Mestrado Integrado Engenharia Biomédica

Universidade do Minho

Caderno das Aulas

Representação de Conhecimento

Docentes:

José Neves

José Machado

Realizado por:

Ana Jorge Leitão nº55505

Domingos Assunção nº55547

Joel Braga nº55572

Janeiro de 2012

Índice

Introdução

Este trabalho, realizado no âmbito da disciplina de Representação de Conhecimento, tem como objectivo mostrar e explicar os conceitos abordados durante as aulas práticas e teóricas. Cada tema abordado durante o semestre será explicitado recorrendo-se aos exemplos e exercícios feitos durante as aulas.

Serão em primeiro lugar apresentados os conteúdos dados nas aulas teóricas onde foram adquiridos conhecimentos sobre Programação em Lógica, incidindo-se particularmente na Programação em Lógica Estendida. Seguindo-se posteriormente à exposição dos conteúdos dados nas aulas práticas.

Para além dos temas referidos anteriormente foi ainda dada a questão do tratamento de informação incompleta e da qualidade de informação e grau de confiança que lhe estão adjacentes.

Nas aulas práticas foi utilizada a linguagem de programação em lógica chamada Prolog, que é uma linguagem de programação declarativa. Apresentam-se alguns exercícios que resultam da tradução directa de alguns dos exercícios das aulas teóricas, sendo deste modo possível apresentar algumas das questões que se pode fazer ao programa e analisar as diferentes respostas dadas.

Programação em Lógica

Na disciplina de Representação de Conhecimento foi introduzido o tema da Programação em Lógica Estendida no âmbito do tratamento de sistemas de Informação Incompleta.

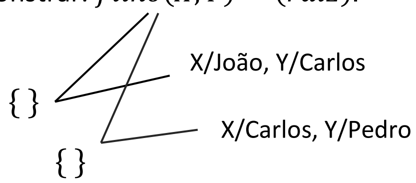
Estes sistemas têm o objectivo de manipular e armazenar informação, utilizando para isso, um conjunto de frases declarativas (predicado) que podem ser apresentadas sob a forma de premissas (factos e regras) ou na forma de conclusões. Num sistema de representação incompleta em Programação Lógica, estas premissas podem tomar apenas o valor Verdadeiro ou Falso, pois este baseia-se no Pressuposto do Mundo Fechado. Este pressuposto define que tudo que não consegue ser provado como sendo Verdade é considerado Falso. A este método de exclusão da veracidade de uma premissa chama-se negação fraca. Visto que todos os exemplos apresentados nas aulas se baseiam na Representação em Lógica Estendida é importante referir que esta resulta da aplicação da negação fraca juntamente com a negação forte ou explícita. Neste tipo de programação passa-se a ter que quando não existe um certo átomo em vez de este ser considerado Falso por falta de prova de que é verdadeiro, vai passar a ser considerado Desconhecido.

Pode-se então passar à análise do exemplo dado nas aulas.

$$\begin{cases} \neg filho(X,Y) \leftarrow \text{não}\ filho(X,Y). \\ filho(João, Carlos). \\ filho(Carlos, Pedro). \end{cases}$$

Agora surge a questão de como interrogar o sistema. Num sistema declarativo temos que não existe atribuição de variáveis mas sim unificação da questão feita com as premissas declaradas no programa. Deste modo aplica-se a negação fraca, tendo-se que um teorema nunca é demonstrado, chegando-se apenas a contrariedades da sua falsidade. Questionando o sistema anteriormente representado e sabendo que este se baseia em árvores de prova para provar os teoremas. Para a pergunta "Quem é filho de quem?" tem-se então o seguinte teorema.

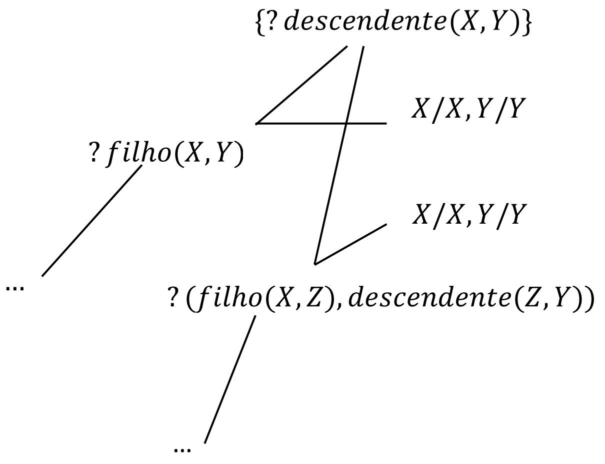
- Teorema a demonstrar: $filho(X,Y) \equiv (raiz)$.



O programa apresentado acima é um programa simples em que o que não está declarado nos factos (ex. filho(Joao,Carlos)) é considerado Falso. Considere-se agora o caso descendente, neste é necessário introduzir a recursividade em que uma função é provada à custa de si própria. Neste tipo de função não existem factos, pois esta é definida a custa de regras que dependem de outras funções. Neste caso tem-se que a função descendente é provada à custa da função filho definida acima. Passamos a ter então a seguinte estrutura.

$$\begin{cases} \neg filho(X,Y) \leftarrow \text{não } filho(X,Y). \\ filho(João, Carlos). \\ filho(Carlos, Pedro). \\ \neg descendente(X,Y) \leftarrow \text{não } descendente(X,Y). \\ descendente(X,Y) \leftarrow filho(X,Y). \\ descendente(X,Y) \leftarrow filho(X,Z), descendente(Z,Y). \end{cases}$$

Tome-se agora em consideração a pergunta “Quem é descendente de quem?”



