$Mãe(y2, z2)

Provar por SLD (por refutação) que Ester é avó de Mariana:

J0) $\neg$Avó(Ester,Mariana)
consortes de J0: I5 e I6
c-índice de J0=2 seleccionamos I5)

Ester/x1, Mariana/z1

J1) $\neg$Mãe(Ester, y1) $\vee$ $\neg$Pai(y1,Mariana)
c-índice de J0=1
consortes de J1: I1
c-índice de J1=1 I1)

Isabel/y1,

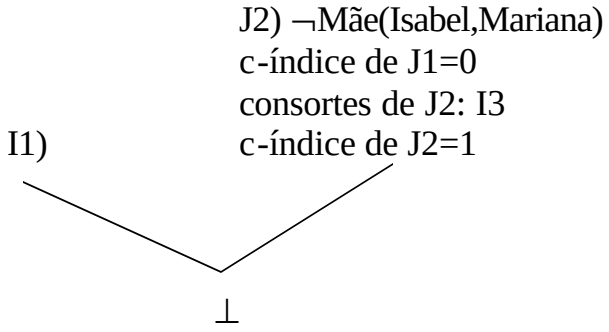
J2) $\neg$Pai(Isabel,MAriana)
c-índice de J1=0
consortes de J2: nenhum temos de retroceder para o estado activo antepassado mais próximo ou seja J0

J0) $\neg$Avó(Ester,Mariana)
consortes de J0: I5(já seleccionado) e I6
c-índice de J0=1 seleccionamos I6)

Ester/x2, Mariana/z2

J1) $\neg$Mãe(Ester, y2) $\vee$ $\neg$Mãe(y2,Mariana)
c-índice de J0=0
consortes de J1: I1
c-índice de J1=1 I1)

Isabel/y2



Exercício: Provar por SLD que ∃v(Avó(v,Mariana))

OBJECTIVO 4:
TÉCNICA LBS
Consideremos a seguinte cláusula

$$\text{LIMPA(João,Quarto)} \vee \neg\text{TRAZ(João, Pano)}$$

'João limpa o quarto ou o João não traz o pano'

Consideremos agora a implicação equivalente:

$$\text{LIMPA(João,Quarto)} \leftarrow \text{TRAZ(João, Pano)}$$

'O João limpa o quarto se traz o pano'

Esta última forma é mais conveniente na escrita de programas em lógica
É mais fácil de entender

A técnica LBS está intimamente ligada à técnica SLD

FORMULAÇÃO DE PREMISSAS E OBJECTIVOS EM LBS

O conjunto de entrada deve ser formado por cláusulas de Horn:

F	positivas (unitárias) chamadas VERDADES ou FACTOS
$F1 \leftarrow F2 \wedge F3 \wedge ... \wedge Fi$	mistas sob a forma de IMPLICAÇÕES equivalentes a $F1 \vee \neg F2 \vee \neg F3 \vee ... \vee \neg Fi$

Estas cláusulas chama-se ASSERÇÕES com as variáveis universalmente quantificadas.
As asserções são derivadas das premissas.

O objectivo a provar corresponde a uma conjunção de átomos com as variáveis quantificadas existencialmente.
Não se nega o objectivo. Exemplos:

1 $\text{ACORDA(João)} \wedge \text{LIMPA(João, Quarto)}$
2 $\exists v\ \text{AVÓ}(v, \text{Isabel})$ representa-se por AVÓ(v, Isabel)

COMPONENTES DA LBS

1 Encadeamento linear para trás (BACKWARD CHAINING) do átomo mais à esquerda
2 Selecção de asserções e BACKTRACKING

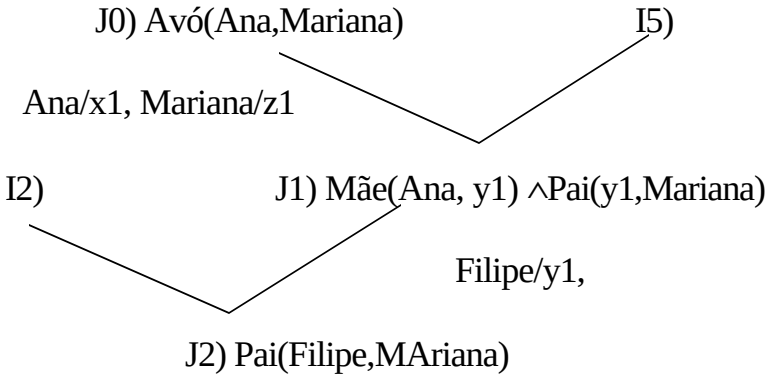
ENCADEAMENTO PARA TRÁS DO ÁTOMO MAIS À ESQUERDA
Se o objectivo a provar é J: $G1 \wedge G2 \wedge G3 \wedge ... \wedge Gj$
e existe uma asserção I: $F1 \leftarrow F2 \wedge F3 \wedge ... \wedge Fi$
em que G1 e F1 são unificáveis através de uma instanciação de variáveis
o novo objectivo a ser provado será:

J': $F2 \wedge F3 \wedge ... \wedge Fi \wedge G2 \wedge G3 \wedge ... \wedge Gj$ com as variáveis substituidas

Definições:
- J é o PAI de J'
- J' é o LIMITE de J
- J é um objectivo ANTEPASSADO de J'
- J' é um objectivo DESCENDENTE de J
- Um átomo é TRIVIAL se o encadeamento para trás por uma asserção produz o limite VAZIO ($\perp$) ou é um literal base (computável) cujo valor calculado é VERDADEIRO.
- Provamos o objectivo J0 se através do encademaneto para trás conseguirmos chegar ao limite vazio.

EXEMPLO:
Dadas as premissas
I1) Mãe(Ester, Isabel)
I2) Mãe(Ana, Filipe)
I3) Mãe(Isabel, Mariana)
I4) Pai(Filipe, Mariana)
I5) Avó(x1, z1) ←Mãe(x1,y1) ∧Pai(y1, z1)
I6) Avó(x2, z2) ←Mãe(x2,y2) ∧Mãe(y2, z2)

Provar que Avó(Ana, Mariana) por encadeamento linear para trás do átomo mais à esquerda



É trivial provar J2 através de I4

SELECÇÃO DE ASSERÇÕES E BACTRACKING
Adimitem-se as mesmas regras da SLD mais:

- Aos Js chamámos limites
- Um objectivo está inerte se não é encadeável para trás através de qualquer asserção ou o seu átomo mais à esquerda é computável e o seu valor FALSO.
- Consortes de Jm são as asserções pelas quais Jm é encadeável para trás

ESTRATÉGIA LBS

1 Encadear Jm para trás com o seu consorte I´k para deduzir Jm+1

2 Decrementar de uma unidade o c-índice de Jm

3 Continuar com o encadeamento para trás dos literais mais à esquerda das cláusulas começando com as iterações raíz em Jm+1, Jm+2,... até deduzir o limite Jm+i (I≥1) tal que:

a) é trivial provar Jm+i (o objectivo foi provado)

b) Jm+i é um limite inerte, não é encadeável para trás através das asserções ou o literal mais à esquerda de Jm+i é computável e o seu valor igual a falso. Então percorremos as cláusulas Jm+i-1, Jm+i-2,..., até encontrarmos um estado activo e retrocedemos (BACKTRACKING) para esse ponto. Se não encontrarmos nenhum estado activo até J0 então damos a prova por impossível.

Exemplo: Dadas as premissas:
I1) Mãe(Ester, Isabel)
I2) Mãe(Ana, Filipe)
I3) Mãe(Isabel, Mariana)
I4) Pai(Filipe, Mariana)
I5) Avó(x1, z1) ←Mãe(x1,y1) ∧Pai(y1, z1)
I6) Avó(x2, z2) ←Mãe(x2,y2) ∧Mãe(y2, z2)

Provar por LBS que Ester é avó de Mariana:

J0) Avó(Ester,Mariana)
consortes de J0: I5 e I6
c-índice de J0=2 seleccionamos I5)

Ester/x1, Mariana/z1

J1) Mãe(Ester, y1) ∧Pai(y1,Mariana)
c-índice de J0=1
consortes de J1: I1
c-índice de J1=1 I1)

Isabel/y1,

J2) Pai(Isabel,MAriana)
c-índice de J1=0
consortes de J2: nenhum temos de retroceder para o estado activo antepassado mais próximo ou seja J0

J0) Avó(Ester,Mariana)
consortes de J0: I5(já seleccionado) e I6
c-índice de J0=1 seleccionamos I6)

Ester/x2, Mariana/z2

J1) Mãe(Ester, y2) ∧Mãe(y2,Mariana)
c-índice de J0=0
consortes de J1: I1
c-índice de J1=1 I1)

Isabel/y2

J2) Mãe(Isabel,Mariana)

c-índice de J1=0
consortes de J2: I3
c-índice de J2=1

É trivial provar J2 com I1

Exercício: Provar por LBS que ∃v(Avó(v,Mariana))

ESCRITA DE PROGRAMAS COM ASSERÇÕES

- Procedimento é uma colecção de asserções com o mesmo fim
- Identificador de uma asserção é o predicado do átomo mais à esquerda na asserção
- Implicação recursiva é quela em o mesmo predicado aparece no consequente e no antecendente
- Procedimento recursivo é aquele que contém uma implicação recursiva
- As asserções devem ser ordenadas da seguinte forma:
  1 juntar as asserções de um mesmo procedimento
  2 colocar factos antes de implicações
  3 colocar as implicações em último lugar num procedimento
  4 os diferentes procedimentos podem estar por qualquer ordem

Exemplo: Escrever o procedimento FAC(x,y) que retorna em y o valor do factorial de x
Pela definição matemática

$$\text{factorial}(x) = \begin{cases} 1 & \text{se } x \leq 1 \\ x * \text{factorial}(x-1) & \text{se } x>1 \end{cases}$$

FAC(x,1) ← LE(x,1)
FAC(x,y) ← GT(x,1) ∧ ASSIGN(x1, sub(x,1)) ∧ FAC(x1,y1) ∧ ASSIGN(y, mult(x,y1))

Predicados computáveis:
LE(x,y) x ≤y GT(x,y) x>y ASSIGN(x,y) x toma o valor de y
Funções:
sub(x,y) x-y mult(x,y) x*y

EXEMPLO: provar FAC(3,y) por LBS

OBJECTIVO 5: LÓGICA VERSUS BASES DE DADOS RELACIONAIS

...