Verifica-se então que ocorre a unificação do teorema com as regras referentes a função descendente, para a primeira solução verifica-se que o sistema unifica descendente(X,Y) com a primeira regra, tentando unificar de seguida filho(X,Y) em que se iria obter a árvore da função filho referida anteriormente. No segundo caso o sistema vai começar por criar uma árvore relativa à função filho e sempre que chegar a uma solução de filho(X,Z) segue para a unificação de descendente(Z,Y), criando uma árvore semelhante à representada acima.

Regra da derivação

O caso apresentado acima foi ainda utilizado para a introdução da regra da derivação ou *Modus Tollens* que define que se $X \rightarrow T$ então se $\neg X$ temos que $\neg T$, isto é,

$$\frac{?\,T \qquad T \leftarrow X}{?\,X}$$

Temos então o sistema anterior com as funções filho e descendente ao qual perguntamos de quem é descendente o João. Pode-se começar por definir a relação de derivabilidade.

$$\vdash = \{< S, s > | S \subseteq LP, s \in LP \text{ e } s \text{ é derivado de } S \text{ através de regras de derivação}\}$$

Aplicando a regra de derivação obtém-se as seguintes árvore de prova e relações de derivabilidade.

$$\{?\,descendente(João, X)\}$$

$$(1) \qquad João/João, X/Y$$

$$?\,filho(João, Y))$$

$$(2) \qquad João/João, Y/Carlos$$

$$\{\,\}$$

$$(1)\ \frac{?\,descendente(João, X) \qquad descendente(X, Y) \leftarrow filho(X, Y)}{?\,filho(João, Y)}$$

$$(2)\ \frac{?\,filho(X, Y) \qquad filho(João, Carlos) \leftarrow V}{?\,V}$$

$$\vdash\, = \{< S, descendente(João, X) >, < S, filho(João, X) >, \{\}\,\}$$

Pela regra da derivação conclui-se que João só é descendente de Carlos.

Acrescente-se agora as funções feminino e masculino ao sistema anterior.

$$\left\{\begin{array}{c}
\neg filho(X, Y) \leftarrow não\ filho(X, Y). \\
filho(João, Carlos). \\
filho(Carlos, Pedro). \\
filho(João, Maria). \\
\neg descendente(X, Y) \leftarrow não\ descendente(X, Y). \\
descendente(X, Y) \leftarrow filho(X, Y). \\
descendente(X, Y) \leftarrow filho(X, Z), descendente(Z, Y). \\
\neg feminino(X) \leftarrow não\ feminino(X). \\
feminino(Maria). \\
\neg masculino(X) \leftarrow não\ masculino(X). \\
masculino(Carlos). \\
masculino(Pedro).
\end{array}\right.$$

Pergunta 1: Quais os pais que têm filhos chamados João.

Hipótese 1: filho(João,Carlos). Esta é a primeira hipótese, pois é seguida a ordem que se utiliza ao definir o sistema.

$$\{?\,descendente(João, X), masculino(X)\}$$

$$(1)$$

$$\text{João}/\text{João}, X/Y$$

$$?\,(filho(\text{João}, Y), masculino(X))$$

$$(2) \qquad \text{João}, Y/Carlos$$

$$?\,masculino(Carlos)$$

$$(3) \qquad Carlos/Carlos$$

$$\{\,\}$$

$$(1)\ \frac{?\,descendente(\text{João}, X) \qquad descendente(X, Y) \leftarrow filho(X, Y)}{?\,filho(\text{João}, Y)}$$

$$(2)\ \frac{?\,filho(\text{João}, Y) \qquad filho(\text{João}, Carlos) \leftarrow V}{?\,(V, masculino(Carlos))}$$

$$(3)\ \frac{?\,masculino(Carlos) \qquad masculino(Carlos) \leftarrow V}{?\,V}$$

$$\vdash = \{< S, descendente(\text{João}, X) >, < S, filho(\text{João}, X) >, < S, masculino(Carlos) >, \{\}\,\}$$

Hipótese 2: filho(João,Maria).

$$\{?\,descendente(\text{João}, X), masculino(X)\}$$

$$(1) \qquad \text{João}/\text{João}, X/Y$$

$$?\,(filho(\text{João}, Y), masculino(X))$$

$$(2) \qquad \text{João}, Y/Maria$$

$$?\,masculino(Maria)$$

$$(3)$$

$$\cancel{\{\}} \equiv insucesso$$

$$(1)\ \frac{?\,descendente(\text{João}, X) \qquad descendente(X, Y) \leftarrow filho(X, Y)}{?\,filho(\text{João}, Y)}$$

$$(2)\ \frac{?\,filho(\text{João}, Y) \qquad filho(\text{João}, Maria) \leftarrow V}{?\,(V, masculino(Maria))}$$

$$(3)\ \frac{?\,masculino(Maria) \qquad -}{-}$$

Pergunta 2: Quais as mães que têm filhos chamados João.

Hipótese 1: filho(João,Carlos).

$\{?\,descendente(João, X), feminino(X)\}$

(1) $\quad João/João, X/Y$

$?\,(filho(João, Y), feminino(X))$

(2) $\quad João, Y/Carlos$

$?\,masculino(Carlos)$

(3)

$\cancel{\{\}} \equiv insucesso$

$$(1)\ \frac{?\,descendente(João, X) \qquad descendente(X, Y) \leftarrow filho(X, Y)}{?\,filho(João, Y)}$$

$$(2)\ \frac{?\,filho(João, Y) \qquad filho(João, Carlos) \leftarrow V}{?\,(V, feminino(Carlos))}$$

$$(3)\ \frac{?\,feminino(Carlos) \qquad -}{-}$$

Hipótese 1: filho(João,Maria).