PROPRIEDADES DAS RELAÇÕES

Seja R uma relação binária sobre um domínio D.

R é REFLEXIVA sse $\forall x \in D, \langle x;x \rangle \in D$

Em lógica uma relação r é reflexiva se defenirmos a cláusula:

r(X,X).

Ou

r(X,X) ← t(X).

Exemplo: as pessoas gostam delas próprias

gosta(X,X) ← pessoa(X).
pessoa(bill).
pessoa(kate).

R é SIMÉTRICA sse $\langle x;y \rangle \in D \Rightarrow \langle y;x \rangle \in D$

Em lógica representamos esta propriedade pela cláusula:

r(X,Y) ← r(Y,X).

Exemplo: considere-se o domínio

{sarah, diana, elizabeth, philip, adrew, charles}

a relação “casado com” é simétrica e pode ser descrita como uma base de dados extensional:

casado(sarah,andrew).
casado(diana,charles).

casado(elizabeth,philip).
casado(andrew,sarah).
casado(charles,diana).
casado(philip,elizabeth).

ou mais compactadamente como a base de dados dedutiva:

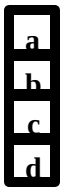
casado(X,Y) ← casado(Y,X).
casado(sarah,andrew).
casado(diana,charles).
casado(elizabeth,philip).

R é ANTI-SIMÉTRICA sse $\langle x;y\rangle \in D$ e $\langle y;x\rangle \in D \Rightarrow x=y$

R é TRANSITIVA sse $\langle x;y\rangle \in D$ e $\langle y;z\rangle \in D \Rightarrow \langle x;z\rangle \in D$

r(X,Z) ← r(X,Y), r(Y,Z).

Exemplo: Considere-se o mundo formado pelos objectos a, b, c e d;



A relação "está em cima de" é transitiva e pode ser definida por uma base de dados puramente extensional:

emcima(a,b).
emcima(a,c).
emcima(a,d).

```
emcima(b,c).
emcima(b,d).
emcima(c,d).
```

ou alternativamente através da base de dados dedutiva:

```
emcima(X,Y) ← emcima(X,Y), emcima(Y,Z).
emcima(a,b).
emcima(b,c).
emcima(c,d).
```

Relação descendente...

R é ASSIMÉTRICA sse $\langle x;y\rangle \in D \Rightarrow \langle y;x\rangle \notin D$

PROBLEMAS DE CICLO INFINITO NAS RELAÇÕES SIMÉTRICAS E TRANSITIVAS:

Qual a resposta à questão ←casado(diana,charles) ?

```
casado(X,Y) ← casado(Y,X).
casado(sarah,andrew).
casado(diana,charles).
casado(elizabeth,philip).
```

Ciclo Infinito. Resolução: passar a regra para último lugar!

```
casado(X,Y) ← casado(Y,X).
casado(sarah,andrew).
casado(diana,charles).
casado(elizabeth,philip).
```

Mas...qual a resposta a ←casado(diana,diana)?

Ciclo Infinito! Resolução: usar antes uma relação auxíliar anti-simétrica

```
casado(X,Y) ← esposa(X,Y).   Estas 2 cláusulas são o fecho
casado(X,Y) ← esposa(Y,X).   simétrico
esposa(sarah,andrew).
esposa(diana,charles).
esposa(elizabeth,philip).
```

Regra geral uma relação simétrica p/2 será definida à custa de uma relação auxiliar q/2 anti-simétrica (FECHO SIMÉTRICO)

```
p(X,Y) ← q(X,Y).
p(X,Y) ← q(Y,X).
q(...)
```

Nas relações TRANSITIVAS, a relação transitiva p/2 será definida através da relação auxíliar q/2 (FECHO TRANSITIVO)

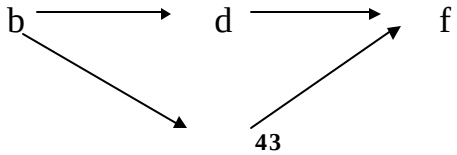
```
p(X,Y) ← q(X,Y).
p(X,Y) ←q(X,Z), p(Z,Y).
```

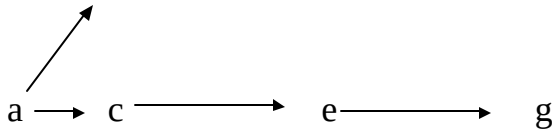
exemplo:

```
emcima(X,Y) ← on(X,Y).                  Estas 2 cláusulas são o
emcima(X,Z) ← on(X,Y), emcima(Y,Z).     fecho transitivo
on(a,b).
on(b,c).
on(c,d).
```

O fecho transitivo pode ser combinado com o fecho reflexivo de uma relação.

Exemplo: considere-se o grafo direccionado (não cíclico)





A relação "caminho" pode ser formalmente definida com uma relação auxíliar assimétrica (ramo/2) que descreve os ramos do grafo:

```
ramo(a,b).
ramo(a,c).
ramo(b,d).
ramo(b,e).
ramo(c,e).
ramo(d,f).
ramo(e,f).
ramo(e,g).
```

O fecho reflexivo e transitivo desta relação é descrito através das cláusluas:

```
caminho(X,X).
caminho(X,Z) ← ramo(X,Y, caminho(Y,Z).
```

HIPÓTESE DO MUNDO FECHADO

Consideremos o seguinte programa lógico (EDB)

```
gosta(joão,maria).
gosta(ana,luis).
gosta(ana,carlos).
```

À pergunta gosta(ana,luis) obtemos a resposta 'yes' i.e. é dedutível.

Mas à pergunta gosta(ana,maria) obtemos a resposta 'no'. O que não quer dizer que ana não gosta de maria. Pura e simplesmente não conseguimos deduzi-lo. Só possuímos conhecimento positivo.

Uma possível resolução era definir no programa de quem a ana não gosta!

Ou então podemos assumir que tudo o que não é dedutível deste programa é falso através da regra de inferência (Meta-regra)

$$\frac{P \nvdash A}{\neg A}$$

CHAMADA HIPÓTESE DO MUNDO FECHADO
CWA(closed world assumption)
A é um "ground atom"

Sempre que uma fórmula A atómica na forma p(a1,...,ak) sem variáveis (ground atom) não é dedutível da Base de Dados então podemos assumir ¬p(a1,...,ak).

Para isso basta acrescentar a meta-regra também chamada negação por falha (negation bay failure)

```
não(P) ← P,!,fail.          em vez de P usar call(P)
não(P).
```

Poderiamos então perguntar: não(gosta(ana,maria))? e a resposta seria 'yes'.

OBJECTIVO 6:
REPRESENTAÇÃO DE CONHECIMENTO ATRAVÉS DE REGRAS

A linguagem das regras se...então também chamadas REGRAS DE PRODUÇÃO é o formalismo mais popular de representar conhecimento (normalmente utilizado nos sistemas periciais).

Exemplos:

se precondição P então conclusão C
se situação S então acção A
se condições C1 e C2 verificam então C não verifica

Vantagens:

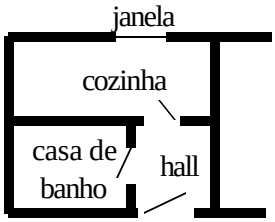
- Forma natural de expressar conhecimento
- Modularidade: cada regra define uma peça do conhecimento
- Incrementabilidade: novas regras podem ser acrescentadas à base de conhecimento independentemente das outras (relativamente)
- Alterabilidade: as regras antigas podem ser aletaradas sem prejuízo para as novas
- Transparência do sistema de suporte (interpretador)
- Facilitam a respostas como:
  Como chegamos a esta conclusão?
  (Questões Como)
  Porque é que estámos interessados nesta informação?
  (Questões Porquê)

Outros exemplos:

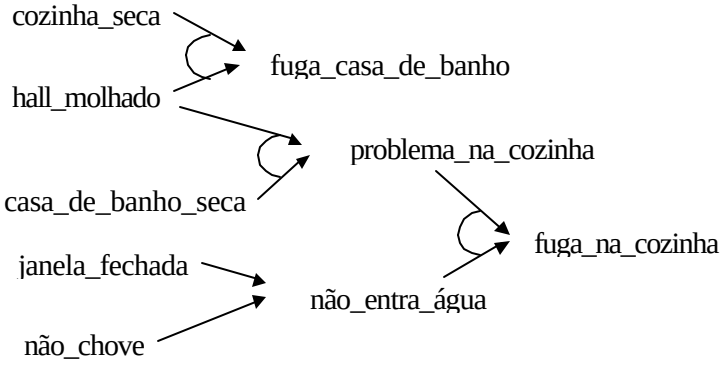
se
1 o paciente tem febre
2 o paciente tem dores
3 o paciente não tem infecções detectáveis
então
diagnosticar gripe com probabilidade (0.7)

Estudo de um caso: uma base de dados para diagnosticar fugas de água num apartamento.

apartamento



rede de inferência



Nodos = proposições
Ramos=regras da BC
Arcos=conjunção entre proposições
Ausência de arco=disjunção
(Grafo AND/OR)

Isto pode ser representado num programa lógico:

se
    hall_molhado e cozinha_seca
então
    fuga_casa_de_banho.

se
    hall_molhado e casa_de_banho_seca
então
    problema_na_cozinha.

se
    janela_fechada ou não_chove
então
    não_entra_água.

se
    problema_na_cozinha e não_entra_água
então
    fuga_na_cozinha.

facto(hall_molhado). Retratam a situação
facto(casa_de_banho_seca).
facto(janela_fechada).