$\{?\,descendente(João, X), feminino(X)\}$

(1) $\quad João/João, X/Y$

$?\,(filho(João, Y), feminino(X))$

(2) $\quad João, Y/Maria$

$?\,feminino(Maria)$

(3) $\quad Maria/Maria$

$\{\,\}$

$$(1)\ \frac{?\,descendente(João, X) \qquad descendente(X, Y) \leftarrow filho(X, Y)}{?\,filho(João, Y)}$$

$$(2)\ \frac{?\,filho(João, Y) \qquad filho(João, Maria) \leftarrow V}{?\,(V, feminino(Maria))}$$

$$(3)\ \frac{?\,feminino(Maria) \qquad feminino(Maria) \leftarrow V}{?\,V}$$

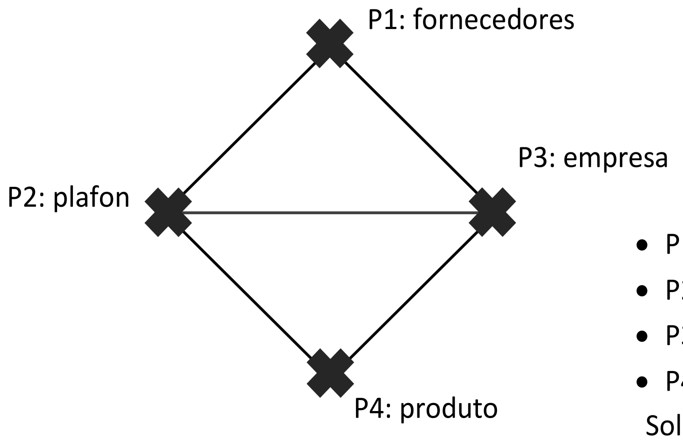
$$\vdash = \{< S, descendente(João, X) >, < S, filho(João, X) >, < S, feminino(Maria) >, \{\}\}$$

Conclui-se que Maria e Carlos são os únicos com um descendente chamado João e que são do sexo feminino e masculino respectivamente.

Invariantes

Os invariantes funcionam como regras que permitem garantir a coesão e a fiabilidade dos dados. O primeiro exemplo utilizado para a introdução deste tema foi o exemplo das empresas e fornecedores que inclui uma estrutura baseada em arcos e nodos. Neste caso os nodos representam as entidades do problema, isto é, as empresas, os produtos, os fornecedores e o respectivo plafon (do primeiro exemplo). A cada nodo associa-se um conjunto de propriedades que caracterizam o seu conhecimento, no universo que se insere. Os arcos retratam o modo como as relações entre as entidades são estabelecidas.

Exemplo 1:



- P1: $X_1, X_2, X_3, X_4 \rightarrow 0, \ldots, 1$
- P2: $X_1, X_5, X_6 \rightarrow 0, \ldots, 1$
- P3: $X_1, X_5, X_7, X_8 \rightarrow 0, \ldots, 1$
- P4: $X_5, X_9, X_{10} \rightarrow 0, \ldots, 1$

Soluções: $X, T, S \rightarrow 0, \ldots, 1$

$$
\begin{cases}
\neg fornecedor(X_1, X_2, X_3, X_4) \leftarrow \text{não } fornecedor(X_1, X_2, X_3, X_4). \\
fornecedor(1, João, Braga, Solteiro). \\
fornecedor(7, Pedro, Tavira, Casado). \\
\neg plafon(X_1, X_5, X_6) \leftarrow \text{não } plafon(X_1, X_5, X_6). \\
plafon(1,17,250). \\
plafon(7,17,150). \\
plafon(7,11,200). \\
\neg produto(X_5, X_7, X_8) \leftarrow \text{não } produto(X_5, X_7, X_8). \\
produto(17,21, Amarelo). \\
\neg empresa(X_1, X_5, X_9, X_{10}) \leftarrow \text{não } empresa(X_1, X_5, X_9, X_{10}). \\
empresa(1,17,10,140). \\
\neg temp(X) \leftarrow \text{não } temp(X). \\
\ldots \\
(2), (3), (4)
\end{cases}
$$

Questão 1: Remova da Base de Conhecimento ou PL o fornecedor com o número de ordem 1.

Em primeiro lugar define-se que não pode existir um fornecedor que não tenha sido declarado em plafon.

$$? (\neg plafon(X_1, X_5, X_6), fornecedor(X_1, X_2, X_3, X_4))$$

$$\boldsymbol{Caso\ Geral}: \{+, -, ?, \ldots\}\ predicado :: \ soluções(X, T, S)$$

$$\boldsymbol{Caso\ Particular}: \ -fornecedor(X_1, X_2, X_3, X_4)$$
$$:: \ soluções(\_, (\neg plafon(X_1, X_5, X_6), fornecedor(X_1, X_2, X_3, X_4)), []).$$

Este invariante apenas serve para o caso da remoção, no entanto pode ser aplicado a qualquer cláusula (nome, cidade, etc…). Para responder à pergunta, temos que saber quais os termos plafon que se pretende remover.