Existem duas formas de raciocinar com regras:

- Backward Chaining (Encaminhamento para trás): a partir de uma hipótese o raciocínio retrocede na rede de inferência até aos factos. Isto é da conclusão tentámos chegar à condição. Do então para o se...
- Forward Chaining (Encominhamento para a frente): a partir de factos o raciocícnio avança pela rede até chegar a uma conclusão (hipótese). Das condições tentámos chegar a uma conclusão. Do se para o então.

INTERPRETADOR PARA REGRAS SE...ENTÃO POR BACKWARD CHAINING

Procedimento demo(T) onde T é a teoria a comprovar.

demo(T) ← facto(T).

demo(T) ← se Condição então T,
demo(Condição).

demo(P1 e P2) ← demo(P1), demo(P2).

demo(P1 ou P2) ← demo(P1).
demo(P1 ou P2) ←demo(P2).

Exemplo: ←demo(fuga_na_cozinha).

INTERPRETADOR PARA REGRAS SE...ENTÃO POR FORWARD CHAINING

Procedimento demo.

```
demo ←      novo_facto(P),
            !,
            write('derivado:'),write(P),nl
            assert(facto(P)),
            demo.
demo ←      write('não há mais factos').

novo_facto(Acção) ←   se Condição então Acção,
                      não(facto(Acção)),
                      facto_composto(Condição).

facto_composto(Cond) ←        facto(Cond).
facto_composto(C1 e C2) ←     facto_composto(C1),
                              facto_composto(C2).
facto_composto(C1 ou C2) ←    facto_composto(C1).
facto_composto(C1 ou C2) ←    facto_composto(C2).
```

Exemplo: ←demo.

```
derivado: problema_na_cozinha
derivado: não_entra_água
derivado: fuga_na_cozinha
não há mais factos
```

GERAÇÃO DE EXPLICAÇÃO

Árvore de prova de como chegámos à conclusão a partir das regras e dos factos da base de conhecimento construída da seguinte forma:

1. se P é um facto então a árvore de prova é P
2. se P foi derivado utilizando a regra:
   se Cond então P

   a árvore de prova é
   P <= Prova_de_Cond

   onde Prova_de_Cond é a árvore de prova da condição Cond
3. se P1 e P2 são proposições cujas árvores de prova são respectivamente Prova1 e Prova2 então se P corresponde a P1 e P2 a árvore de prova é Prova1 e Prova2. Se P é P1 ou P2 então a árvore de prova é Prova1 ou Prova2

INTERPRETADOR COM GERAÇÃO DE PROVAS

procedimento demo(T,P) T é o teorema a provar P é a prova

demo(T, T) ← facto(T).

demo(T, T <=ProvaCond) ← se Condição então T,
demo(Condição, ProvaCond).

demo(P1 e P2, Prova1 e Prova2) ← demo(P1, Prova1), demo(P2, Prova2).

demo(P1 ou P2, Prova) ← demo(P1, Prova).
demo(P1 ou P2, Prova) ←demo(P2, Prova).

Exemplo:

se
    positiva e sem_faltas
então
    passo.

se
    estudei
então
    positiva.

se
    não_estudei
então
    negativa.

facto(estudei).
facto(sem_faltas).

Questão: ←demo(passo,P).

P = passo <= positiva <= estudei e sem _faltas

INCERTEZA

Podemos associar às asserções outra qualificação para além de verdadeiro e falso. Isto pode ser feito através de descrições como:

verdadeiro
altamente provável
provável
pouco provável
falso

Alternativamente, podemos associar às asserções o grau de veracidade, um número real C compreendido entre 0 e 1. Podemos chamar-lhe FACTOR DE CERTEZA, MEDIDA DE VERACIDADE...

Representação:
Para as proposições

Proposição: C.

Para as regras

se Condição então Acção: C.

Como combinar a medida de veracidade das proposições e das regras?

Consideremos duas proposições P1 e P2 com certeza c(P1) e c(P2) respectivamente. Então:

Combinações

c(P1 e P2) = min(c(P1), c(P2))
c(P1 ou P2) = max(c(P1), c(P2))

Regras

se P1 então P2: C c(P2) = c(P1)*C

Exemplo de uma Base de Conhecimento:

se
hall_molhado e casa_banho_seca
então
problema_na_cozinha: 0.9. 90% Certo

hall_molhado:1. 100% Certo
casa_banho_seca:1.
cozinha_seca:0. Falso
não_chove:0.8. 80% Certo
janela_fechada:0.

INTERPRETADOR PARA REGRAS COM INCERTEZA
(Backward Chaining)

Procedimento: demo(Proposição, Certeza)

```
demo(P,Certeza) ←           P: Certeza.

demo(Cond1 e Cond2, Cert) ←  demo(Cond1, Cert1),
                             demo(Cond2, Cert2),
                             min(Cert1, Cert2, Cert).

demo(Cond1 ou Cond2, Cert) ← demo(Cond1, Cert1),
                             demo(Cond2, Cert2),
                             max(Cert1, Cert2, Cert).

demo(P, Certeza) ←           se Cond então P: C1,
                             demo(Cond, C2),
                             Certeza is C1 * C2.
```

Exemplo:

←demo(fuga_na_cozinha,C)

C=? 0.8

Exercício: desenvolver um interpretador que lide com incerteza através de forward chaining.

OBJECTIVO 7:
REPRESENTAÇÃO DE CONHECIMENTO ATRAVÉS DE REDES SEMÂNTICAS

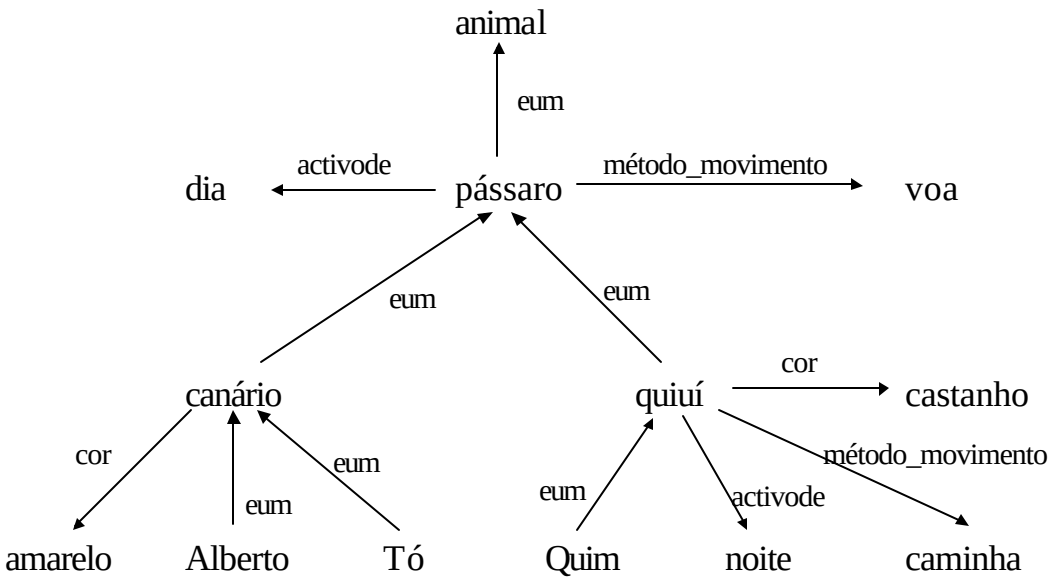
Objectivo:

- Representar, de uma forma estruturada, grandes conjuntos de factos.
- O conjunto de factos é estruturado e por vezes compactado, pois os factos podem ser reconstruídos através de inferência.

Definição:

- Rede de ENTIDADES e relacionamento entre as ENTIDADES.
- Corresponde a um GRAFO.
- Os NODOS correspondem a ENTIDADES.
- Os RAMOS correspondem às RELAÇÕES e são etiquetados com o nome da relação.

Exmplo:



Esta rede representa os seguintes factos:

- Um pássaro é uma espécie de animal.
- O voo é o método de movimentação normal dos pássaros.
- Um canário é um pássaro.
- O Alberto é um canário e o Tó também.

A relação eum:

relaciona uma classe de objectos com uma superclasse
  animal é uma superclasse de pássaro
relaciona uma instância de uma class com a própria classe
  Alberto é uma instância da classe canário

Representação através de cláusulas: BASE DE CONHECIMENTO

```
eum(pássaro, animal).
eum(canário, pássaro).
eum(quiuí,pássaro).
eum(alberto,canário).
eum(tó, canário).
eum(quim, quiuí).
método_movimento(pássaro, voa).
método_movimento(quiuí, caminha).
cor(canário, amarelo).
cor(quiuí, castanho).
activode(pássaro, dia).
activode(quiuí, noite).
```

Como inferir outros factos da rede?
Através da regra de HERANÇA.