Passos:

i. $?\, soluções\ (INV, -fornecedor(1, X_2, X_3, X_4) :: INV, S).$
ii. $?\, invocar\ (S).$
iii. $?\, remover(fornecedor(1, \_, \_, \_)).$

De modo a se realizar as operações de invocar, remover e soluções, é necessário substituir no sistema representado no início do exemplo os valores (2), (3), (4) pelos seguintes subsistemas.

$$
(2)
\begin{cases}
soluções(X, T, S) \leftarrow \ \mathbf{T}, \mathrm{assert(temp(X)), fail.} \\
soluções(X, T, S) \leftarrow construir(S, S). \\
construir(S, S) \leftarrow retract(temp(X)), construir([X|S], S). \\
construir(S, S).
\end{cases}
$$

$$(3)\begin{cases} \neg invocar(X) \leftarrow \text{ não } invocar(X). \\ invocar([\,]). \\ invocar([X|Y]) \leftarrow X, invocar(Y). \end{cases}$$

$$(4)\begin{cases} \neg remover(X) \leftarrow \text{ não } remover(X). \\ remover([\,]). \\ remover([X|Y]) \leftarrow retract(X), remover(Y). \end{cases}$$

Questão 2: Quais os fornecedores que proporcionam produtos de cor preta?

Em primeiro lugar há que interpretar a questão dada. Tem que se declarar os diversos arcos, por forma a possibilitar caminhar sobre eles.

$$\boldsymbol{arco}: Nodo_i, Nodo_t \rightarrow 0, \ldots, 1$$

$$\begin{cases} \neg arco(N_i, N_t) \leftarrow \text{não } arco(N_i, N_t). \\ arco(fornecedor(X_1, X_2, X_3, X_4), plafon(X_1, X_5, X_6)). \\ arco(fornecedor(X_1, X_2, X_3, X_4), empresa(X_1, X_5, X_9, X_{10})). \\ arco(plafon(X_1, X_5, X_6), empresa(X_1, X_5, X_9, X_{10})). \\ arco(plafon(X_1, X_5, X_6), produto(X_5, X_7, X_8)). \\ arco(empresa(X_1, X_5, X_9, X_{10}), produto(X_5, X_7, X_8)). \end{cases}$$

De seguida é apresentado o predicado que permite caminhar no grafo.

$$\boldsymbol{caminho}: [N_i|X], N_t, S \rightarrow 0, \ldots, 1$$

i. Quando se caminha (à partida) do N_i para o N_t , e $N_i = N_t$, então $[N_i|X] = [N_i|[\,]\,] = [N_i] = [N_t]$;
ii. Quando se caminha do N_i para o N_t e $N_i \neq N_t$, então tem-se que em $[N_i|X], X \neq [\,]$;
iii. $$\begin{cases} \neg caminhar(X, Y, Z) \leftarrow \text{ não } caminhar(X, Y, Z). \\ caminhar([N_{\boldsymbol{T}}|\mathbf{T}_{\boldsymbol{nodos}}],\ \boldsymbol{T}_{\boldsymbol{erminal}}, [N_{\boldsymbol{T}}|\mathbf{T}]). \\ caminhar([N_{\boldsymbol{i}}|\mathbf{T}], N_{\boldsymbol{T}}, Z) \leftarrow (arco(N_{\boldsymbol{i}}, N_{\boldsymbol{intermedio}});\ arco(N_{\boldsymbol{intermedio}}, N_{\boldsymbol{i}})), \\ \neg elemento(N_{\boldsymbol{intermedio}}, \mathbf{T}),\ caminhar([N_{\boldsymbol{intermedio}}, N_i|\mathbf{T}], N_{\boldsymbol{T}}, Z). \end{cases}$$

Temos então que,

$$?\, soluções(Z, (caminhar([fornecedor(X_1, X_2, X_3, X_4)], produto(X_5, X_7, X_8), Z), S)$$

$$\begin{cases} soluções(X, T, S') \leftarrow \mathbf{T}, \text{assert(temp(X)), fail.} \\ soluções(X, T, S') \leftarrow construir([\,], S'). \\ construir(P, Q) \leftarrow retract(temp(X)), construir([X|P], Q). \\ construir(Q, Q). \end{cases}$$

Obtém-se várias interpretações da questão dada por parte do sistema computacional.

$$S = [\ [fornecedor(X_1, X_2, X_3, X_4), plafon(X_1, X_5, X_6), produto(X_5, preta, X_8)], [fornecedor(X_1, X_2, X_3, X_4), plafon(X_1, X_5, X_6), empresa(X_1, X_5, X_9, X_{10}), produto(X_5, preta, X_8)], [fornecedor(X_1, X_2, X_3, X_4), empresa(X_1, X_5, X_9, X_{10}), produto(X_5, preta, X_8)]\ ].$$

Agora tem que se proceder à escolha de uma opção para obter o resultado.

Finalmente:

$? soluções([X_1, X_3], (fornecedor(X_1, X_2, X_3, X_4), plafon(X_1, X_5, X_6), produto(X_5, X_7, X_8)), S'').$

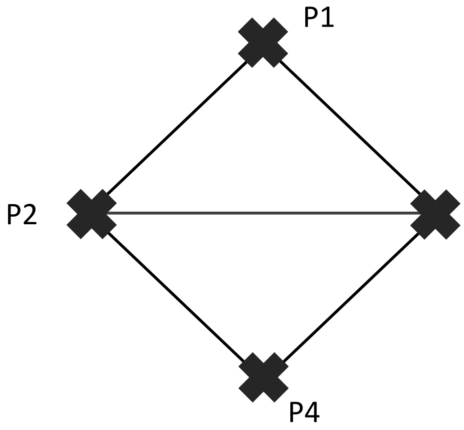
Caso não exista nenhum resultado para fornecedor então o conjunto S'' é uma lista de listas vazias.

$$S'' = [[\ ]]$$

$$\dots$$

$$S'' = [[17, Pedro], [35, Luís]].$$

Exemplo 2: Este exemplo introduz a função excepção, que declara uma excepção aos factos e regras definidas pelo programa.