Exemplo: qual o modo de movimentação de Tó e Quim?

```
metodo_movimento(X, Método) ←
                    eum(X, SuperX),
                    metodo_movimento(SuperX, Método).
```

Quais as respostas obtidas para:

```
←metodo_movimento(quim,M)
←metodo_movimento(tó,M)
←metodo_movimento(alberto,M)
```

Vamos desenvolver uma regra de herança para cada relação?
Alternativa 1:

```
rel(eum, pássaro, animal).
rel(eum, canário, pássaro).
rel(eum, quiuí,pássaro).
rel(eum, alberto,canário).
rel(eum, tó, canário).
rel(eum, quim, quiuí).
rel(método_movimento, pássaro, voa).
rel(método_movimento, quiuí, caminha).
rel(cor, canário, amarelo).
rel(cor, quiuí, castanho).
rel(activode, pássaro, dia).
rel(activode, quiuí, noite).
```

```
facto(Rel,Arg1,Arg2) ← rel(Rel,Arg1,Arg2), !.
facto(Rel,Arg1,Arg2) ← rel(eum,Arg1,SuperArg),
                       facto(Rel,SuperArg,Arg2).
```

Podemos então colocar questões da forma:

```
← facto(metodo_movimento, quim, M).

M=caminha

← facto(X, tó, dia).

X=activode

← facto(eum, X, pássaro).

X=canário;
X=quiuí;
X=alberto;
X=tó;
X=quim;
no
```

Alternativa 2:

```
facto(F) ← F, !.
```

```
facto(F) ← F=..[Rel, Arg1, Arg2],
           eum(Arg1, SuperArg),
           SuperF=..[Rel, SuperArg, Arg2],
           facto(SuperF).
```

Atenção! O operador =..

functor(Arg1, Arg2,...,Argn)=..[functor,Arg1,Arg2,...,Argn]

Podemos então colocar questões da forma:

← facto(metodo_movimento(quim, M)).

M=caminha

...

OBJECTIVO 8:
REPRESENTAÇÃO DE CONHECIMENTO ATRAVÉS DE ENQUADRAMENTOS (FRAMES)

- Os factos são ASSOCIADOS aos objectos.
- Objecto corresponde a um objecto físico ou um conceito mais abstracto como uma classe de objectos ou até a uma situação.

- A frame é uma estrutura de dados cujos componentes se chamam slots.
- Os slots são identificados por nomes e acomodam informação de vários tipos: valores simples, referência a outras frames e até procedimentos que calculam o valor do slot a partir de outra informação.
- Um slot pode não estar preenchido e ser preenchido através de inferência.
- O mecanismo mais usual de inferência é o da herança.

Exemplo: conhecimento acerca de pássaros

FRAME: pássaro
espéciede: animal (relação eum)
método_movimento: voa
activode: dia

FRAME: canário herda o método_movimento
espéciede: pássaro e activode da frame pássaro
cor: amarelo
tamanho: 20

FRAME:quiuí
espéciede: pássaro
método_movimento: caminha
activode: noite
cor: castanho
tamanho:40

FRAME: alberto
instânciade: canário
tamanho: 15

espéciede corresponde à relação entre uma classe e uma superclasse

instânciade cooresponde à relação entre um membro de uma classe e a classe

Representação das frames como um conjunto de factos da forma:
frame(Frame, Slot, Valor)

Teremos

```
frame(pássaro, espéciede, animal).
    frame(pássaro, método_movimento, voa).
    frame(pássaro, activode, dia).

    frame(canário, espéciede, pássaro).
    frame(canário, cor, amarelo).
    frame(canário, tamanho, 20).

    frame(quiuí, espéciede, pássaro).
    frame(quiuí, cor, amarelo).
    frame(quiuí, tamanho, 40).
    frame(quiuí, método_movimento, caminha).
    frame(quiuí, activode, noite).

    frame(alberto, instânciade, canário).
    frame(alberto, tamanho, 15).

    frame(tó, espéciede, canário).

    frame(animal, tamanho_relativo,
      executa(tamanho_relativo(Objecto, Valor), Objecto, Valor)).


    tamanho_relativo(Objecto, Tamanho) ←
                        valor(Objecto,tamanho,T),
                        valor(Objecto, instânciade, Objclass),
                        valor(Objclass, tamanho, Tclass),
                        Tamanho is T / Tclass * 100.
```

Procedimento para extrair valores dos slots das frames através de herança e com slots calculados:

Directamente:

```
valor(Frame, Slot, Valor) ←
                frame(Frame, Slot, Informação),
                processa(Informação, Frame, Valor).
```

Por herança:

```
valor(Frame, Slot, Valor) ←
                super_frame(Frame, Superframe),
                valor(Superframe, Slot, Valor).
```

Relação *espéciede* e *instânciade*

```
super_frame(Frame,Superframe)←
                        frame(Frame,espéciede, Superframe).
super_frame(Frame,Superframe)←
                        frame(Frame,instânciade, Superframe).
```

Cálculo do valor de um slot:

```
processa(executa(Objectivo, Frame, Valor), Frame, Valor) ←
                        !, Objectivo.
processa(Valor, _, Valor).
```

Exemplos de questões:

← valor(alberto, activode, X)

X= dia

← valor(quim, activode, X)

X= noite

← valor(tó, tamanho_relativo, X)

X= 75 tó mede 75% em relação ao tamanho da classe

OBJECTIVO 9:

PROCURA NUM ESPAÇO DE ESTADOS

Muitos problemas em ciências da computação podem ser formulados como um conjunto S possivelmente infinito de ESTADOS e uma relação de TRANSIÇÃO T ao longo deste espaço de estados.

Dados: um estado inicial $s \in S$
um conjunto de estados finais (objectivos) $G \subseteq S$
Estes problemas consistem em determinar se existe a sequência:

$\langle s;s_1\rangle\langle s_1;s_2\rangle\langle s_2;s_3\rangle \ldots \langle s_{n-1};s_n\rangle \in T^*$ em que $s_n \in G$

Se considerarmos:

ESTADOS	= NODOS num grafo
PARES DA RELAÇÃO DE TRANSIÇÃO	= ARCOS do grafo

Estes problemas reduzem-se a PROCURAR um caminho desde um estado inicial até um dos estados finais.

ALGUNS EXEMPLOS:

- O estado inicial corresponde a um robot e alguns materiais. O objectivo é encontrar um estado em que os materiais formam um produto final.
- Parsing de uma string a. A partir de uma string não-terminal A encontrar um caminho até ao estado final a.
- Também o próprio procedimento SLD pode ser formulado desta forma. O estados são os objectivos e a relação de transição consiste no princípio da resolução. O estado inicial é o objectivo a provar e o estado final a cláusula vazia.
- ...

Representação de um grafo (orientado):

```
arco(a,b).
arco(b,a).
arco(a,c).
arco(b,d).
arco(b,e).
arco(c,e).
arco(d,f).
arco(e,f).
arco(e,g).
```

DESENHAR O GRAFO...

```
final(f).
final(g).
```

Se quisermos representar estados finais

Procedimento caminho/2:

```
caminho(X,X).
caminho(X,Z) ← arco(X,Y), caminho(Y,Z).
```

ou caminho(X,Z) ← final(Z).

Exemplo:

```
←caminho(a,f).
```

Como evitar ciclos infinitos?

```
caminho(X,Z) ← caminho(X,Z, [X]).

caminho(X, X,_).
caminho(X, Z, Visitado) ←   arco(X,Y),
                            nao(membro(Y, Visitado)),
                            caminho(Y, Z, [Y|Visitado]).
```

Exemplo:

```
←caminho(a,f).
```

Como obter o caminho percorrido (estados)?

```
caminho(X,Z,Estados) ← caminho(X,Z,[X],Estados).

caminho(X, X, Visitado,Visitado).
caminho(X, Z, Visitado, Estados) ←
                    arco(X,Y),
                    nao(membro(Y, Visitado)),
                    caminho(Y, Z, [Y|Visitado],Estados).
```

Problema dos Jarros de água

Dados 2 jarros de água: o J4 de 4 litros e o J3 de 3 litros respectivamente. Nenhum possui qualquer marca intermédia. Perto existe uma fonte de água ininterrupta.
Como podemos medir exactamente 2 litros de água no jarro de 4 litros? Inicialmente ambos os jarros estão vazios!

Podemos formular este problema como uma travessia ao longo de um espaço de estados:

ESTADO = par X:Y X é o volume de água no J4
Y é o volume de água no J3
ESTADO INICIAL = 0:0
ESTADO FINAL = 2:_

TRANSFORMAÇÕES ENTRE ESTADOS:

Transformação	Representação por asserções: accao(Estado, NovoEstado)
esvaziar o J4 se não estiver vazio	**accao(X:Y, 0:Y) ¬ X>0.**
esvaziar o J3 se não estiver vazio	**accao(X:Y, X:0) ¬ Y>0.**
encher o J4 se não estiver cheio	**accao(X:Y, 4:Y) ¬ X<4.**
encher o J3 se não estiver cheio	**accao(X:Y, X:3) ¬ Y<3.**
se houver água suficiente no J3 usá-la para completar o J4	**accao(X:Y, 4:Z) ¬ X<4,** **Z is Y - (4-X), Z ³ 0.**
se houver água suficiente no J4 usá-la para completar o J3	**accao(X:Y, Z:3) ¬ Y<3,** **Z is Y - (3-Y), Z ³ 0.**
se couber em J4 verter para lá toda a água do J3	**accao(X:Y, Z:0) ¬ Y>0,** **Z is X+Y, Z £4.**
se couber em J3 verter para lá toda a água do J4	**accao(X:Y, 0:Z) ¬ X>0,** **Z is X+Y, Z £3.**

Procedimento caminho (acíclico) com retorno do caminho:

```
caminho(C) ← caminho(0:0, [0:0], C).          Estado final

caminho(2:_, Visitado, Visitado).
caminho(Estado, Visitado, Caminho) ←
        accao(Estado, NovoEstado),
        nao(membro(NovoEstado, Visitado)),
        caminho(NovoEstado,[NovoEstado|Visitado], Caminho).
```

Exemplo:

```
        ←caminho(X).

        X=[2:0, 0:2, 4:2, 3:3, 3:0, 0:3, 0:0]      ler do fim p/ princípio
```

Exercício: Generalizar para qualquer medida de água nos jarros.

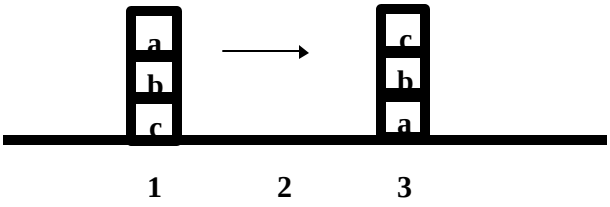
Estados = nodos
Acções = transição entre estados (função de transição)
Caminho = encontrar uma solução

```
caminho(Einicial, Efinal, Caminho) ←
            caminho(Einicial, Efinal, [Einicial],Caminho).

caminho(E,E,V,V).
caminho(E,Ef,V,C) ←   accao(E,NovoE),
                      nao(membro(NovoE,V)),
                      caminho(NovoE,Ef,[NovoE|V],C).
```

O Problema dos blocos:

Considerar uma mesa com 3 posições distintas.
Em cima da mesa existe um determinado número de blocos que podem ser empilhados uns em cima dos outros.
O objectivo é mover os blocos de um estado inicial para um estado final.
Só os blocos livres (no topo) podem ser movidos.



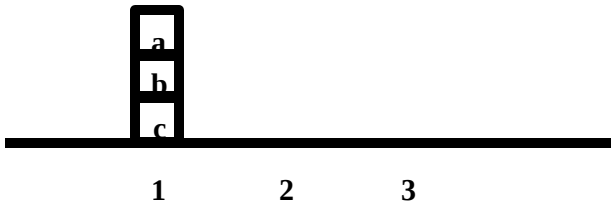
Solução:

o functor ternário *estado/3* representa as 3 posições na mesa
a constante *mesa* representa a mesa
em(X,Y) representa o facto de X estar em cima de Y

Exemplo:

estado(em(a,em(b,em(c,mesa))), mesa, mesa)

representa o estado:



Transformações:

Se a primeira posição não está vazia, o bloco do topo pode ser movido quer para a segunda quer para a terceira posição.

```
accao(estado(em(X,NovoX),AntigoY,Z), estado(NovoX,em(X,AntigoY),Z)).
accao(estado(em(X,NovoX),Y,AntigoZ), estado(NovoX,Y,em(X,AntigoZ))).
```

Se a segunda posição não está vazia, o bloco do topo pode ser movido quer para a primeira quer para a terceira posição.

accao(...)

Se a terceira posição não está vazia, o bloco do topo pode ser movido quer para a primeira quer para a segunda posição.

accao(...)

Exemplo: (utilizar o último programa caminho)

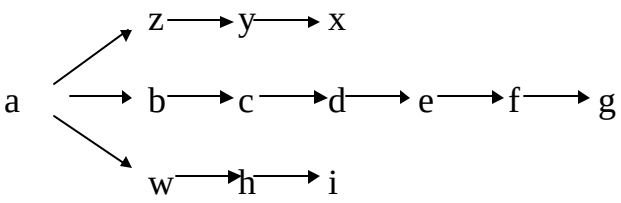
```
←caminho(    estado(em(c,em(b,em(a,mesa))),mesa,mesa),
             estado(mesa,mesa,em(c,em(a,em(b,mesa))), X).

X= [    estado(mesa, mesa, em(c,em,a,em(b,mesa)))),
        estado(mesa,em(c,mesa),em(a,em(b,mesa))),
        estado(em(a,mesa),em(c,mesa),em(b,mesa)),
        estado(em(b,em(a,mesa)),em(c,mesa),mesa),
        estado(em(c,em(b,em(a,mesa))),mesa,mesa)]
```

ESTRATÉGIAS DE PROCURA ALTERNATIVAS

Em relação à procura em PROFUNDIDADE (DEPTH-FIRST) abordada:

para muitos problemas o número de caminhos e de ramificações é de tal ordem que se torna impossível tentar todos os caminhos.



Diferença no esforço para ←caminho(a,h)
e para ←caminho(a,i)

RESOLUÇÃO:

Acrescentar conhecimento heurístico para reduzir o número de caminhos potênciais, ou
Utilizar a estratégia de procura em LARGURA (BREADTH-FIRST)

Para encontrar um caminho num Grafo ou numa Árvore:

Primeiro tentar todos os caminhos de comprimento 1
Depois tentar os caminhos de comprimento 2
...
O processo é repetido até encontrar um caminho desde o estado inicial X até ao estado objectivo (final) Y

```
caminho(X,Y) ← caminho([[X]],Y).          [[X]] é uma lista de
                                          caminhos (listas)
                                          invertidos
caminho([[Y|_] | _ ], Y).
caminho([[X| R] | Rs],Y) ←
                acções(X,[],Adjacentes),
                expande([X|R],Adjacentes,Expandido),
                append(Rs,Expandido,NovoR),
                caminho(NovoR,Y).
caminho([[X|R] | Rs],Y) ←
                caminho(Rs,Y).
```

Acções possíveis a partir de X

```
acções(X,L,Adjacentes) ←        acção(X,NovoX),
                                nao(membro(NovoX,L)),
                                acções(X,[NovoX|L ],Adjacentes).
acções(_,L,L).
```

Expansão do caminho que termina em X para todas as suas ramificações possíveis para um novo estado

```
expande(X,[],[]).
expande(X,[Y|Z],[[Y|X]|W]) ←    expande(X,Z,W).
```

Exemplificar para ←caminho(a,i)

1	caminho([[a]],i)
2	caminho([[z,a], [b,a], [w,a]], i)
3	caminho([[y,z,a], [c,b,a], [h,w,a]], i)
4	caminho([[x,y,z,a], [d,c,b,a], [i,h,w,a]], i)

OBJECTIVO 10:
REPRESENTAÇÃO DE CONHECIMENTO ATRAVÉS DE DEPENDÊNCIAS CONCEPTUAIS E SCRIPTS
(GRAFOS CONCEPTUAIS)

OBJECTIVO:
Representar conhecimento acerca de EVENTOS que está usualmente contido nas frases da linguagem natural.

- Facilita a inferência a partir dessas frases.
- É independente da linguagem utilizada para a construção das frases.

Não representar uma frase através das primitivas correspondentes às palavras presentes na frase

Em vez disso utilizar PRIMITIVAS CONCEPTUAIS que podem ser combinadas para formar os significados das palavras em qualquer linguagem paricular.

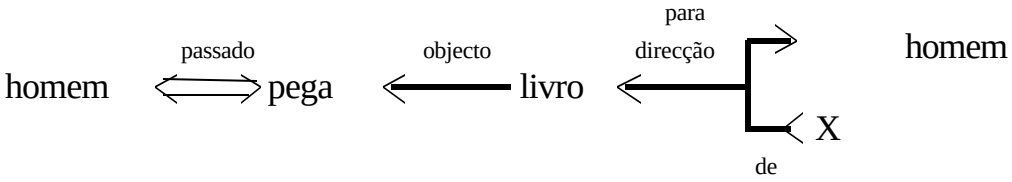
As redes semânticas providenciam estruturas na qual os nodos podem representar informação de qualquer nível.

Os grafos conceptuais providenciam essa estrutura e ainda um conjunto específico de primitivas a um determinado nível de detalhe a partir das quais podemos construir peças de informação.

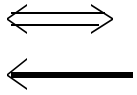
Conseguem-se assim representações com semântica equivalente, de tal forma que as várias frases de uma linguagem que tenham o mesmo significado são representadas por uma única dependência conceptual.

Isto permite o RECONHECIMENTO (PARSING) de SINTAXE MÍNIMA.

EXEMPLO: CONCEITO E DEPENDÊNCIAS para
"O homem pegou num livro"



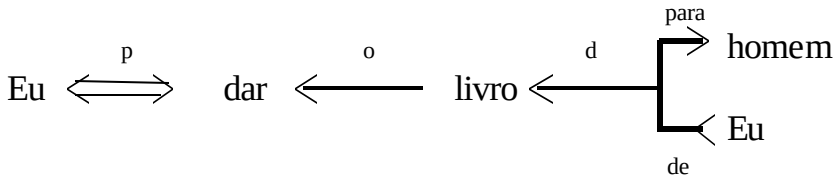
SIGNIFICADO DOS SÍMBOLOS:



ligação nos dois sentidos entre o ACTOR e a ACÇÃO
dependência e o seu sentido

passado	tempo verbal passado	p
objecto	relação objecto	o
direcção	relação recipiente	d

OUTRO EXEMPLO: "Eu dei um livro ao homem"

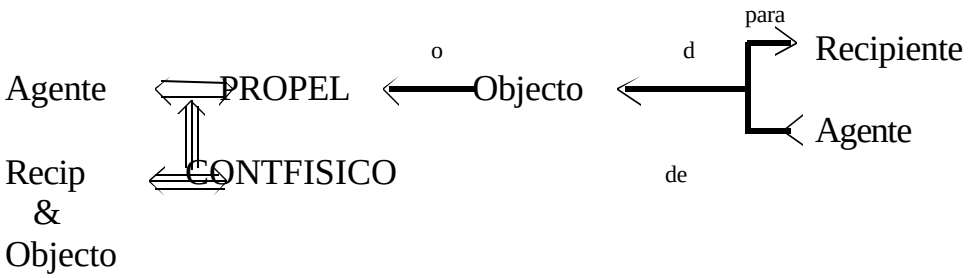


As ACÇÕES são construidas a partir de um conjunto de actos primitivos. Um conjunto possível é o seguinte

Dependência Conceptual	**Significado**	**Exemplos**
atrans	transferência abstracta (posse)	dar, pegar, receber, vender, comprar
ptrans	transferência de lugar	ir, andar, mover, caír
mtrans	transferência mental	ler, dizer, esquecer, ensinar, prometer
ingest	colocar dentro	comer, beber, respirar
propel	aplicar força em	golpear, pontapear, bater
mbuild	Construção mental	realizar, espantar, executar
grasp move	elaborar actos mover parte do corpo	segurar, apanhar pontapé, empurrar
speak	elaborar verbos	dizer, contar, ...
attend	entradas sensoriais	ouvir, olhar
expel	enviar para fora (do corpo)	soprar, cuspir, espirrar

As implicações semânticas de um acto, podem ser embebidas nas entradas de cada verbo (que implica acção) num dicionário.