- P1: $X, Y, Z \rightarrow 0, \dots, 1$
- P2: $X, A, B \rightarrow 0, \dots, 1$
- P3: $X, A, C, D \rightarrow 0, \dots, 1$
- P4: $A, E, F \rightarrow 0, \dots, 1$

Soluções: $X, T, S \rightarrow 0, \dots, 1$

$$\boldsymbol{caminhar}: [N_i], N_f, C \leftarrow 0, \dots, 1$$

$$\left\{ \begin{array}{c} \neg caminhar(X, Y, Z) \leftarrow\ não\ caminhar(X, Y, Z), não\ excepção_{caminhar}(X, Y, Z). \\ caminhar[X|Y], X, [X|Y]). \\ caminhar([X|Y], T_{erminal}, C) \leftarrow (arco(X, Int);\ arco(Int, X)), \\ \neg elemento(Int, X), caminhar([Int, X|Y], T, C). \\ \neg arco(X, Y) \leftarrow\ não\ arco(X, Y), não\ excepção_{arco}(X, Y). \\ arco(p1(X, Y, Z), p2(X, A, B)). \\ arco(p2(X, A, B), p3(X, A, C, D)). \end{array} \right.$$

Questão: Como ir de P1 para P4?

$$? \, soluções(C, (caminhar([p1(X,Y,Z)], p4(A,E,F), C), S)$$

$$\begin{cases} soluções(X,T,S) \leftarrow \mathbf{T}, \text{assert(temp(X)), fail.} \\ soluções(X,T,S) \leftarrow construir([\,], S). \\ construir(P,Q) \leftarrow retract(temp(X)), construir([X|P], Q). \\ construir(Q,Q). \end{cases}$$

Valores do tipo Disjunto e Desconhecido

No exemplo anterior, foi possível observar que surgiu uma nova função chamada excepção, sem no entanto esta ter sido explicada. Esta função é utilizada quando se pretende expressar valores do tipo Desconhecido, Disjunto ou Nulo. Tome-se o seguinte exemplo, apresentado na tabela.

Exemplo 1:

Professor	Disciplinas
Luís	Francês
Pedro	{Latim, Inglês}
Ana	{Chinês, Latim}
Luciana	$\perp$

Tem-se que para o professor Luís a informação se encontra completa, em que se conclui que Luís lecciona Francês, no entanto o mesmo não acontece para os restantes casos. No caso do Pedro em que temos um "ou não exclusivo" isto é um caso disjunto, vão existir três soluções possíveis que vão ser expressas pela função $Excepção_{Lecciona}$. O mesmo se vai verificar para caso da Ana em que se encara um "ou exclusivo" ou não disjunto, nesta situação tem-se que a Ana não pode leccionar as duas disciplinas em simultâneo. E por fim tem-se o caso da Luciana em que a disciplina que esta lecciona é desconhecida. Obtemos então o seguinte sistema.

$$(1) \begin{cases} lecciona: x, y \rightarrow 0, \ldots, 1 \\ \neg lecciona(X,Y) \leftarrow \text{não } lecciona(X,Y), \text{não } excepção_{lecciona}(X,Y). \\ lecciona(Luis, Frances). \\ excepção_{lecciona}(Pedro, Latim). \\ excepção_{lecciona}(Pedro, Ingles). \\ \ldots \\ (2), (3) \end{cases}$$

Os cenários de solução que se podem gerar considerando apenas o Luís e o Pedro são os representados em seguida. A análise destes cenários quando se está perante um caso de Conhecimento Incompleto é importante, pois permite atribuir valores de qualidade de informação (QoI) e de graus de confiança que ajudam na escolha entre cenários. Os cenários apresentados serão então apresentados em função do predicado lecciona em que o valor de QoI atribuído a cada termo será apresentado com primeiro argumento e o DoC como o último.

<u>1ºcenário</u>:

$$\begin{cases} \neg lecciona(X,Y,Z,W) \leftarrow \text{ não } lecciona(X,Y,Z,W). \\ lecciona(1, Luís, Francês, 1). \\ lecciona(0{,}33, Pedro, Latim, 0{,}75). \end{cases}$$

Em que X corresponde ao valor de QoI e W ao valor correspondente de DoC.

<u>2ºcenário</u>:

$$\begin{cases} \neg lecciona(X,Y,Z,W) \leftarrow \text{ não } lecciona(X,Y,Z,W). \\ lecciona(1, Luís, Francês, 1). \\ lecciona(0{,}33, Pedro, Ingles, 0{,}75). \end{cases}$$

<u>3ºcenário</u>:

$$\begin{cases} \neg lecciona(X,Y,Z,W) \leftarrow \text{ não } lecciona(X,Y,Z,W). \\ lecciona(1, Luís, Francês, 1). \\ lecciona(0{,}165, Pedro, Latim, 0{,}75). \\ lecciona(0{,}165, Pedro, Ingles, 0{,}75). \end{cases}$$

Sabendo-se que os valores da qualidade de informação estão entre o intervalo [0,1], em que 0 corresponde ao caso em que é desconhecido e 1 ao caso em que se tem todo o conhecimento relativamente a um termo, tem-se que todos os outros casos se vão encontrar entre este intervalo. No entanto, no caso de um termo com um valor do tipo disjunto, tem-se que o somatório dos QoIs de todos os cenários possíveis tem que dar um. Deste modo é possível representar estes cálculos na forma de um gráfico circular.