Exemplo:
Uma entrada no dicionário de Dependências Conceptuais para o acto "golpear"

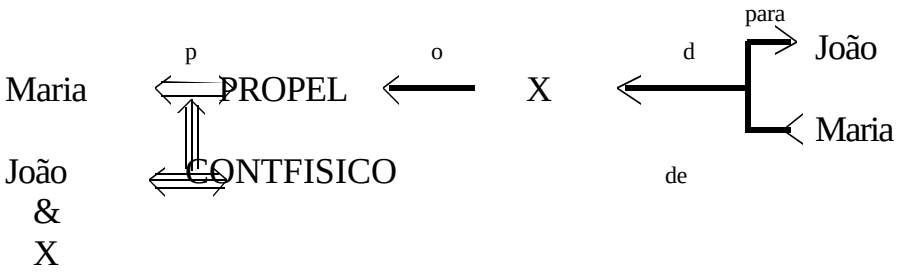


Esta entrada estipula que o Agente do acto está a fazer uma PROPEL (Depedência Conceptual de aplicação de força).

Aqui representa-se uma seta extra (com três linhas) para indicar que uma inferência necessário do verbo golpear como uma PROPEL é que os papeis de Recipiente e Objecto estão em contacto físico (CONTFISICO).

Vejamos então como representar o SIGNIFICADO da frase:

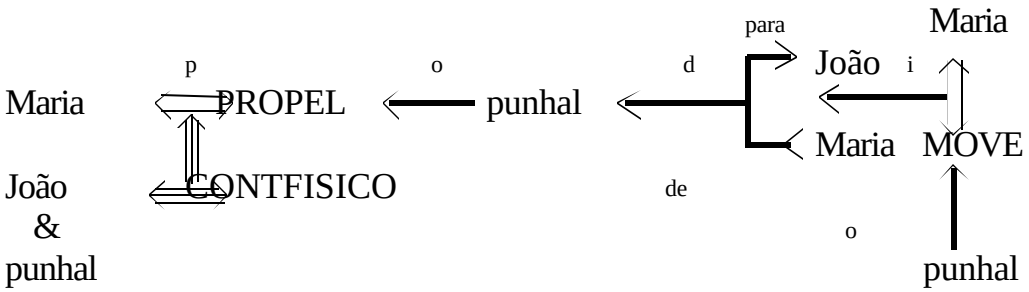
"A Maria golpeou o João"



O objecto utilizado não é específicado na frase por isso usámos X (variável).

Vejamos como representar o SIGNIFICADO da frase similar:

"A Maria apunhalou o João"



i corresponde ao INSTRUMENTO utilizado no acto

O objectivo final é fazer o reconhecimento (PARSING) semântico de frases escritas em linguagem natural para a notação semântica dos GRAFOS CONCEPTUAIS.
De seguida podemos efectuar inferências e acrescentar mais conhecimento ao sistema.
Mas que inferências? e quantas?

OBJECTIVO 11: SCRIPTS

O CONTEXTO indica quais as inferências são razoáveis num certo tipo de situações.
Pode ser representado como uma estrutura de memória humana.

Por exemplo: a história
O José foi à Pizza Hut. E comeu uma margerita. Depois foi ao cinema.
Facilmente podemos fazer as inferências necessárias para responder às questões:

Que tipo de lugar é a Pizza Hut?
Que comeu o José?
A quem pediu o José a comida?
...

Nós fazemos isto porque possuimos conhecimento acerca de situações esterotipadas que:
a) organiza o conhecimento para compreensão de histórias;
b) indica qual o comportamento apropriado;
c) permite-nos descobrir detalhes omitidos.

Este tipo de conhecimento corresponde à estrutura SCRIPT.

As SCRIPTS são um mecanismo acerca de sequências de eventos.

São estruturas que descrevem uma sequência estereotipada de eventos num contexto particular, compostas por SLOTS (similares às FRAMES).

Exemplo: SCRIPT restaurante

evento 1: O Actor vai a um Restaurante
evento 2: O Actor vai da Porta até um Assento
evento 3: O Actor faz o Pedido ao Empregado
evento 4: O Actor come a Comida num Prato
evento 5: O Actor dá dinheiro ao Empregado
evento 6: O Actor deixa o Restaurante

Esta sequência de eventos genéricos é guardada na memória e quando queremos entender uma história específica esta SCRIPT é activada e utilizada para compreendermos a história.

REPRESENTAÇÃO DE CONHECIMENTO
ACTOS SIMPLES (uma DC)
Como representar as Dependências Conceptuais (DC) em lógica de predicados?
Uma vez que a acção é o foco da representação da DC, para um evento podemos adoptar a seguinte notação:

act(N, DC, [Agente, Objecto, De, Para]) para cada evento!

Exemplo: "O João deu o livro à Maria" é representado por

act(N, atrans, [joao, livro, joao, maria]).
Exemplo: "O João foi ao cinema" é representado por

act(N, ptrans, [joao, joao, _, cinema])

HISTÓRIAS (várias DC)

story(N, [evento1, evento2,...])

Exemplo: a história da Pizza-Hut

```
story( test1,
        [   act(A, ptrans, [joao, joao, _, pizza-hut]),
            act(B, ingest, [_, pizza, _, _]),
            act(C, ptrans, [_, _, _ ,_])
        ]).
```

O objectivo do PARSER é detectar os actos, classificá-los segundo uma Dependência Conceptual primitiva e preencher os SLOTS apropriados.

REPRESENTAÇÃO DE SCRIPTS

script(Nome, [Evento1, Evento2,...]).

Exemplo: restaurante

```
script(restaurante,
    [
    act(1, ptrans, [Actor, Actor, LugarX, Restaurante]),
    act(2, ptrans, [Actor, Actor, Porta, Assento]),
    act(3, mtrans, [Actor, Pedido, Actor, Empregado]),
    act(4, ingest, [Actor, Comida, Prato, Actor]),
    act(5, atrans, [Actor, Dinheiro, Actor, Empregado]),
    act(6, ptrans, [Actor, Actor, Restaurante, LugarY])
    ]).
```

COMO ENCONTRAR UMA SCRIPT

Relacionando a cada SCRIPT na memória uma palavra ou palavras. Isto é, uma palavra que quando ocorre numa slot da história sugere a utilização de uma SCRIPT em particular para entender a história.

trigger(Palavra, Script)

Exemplo:

```
trigger(pizza_hut, restaurante).
trigger(empregado, restaurante).
```

EXERCÍCIO: Codificar manualmente as seguintes frases para Dependências Conceptuais.

a) O João caminhou da porta para uma mesa; act(_, ptrans, [joao, joao, porta, mesa])
b) O João caminhou até uma mesa; act(_, ptrans, [joao, joao, _, mesa])
c) O empregado tirou o casaco ao João; act(_, atrans, [empregado, casaco, joao, empregado])
d) O João pediu um bife; act(_, mtrans, [joao, bife, _, _])
e) O João foi para o bar; act(_, ptrans, [joao, joao, _, bar])
f) A Maria bebeu cerveja de uma caneca; act(_, ingest, [maria, cerveja, caneca, maria])
g) O empregado traz a comida da cozinha; act(_, ptrans, [empregado, comida, cozinha, _])

EXERCÍCIO: Representar a seguinte história como uma sequência de actos (Depedências Conceptuais): “Zap foi ao Unirest, o restaurante que fica no fim do universo. Após ter comido um bife, deu uma gorjeta ao empregado e tele-transportou-se para a estação Beta”.

```
story( zap, [   act(1, ptrans, [zap,zap,_, unirest]),
                act(2, ingest, [zap, bife, _, _]),
                act(3, atrans, [_, gorjeta, _, _]),
                act(4, ptrans, [_, _, _, beta])
            ]).
```

PREDICADOS PRINCIPAIS DE INFERÊNCIA

demo(História, Interp)
- História = nome de uma história na BD
- Interp = como a história foi entendida (interpretada)

demo(História, Interp) ←

story(História, Actos),	procura a história História na BD e indica o conjunto de eventos Actos
infere(Actos, Interp).	dá uma interpretação Interp para os eventos Actos

infere(Actos, Interp) ←

encontra(Actos, NomeScript),	encontra uma Script com o cujo nome esteja contido numa slot de um acto
script(NomeScript, Interp),	encontra na BD essa Script
unifica(Actos, Interp).	unifica os actos da história com os actos da script

encontra(História, NomeScript) ← membro(act(_,_, Slots), História),
membro(Palavra, Slots),
nonvar(Palavra),
trigger(Palavra, NomeScript).

unifica([], _).
unifica([Ln| Rhistória], [Ln|Rscript]) ← unifica(Rhistória, Rscript).
unifica(História, [_|Rscript]) ← unifica(História, Rscript).

Exemplo: act(A, ptrans, [joao, joao, _, pizza_hut])

unifica com: act(1, ptrans, [Actor, Actor, LugarX, Restaurante])

com as variáveis instanciadas da seguinte forma:

A = 1 Actor = joao LugarX = _ Restaurante = pizza_hut

Exemplo:

← demo(teste1, I).

A partir de uma história como

"O João foi à Pizza Hut, comeu uma pizza e saíu"

Pretende-se gerar interpretações do género:

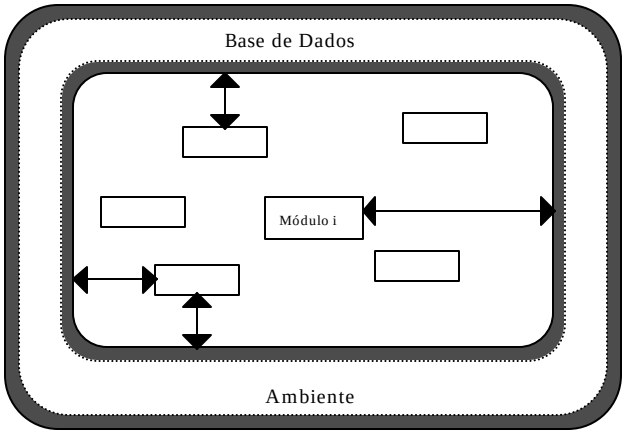
o joao foi do lugar 1 para a pizza-hut
o joao foi da porta para uma mesa
o joao fez o pedido ao empregado
o empregado trouxe a pizza da cozinha para a mesa
o joao comeu a pizza
o joao deu dinheiro ao empregado
o joao foi da pizza-hut para o lugar 2

OBJECTIVO 12: **PROGRAMAÇÃO ORIENTADA POR PADRÕES**

- Os sistemas orientados por padrões correspondem a uma arquitectura para programar sistemas.
- Um ***programa orientado para padrões*** (POPP) é uma colecção de módulos OPP. Cada módulo é definido por:

  1) um padrão que constitui a pré-condição, e
  2) uma acção a ser executada se os dados do ambiente unificam com o padrão.
- A esses módulos passaremos a chamar ***módulos orientados para padrões*** (módulos OPP).
- Na organização convencional, os módulos de um sistema invocam-se uns aos outros de acordo com um esquema pré-definido. Cada módulo decide sobre quais os módulo devem ser executados a seguir invocando explicitamente esses módulos. O fluxo de execução associado a este mecanismo é sequêncial e deterministico.
- Em contraste, na organização orientada para padrões os módulos do sistema não são invocados directamente por outros módulos. Em vez disso, eles são invocados por padrões que ocorrem no seu ambiente de dados.

A execução dos módulos é despelotada pelos padrões que ocorrem no ambiente do sistema. O ambiente de dados é normalmente chamada a ***base de dados***.



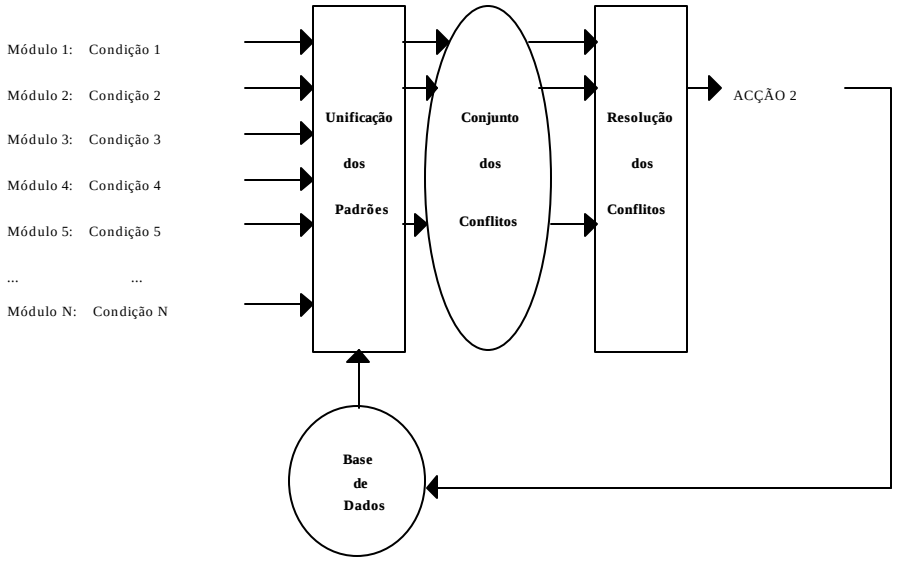
O elevado nível de modularidade é especialmente desejável em sistemas com bases de conhecimento complexas porque é difícil prevêr com antecedência todas as interacções entre cada uma das peças elementares de informação na base. A arquitectura OPP oferece uma solução natural para este tipo de problemas: cada peça de conhecimento, representada por uma regra se-então, pode ser vista como um módulo OPP.

VANTAGENS

- Não existe qualquer hierarquia entre os módulos, e não existe a indicação explicita acerca de que módulo pode invocar que módulo.
- Os módulos comunicam com a base de dados e não entre eles directamente. Permite a execução paralela de vários módulos, uma vez que o estado da base de dados pode satisfazer simultaneamente várias pré-condições.
- Modelo natural de computação paralela na qual cada módulo pode ser implementado fisicamente pelo seu próprio processador.
- Cada módulo pode ser desenhado e implementado de uma forma relativamente autónoma.
- O sistema pode resistir à remoção/inserção/modificação de alguns módulos, sem que isso lhe seja necessariamente fatal.

CICLO DE VIDA DOS SISTEMAS OPP

1)Unificação dos padrões: Encontrar na base de dados todas as ocorrências dos padrões (pré-condições) dos módulos. Constituindo um conjunto de conflitos.
2)Resolução do conflito:Seleccionar um dos módulos do conjunto de conflitos.
3)Execução: Executar o módulo seleccionado na etapa 2.



Os sistemas OPP também podem ser vistos como um estilo particular de escrever programas e de pensar acerca dos problemas, chamada programação OPP.

ESCRITA DE UM PROGRAMA OPP (EXEMPLO)

Consideremos um exercício elementar de programação: calcular o maior divisor comum (mdc) D de dois números inteiros A e B, através do algoritmo de Euclides:

```
⌈enquanto A ≠ B fazer
|     ⌈se A > B então
|     |          { A ← A - B
{     {       senão
|     |          { B ← B - A
⌊D ← A
```

Podemos definir o mesmo processo através de dois módulos OPP:

Módulo 1

Condição Existem dois números X e Y na base de dados tal que X > Y.

Acção Substituir X na base de dados pela diferença X - Y.

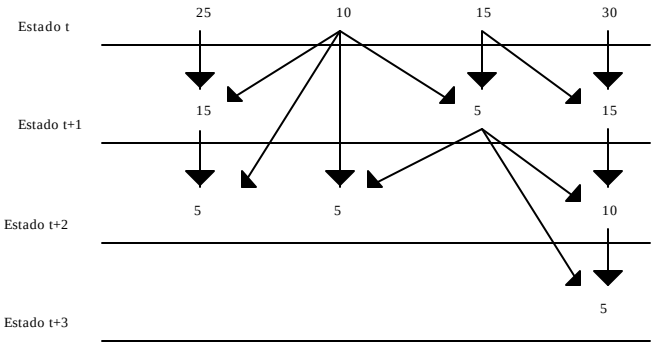
Módulo 2

Condição Existe um número X na base de dados.

Acção Apresentar o valor de X e abandonar.

Este programa OPP é capaz de calcular o maior divisor comum de um conjunto qualquer de números inteiros. Se colocarmos n números inteiros na base de dados o sistema calculará o mdc de todos eles.

Possivel sequência de estados e transformações pelos quais a base de dados passa para a obtenção do mdc dos quatro números: 25, 10, 15, 30.



META-INTERPRETADOR DE PROGRAMAS OPP

Adoptemos a seguinte sintaxe para especificar os módulos OPP:

Condições ---> Acções

Condições correspondem a uma lista de condições da forma

[Condição1, Condição2, Condição3, ...]

onde Condição1, Condição2, etc. são meros objectivos (goals) do Prolog. A pré-condição é satisfeita se todos os objectivos na lista forem satisfeitos. Acções corresponde a uma lista de acções:

[Acção1, Acção2, ...]

Cada acção é, também um mero objectivo do Prolog. Para executar uma lista de acções, todas as acções na lista têm de ser executadas. Isto é, todos os objectivos correspondentes terão de ser satisfeitos. Nas acções disponíveis terão de haver algumas que manipulem a base de dados: inserir, retirar ou substituir objectos na base de dados. A acção 'stop' finaliza a execução do meta-interpretador.

```
% Regras de produção para encontrar o mdc (algoritmo de Euclides)

:- op( 300, fx, numero).

[ numero X, numero Y, X > Y ]  --->  [ NovoX is X - Y,
                                       substitui( numero X, numero NovoX)
                                       ].

[ numero X ]  --->  [ write(X), stop ].

% Base de Dados inicial
```

numero 25.

numero 10.

numero 15.

numero 30.