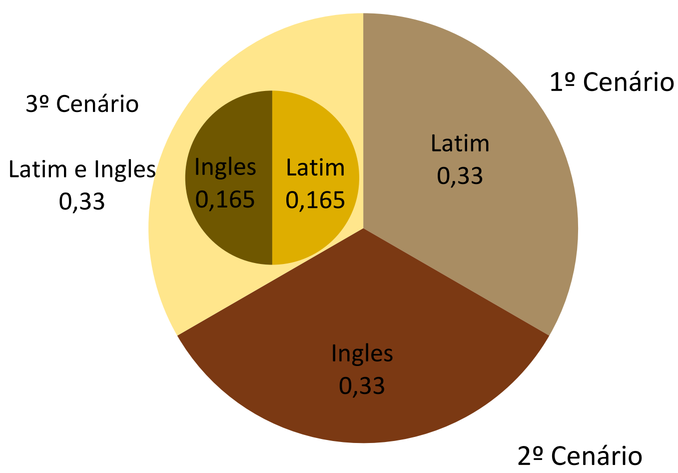


Figura 1- Gráfico da Qualidade de Informação relativa a Pedro

No que respeita aos valores do grau de confiança de cada termo, estes dependem de cada um dos argumentos do termo isto é, sabendo que apenas existe uma solução e que o grau de confiança de cada argumento é um temos o seguinte cálculo.

$$DoC(Luís, Francês) = \frac{1(Luís) + 1(Francês)}{2(número\ de\ argumentos)} = 1$$

No caso dos dois primeiros cenários de Pedro teríamos um valor de DoC apresentado na equação a baixo.

$$DoC = \frac{1(Pedro) + 0{,}5(disciplina)}{2(número\ de\ argumentos)} = 0{,}75$$

No terceiro cenário ter-se-ia o seguinte raciocínio.

$$DoC = \frac{1(Pedro) + 0{,}5(Latim\ e\ Ingles)}{2(número\ de\ argumentos)} = 0{,}75$$

Pode-se agora analisar o sistema considerando os termos relativos ao caso em que $professor = Ana$.

$$(2) \left\{ \begin{array}{c} \ldots \\ excepção_{lecciona}(Ana, Chines). \\ excepção_{lecciona}(Ana, Latim). \\ ?\,(excepção_{lecciona}(Ana, X) \vee excepção_{lecciona}(Ana, Y)) \\ \wedge \\ \neg(excepção_{lecciona}(Ana, X) \wedge excepção_{lecciona}(Ana, Y)). \\ \ldots \\ (3) \end{array} \right.$$

Neste caso teremos apenas dois cenários tal como já foi referido anteriormente. O último termo definido no sistema acima é o termo que restringe os cenários existentes para Ana, este define que Ana só pode leccionar uma disciplina de cada vez e nunca as duas em simultâneo. O gráfico seguinte traduz a QoI relativa a cada um dos destes cenários.

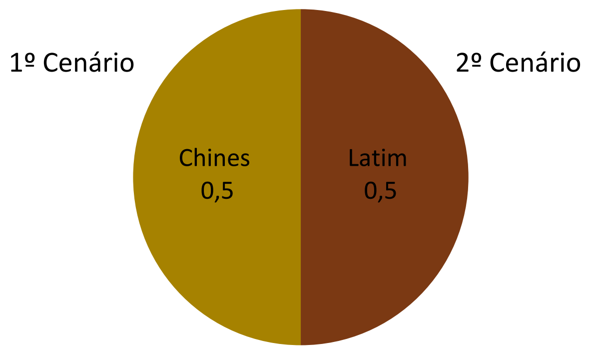


Figura 2- Cenários isolados para Ana

Juntando estes cenários com o sistema que se tinha anteriormente ficamos com $1\ cenário\ de\ Luís\ \times 3\ cenários\ de\ Pedro \times\ 2\ cenários\ de\ Ana\ =\ 6\ cenários\ do\ sistema$, ou seja, obtém-se os seguintes cenários.

<u>1ºcenário</u>:

$$\begin{cases} \neg lecciona(X,Y,Z,W) \leftarrow \ não\ lecciona(X,Y,Z,W). \\ lecciona(1, Luís, Francês, 1). \\ lecciona(0{,}33, Pedro, Latim, 0{,}75). \\ lecciona(0{,}5, Ana, Latim, 0{,}75). \end{cases}$$

<u>2ºcenário</u>:

$$\begin{cases} \neg lecciona(X,Y,Z,W) \leftarrow \ não\ lecciona(X,Y,Z,W). \\ lecciona(1, Luís, Francês, 1). \\ lecciona(0{,}33, Pedro, Ingles, 0{,}75). \\ lecciona(0{,}5, Ana, Latim, 0{,}75). \end{cases}$$

⋮

<u>6ºcenário</u>:

$$\begin{cases} \neg lecciona(X,Y,Z,W) \leftarrow \ não\ lecciona(X,Y,Z,W). \\ lecciona(1, Luís, Francês, 1). \\ lecciona(0{,}165, Pedro, Latim, 0{,}75). \\ lecciona(0{,}165, Pedro, Ingles, 0{,}75). \\ lecciona(0{,}5, Ana, Chines, 0{,}75). \end{cases}$$

Admita-se agora, que existe uma professora chamada Luciana e que não se sabe o que esta lecciona. O valor correspondente de QoI deste termo é 1, pois qualquer que seja a disciplina, este pode ser sempre considerado verdadeiro.

$$(3) \begin{cases} \quad\quad\quad\quad\quad\quad\quad\quad \ldots \\ \neg excepção_{lecciona}(Luciana, X) \ \leftarrow não\ lecciona(Luciana, desconhecida). \\ lecciona(Luciana, desconhecida). \end{cases}$$

O número de cenários vai manter-se igual.

$$cenários\ do\ sistema = 1\ cenário\ de\ Luís\ \times 3\ cenários\ de\ Pedro \times\ 2\ cenários\ de\ Ana\ \times\ cenário\ de\ Luciana\ = 6\ cenários$$