META-INTERPRETADOR SIMPLES PARA PROGRAMAS OPP

A execução do meta-interpretador é iniciada pelo objectivo: ?- demo(T).
T corresponde à teoria a ser demonstrada, neste caso o conjunto de módulos OPP.

```
% Meta-interpretador de programas orientados para padrões
% A base de dados do sistema é manipulada pelas acções: insere, retira e substitui
% A negação de uma condição é feita pelo operador ~
% Declaração de operadores
:- op( 800, xfx, ---> ).
:- op(600, fx, ~ ).
demo( T ) :-
        consult( T ),
        run.
% run: executa os módulos OPP até que seja encontrada a acção 'stop'
run:-
        Condicao ---> Accao,    % Regra de produção
        testa( Condicao ),      % A pré-condição é satisfeita?
        executa( Accao ).       % Se sim executa a acção.
% testa([ Condicao1, Condicao2, ... ]) se todas as condições forem verdadeiras
testa( [] ).                            % Condição vazia
testa( [~Primeira | Resto ]) :-         % Negação de uma condição
                !,
                nao( Primeira ),
                testa( Resto ).
testa( [Primeira | Resto ]) :-          % Conjunção de condições
                !,
                call( Primeira ),
                testa( Resto ).
nao( Condicao ) :-                      % Verdadeiro se Condicao é falsa
                call( Condicao ),
                !,
                fail.
nao( _ ).
% executa( [ Accao1, Accao2, ... ] )  executa uma lista de acções
executa( [ stop ] ) :-    !.  % paragem do meta-interpretador
executa( [] ) :-            % Acção vazia
                demo.           % continua com a próxima regra de produção -
executa( [ Primeira | Resto ] ) :-
                call( Primeira ),
```

executa(Resto).

```
% Predicados de manipulação da base de dados
% substitui( A, B ) substitui o objecto A pelo objecto B
substitui( A, B ) :-
        retract( A ),
        !,
        asserta( B ).
% insere( A ) insere o objecto A
insere( A ) :-
        asserta( A ).
% retira( A ) retira o objecto A
retira( A ) :-
        retract( A ).
```

REFINAMENTOS

PARTES NEGATIVAS	MELHORAMENTO POSSÍVEL
A resolução de conflitos é reduzida e fixada à ordem pré-definida das regras de produção.	Inclusão de um módulo de controlo para possibilitar outro tipo de resolução de conflitos.
Quando a base de dados tiver um tamanho considerável e o número de regras de produção for grande, o processo de unificação torna-se extremamente ineficiente.	Alterar a organização da base de dados. Por exemplo através da indexação da informação, ou partição da informação em várias sub-bases, ou distribuindo as regras de produção em sub-conjuntos. A ideia de partição é tornar acessível apenas um sub-conjunto da base de dados e/ou das regras de produção em cada instante de tempo,.
O meta-interpretador, como está programado, não possibilita o retrocesso (backtracking) dada a forma de manipulação da base de dados. Não podemos assim explorar caminhos de execução alternativos.	Implementação alterada da base de dados, sem utilizar assert e retract. Uma possibilidade é representar o estado da base de dados por um termo do Prolog passado ao predicado demo. A forma mais simples deste termo corresponde à lista dos objectos da base de dados.

OBJECTIVO 13: REPRESENTAÇÃO DE INFORMAÇÃO INCOMPLETA - BASES DE CONHECIMENTO INCOMPLETAS

REPRESENTAÇÃO DE VÁRIAS FORMAS DE VALORES NULOS:

Colecção de técnicas usadas para distinguir entre valores de atributos CONHECIDOS e DESCONHECIDOS numa base de dados lógica.

EXEMPLO 1: Distribuição do serviço docente de uma Universidade

Consideremos um departamento com 2 professores, o Miguel e o João.

Professor
miguel
joão

Departamento oferece 4 cursos diferentes: Pascal, Fortran, Prolog e Lisp.

Curso
pascal
fortran
prolog
lisp

A tabela seguinte representa a informação completa sobre a distribuição para um curso de verão:

Professor	Curso
miguel	pascal
joão	fortran

A informação das tabelas pode então ser representada pelo programa lógico:

professor(miguel).
professor(joão).
¬professor(X)←não(professor(X)). Pressuposto do mundo fechado

curso(pascal).
curso(fortran).
curso(prolog).
curso(lisp).
¬curso(X)←não(curso(X)). Pressuposto do mundo fechado

ensina(miguel, pascal).
ensina(joão, fortran).
¬ensina(X,Y) ← não(ensina(X,Y)). Pressuposto do mundo fechado

permite que o programa conclua correctamente por exemplo :
¬ensina(miguel, fortran)

A interpretação de questões (possível):

demo(P,verdadeiro) ← P.
demo(P,falso) ← ¬P.

exemplos:

←demo(professor(joão),X)?
←demo(professor(manuel),X)?
←demo(ensina(joão,fortran),X)?
←demo(ensina(joão,prolog),X)?
←demo(¬ensina(joão,pascal),X)?

VALORES NULOS DO TIPO DESCONHECIDO
A situação altera-se se a tabela anterior for alterada para:

Professor	Curso
miguel	pascal
joão	fortran
docente	lisp

Onde *docente* é um VALOR NULO, correspondente a um professor DESCONHECIDO (possivelmente diferente de miguel e joão)

Perante esta informação o sistema de inferência deverá dar as seguintes respostas:

Pergunta	Resposta
←ensina(X,fortran)	X=joão
←ensina(X,lisp)	X é desconhecido
←ensina(miguel,fortran)	não
←ensina(miguel,lisp)	desconhecido

Como representar esta tabela?
=>FORMALIZAÇÃO APROPRIADA DO PRESSUPOSTO DO MUNDO FECHADO PARA LINGUAGENS COM VALORES NULOS ATRAVÉS DA REPRESENTAÇÃO DE ANORMALIDADES (predicado ab)

Substituir a regra ¬ensina(X,Y) ← não(ensina(X,Y)) pelas regras:

¬ensina(X,Y) ← não(ensina(X,Y)), não(ab(X,Y)).
ab(X,Y) ← ensina(docente, Y), professor(X).

Interpretação de questões:

demo(P,verdadeiro) ← P.
demo(P,falso) ← ¬P.
demo(P,desconhecido) ← não(P), não(¬P).

VALORES NULOS DO TIPO ENUMERADO

Consideremos agora a tabela:

Professor	Curso
miguel	pascal
joão	fortran
{miguel, joão}	prolog
docente	lisp

NULO DO TIPO ENUMERADO - Valor desconhecido mas um de um conjunto finito de valores (miguel ou joão)

Para representar esta informação temos de expandir o programa anterior com as anormalidades:

ab(miguel,prolog).
ab(joão, prolog).

Resposta a questões:

←demo(ensina(António,prolog),X)?
←demo(ensina(joão,prolog),X)?
←demo(ensina(miguel,prolog),X)?

VALORES NULOS DO TIPO NÃO PERMITIDO

Consideremos a tabela com os empregados de uma empresa

Executivo	Empregado	Salário
joão	joão	ω
	miguel	200000

NULO DO TIPO NÃO PERMITIDO

Representaçãoda informação:

executivo(joão).
empregado(joão).
empregado(miguel).
salário(joão, ω).
salário(miguel, 200000).
¬executivo(X) ← não(executivo(X)).
¬salário(E,S) ← não(salário (E,S)), não(ab(E)).
ab(E)←executivo(E).

Às questões	o sistema deverá responder
←salário(miguel,100000)	no
←salário(joão,300000)	desconhecido

COMO MANTER A CONSISTÊNCIA DA INFORMAÇÃO?
Não permitir a assimilação de informação contraditória!
O sistema não deverá permitir a inserção de cláusulas como:

salário(joão,400000)

DEFINIÇÃO DE INVARIANTES

nulo(ω). Valor especial

invariantes (têm de se verificar sempre na base de conhecimento)

inv←¬(executivo(V), salário(V,X), não(nulo(X))).

Inserção de cláusulas:

insere(P) ← assert(P), não(inv).
insere(P) ← retract(P).

Exercício:

Considerar o sistema de conhecimento que serve de suporte ao processo de atribuição de licenças a barcos para pesca em águas comunitárias:

Relação: barcos

Barco	Nacionalidade	Tonelagem	Autonomia (dias)
25	Russa	2500	30
75	Americana	1900	ω
45	Espanhola	2600	40

Relação: pesca

Barco	Captura	Peso (Kg)
25	sargo	{2000,2500}
75	*peixe*	3000
45	carapau	1500

onde: ω é um nulo do tipo não permitido.
peixe é um nulo do tipo desconhecido.

a) Representar este conhecimento.
b) Desenvolver o sistema de inferência.
c) Impôr os seguintes invariantes:
   1. nunca são admitidos barcos de nacionalidade mongol
   2. não podem ser removidos barcos sem remover primeiro as referências na tabela pesca
d) elaborar um procedimento para inserção de barcos

Resolução:
a) Representação do conhecimento (incompleto)

barcos(25,russa,2500).
barcos(75,americana,1900).
barcos(45,espanhola,2600).
¬barcos(X,Y,Z) ← não(barcos(X,Y,Z)).

autonomia(25,30).
autonomia(75,ω).
autonomia(45,40).
¬autonomia(X,Y) ← não(autonomia(X,Y)), não(ab_{auton}(X)).
ab_{auton}(75).

captura(25,sargo).
captura(45,carapau).
¬captura(X,Y)←não(captura(X,Y)), não(ab_{cap}(X,Y)).
ab_{cap}(75,peixe).

peso(75,3000).
peso(45,1500).
¬peso(X,Y)←não(peso(X,Y)), não(ab_{peso}(X,Y)).
ab_{peso}(25,2000).
ab_{peso}(25,2500).

b) Sistema de inferência
demo(P,verdadeiro) ← P.
demo(P,falso) ← ¬P.
demo(P,desconhecido) ← não(P), não(¬P).

c) Invariantes
1. ∀B,T: ¬barcos(B,mongol,T)

inv(barcos(B,N,T)) ← não(solucoes(B, barcos(B,mongol,T),[])).

2. ∀B,T,A,C,P: ¬barcos(B,mongol,T) ∧ ¬autonomia(B,A) ∧ ¬captura(B,C)
∧ ¬peso(B,P)

inv(barcos(B,N,T)) ← não(solucoes(B, [barcos(B,N,T),autonomia(B,A),
captura(B,C), peso(B,P)], [])).

d) Inserção

inserir(B,N,T,A) ← assert(barcos(B,N,T)), assert(autonomia(B,A)),
não(inv(barcos(B,N,T))).

inserir(B,N,T,A) ← retract(barcos(B,N,T)), retract(autonomia(B,A)).