<u>1ºcenário</u>:

$$\begin{cases} \neg lecciona(X,Y,Z,W) \leftarrow \ não\ lecciona(X,Y,Z,W). \\ lecciona(1, Luís, Francês, 1). \\ lecciona(0{,}33, Pedro, Latim, 0{,}75). \\ lecciona(0{,}5, Ana, Latim, 0{,}75). \\ lecciona(1, Luciana, true, 0{,}5). \end{cases}$$

$$\vdots$$

<u>6ºcenário</u>:

$$\begin{cases} \neg lecciona(X,Y,Z) \leftarrow \text{ não } lecciona(X,Y,Z). \\ lecciona(1, Luís, Francês, 1). \\ lecciona(0{,}165, Pedro, Latim, 0{,}75). \\ lecciona(0{,}165, Pedro, Ingles, 0{,}75). \\ lecciona(0{,}5, Ana, Chines, 0{,}75). \\ lecciona(1, Luciana, true, 0{,}5). \end{cases}$$

É importante referir que para o cálculo do DoC, o valor associado à disciplina é 0, pois esta é desconhecida. Tem-se então o seguinte raciocínio.

$$DoC = \frac{1(Luciana) + 0(disciplina)}{2(número\ de\ argumentos)} = 0{,}5$$

Exemplo do Máximo Divisor Comum

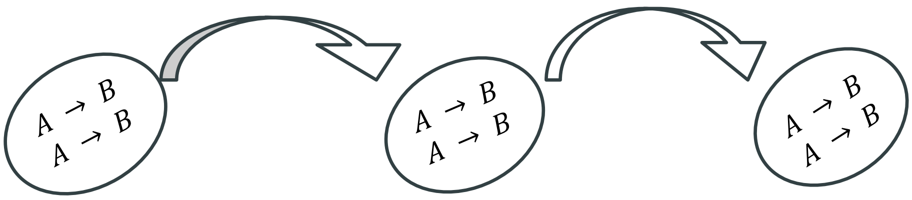
O objectivo deste exemplo foi o desenvolvimento de um meta-predicado que permitisse a extensão da solução a qualquer número de elementos.

$$\begin{cases} \neg nu_{mero}: y, x \rightarrow \{V, F\} \\ \neg nu(X) \leftarrow \text{ não } nu(X). \\ nu(25). \\ nu(15). \\ nu(10). \end{cases}$$

Questão: Qual o máximo divisor comum de 25, 15 e 10?

$$Condição \rightarrow Acção$$

$$\begin{cases} [nu(A), nu(B), A > B] \rightarrow [nu(C) = A - B], remover(nu(A)), inserir(nu(C)). \\ [ler(nu(A))] \rightarrow [imprimir(A), parar]. \end{cases}$$



$$\begin{cases} \neg nu(A) \leftarrow \text{ não } nu(A). \\ nu(25). \\ nu(15). \\ nu(10). \end{cases} \qquad \begin{matrix} nu(10). \\ nu(15). \\ nu(10). \end{matrix} \qquad \begin{matrix} nu(5). \\ nu(5). \\ nu(5). \end{matrix}$$

Tem-se então,

$$\begin{cases} demo: condicção(X) \rightarrow Acção(Y), ver_{ificar}(X), ex_{ecutar}(Y). \\ ver([\,]). \\ ver([X|Y]\,) \leftarrow demo(X), ver(Y). \\ ex([\,]) \leftarrow demo. \\ ex([parar]). \\ ex([X|Y]) \leftarrow demo(X), ex(Y). \\ demo(true). \\ demo([X|Y]) \leftarrow demo(X), demo(Y). \\ demo(X) \leftarrow X \leftarrow Q, demo(Q). \\ demo(X) \leftarrow X. \end{cases}$$

Redes Semânticas

Um outro tipo de relações que se baseiam numa estrutura arco e nodo são as relações hierárquicas. Neste tipo de relações define-se um relacionamento entre subclasse (ou instancia) e superclasse. Tem-se então o predicado *é-um(,)* que permite estabelecer as relações.

Exemplo:

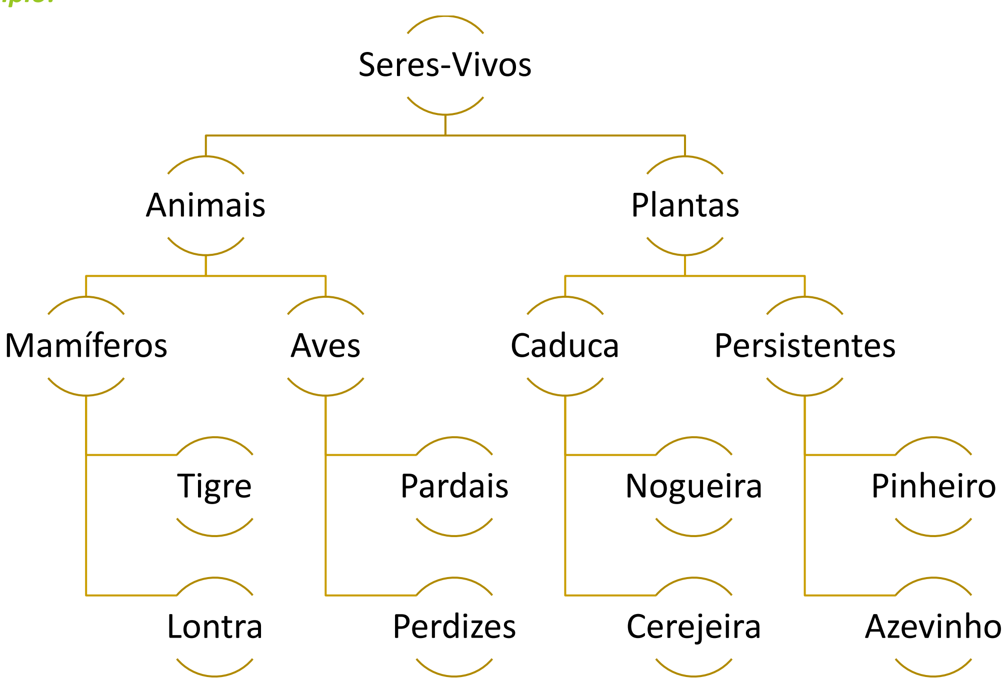


Figura 3- Rede Semântica Seres-Vivos

$$\boldsymbol{objecto}: nome\ do\ objecto, teoria \ \rightarrow \{V, F\}$$

$$\text{é} - \boldsymbol{um}: objecto, classe \rightarrow \{V, F\}$$

Como vamos perguntar ao sistema se o Tigre é um Ser-vivo?

$$?\, mensagem(objecto, teorema).$$

Em que o teorema traduz o conteúdo da mensagem enviada ao sistema e a teoria do objecto está presente no teorema a ser demonstrado.

$$\neg mensagem(objecto, teorema) \leftarrow \text{ não } mensagem(objecto, teorema).$$
$$mensagem(Objecto, teorema) \leftarrow objecto(Objecto, teoria), demo_{padrão}(teoria, teorema).$$

Então $?\, mensagem(Tigre, Seres - vivos)$. vai seguir o seguinte raciocínio.

$$mensagem(Objecto, teorema) \leftarrow$$
$$\text{é} - um(Objecto, Classe), mensagem(Classe, teorema).$$

Admitindo-se que estariam já declarados os factos relativos ao predicado é-um, que declara as relações entre os varios nodos da rede semântica.

$$\text{é} - um(Animais, Seres - vivos).$$
$$\text{é} - um(Tigre, Mamíferos).$$
$$\ldots$$

Faltarnos-ia apenas declarar o que é um objecto.

$$objecto(seres - vivos, (\neg seres - vivos(X) \leftarrow$$
$$\text{não } seres - vivos(X),$$
$$seres - vivos(Animais),$$
$$seres - vivos(Plantas).$$

Prolog

Nas aulas práticas de Representação de Conhecimento foi introduzida a linguagem de programação em lógica, Prolog. Esta é uma linguagem declarativa que usa um fragmento da lógica de 1º ordem (as Cláusulas de Horn) para representar o conhecimento sobre um dado problema. Um programa em Prolog é constituido por axiomas e regras de inferencia que definem um dado problema.

Cláusulas de Horn são fórmulas da forma $p \Leftarrow q1 \wedge q2 \wedge \ldots \wedge qn$ que são representadas por Prolog do seguinte modo $p :- q1,\ \ q2, \ldots ,\ \ qn$.

O objectivo da utilização desta liguagem nas aulas práticas da disciplina foi poder implementar e testar todos os conceitos dados nas aulas teóricas e completar alguns desses conhecimentos. Deste modo, neste capítulo serão apresentados os programas feitos.

Os dois primeiros exemplos resultam da implementação directa de dois dos programas criados nas aulas teóricas. Nestes vão ser declarados os factos em primeiro lugar e depois as regras, pois, como esta é uma linguagem em que se testa a veracidade de uma questão através da unificação com a base de dados e conhecimento.

Exemplo 1: **Filhos e Descendentes**

```
filho(joao, carlos).
filho(joao, maria).
Filho(carlos, pedro).
Masculino(pedro).
masculino(carlos).
feminino(maria).
nao_filho(X,Y):-nao(X,Y).
nao_des(X,Y):-nao(X,Y).
des(X,Y):-filho(X,Y).
des(X,Y):-filho(X,Z), des(Z,Y).
nao_masculino(X):-nao(X).
nao_feminino(X):-nao(X).
```

Questão 1: Quem é que tem um filho chamado João?

```
?- bagof((joao,X), filho(joao,X),S).
S=[(joao, carlos),(joao,maria)]  ?
yes
```

Questão 2: Quem é que tem um descendente chamado João?

```
?-bagof((joao,X),des(joao,X),S).
S=[(joao, carlos),(joao,maria),(joao,pedro)]  ?
yes
```

Questão 3: Pedro é pai de João?

?-filho(joao, pedro).

No

Questão 4: Quais os ascendentes masculinos de João?

?-des(joao,X), masculino(x).

X= carlos ?;

X= pedro ?;

no

Questão 5: Existe algum ascendente feminino de João?

?-des(joao,X), feminino(x).

X= maria ?

yes

Estas são apenas algumas das perguntas que se pode fazer ao programa. Na questão 1 e 2 foi utilizada a função bagof(X,Y,S) em que o argumento X corresponde ao formato em que são dispostas as respostas, Y é a pergunta que se pretende efectuar e S é a lista do conjunto de soluções. Esta função retorna todos os valores verdadeiros para a nossa questão. Na pergunta 4 está presente uma outra particularidade que é o ";". Uma das particularidades do Prolog é que este apenas nos dá uma solução, a não ser que se utilize a função bagof, que o obriga a retornar todas as soluções num conjunto ou então é possível utilizar ";" que o obriga a fazer uma nova pesquisa. No fim da questão 4 obteve-se um não como resposta, que significa que não existem mais soluções.

Exemplo 2: **Máximo divisor comum**

```
:-op(800,xfx,--->).
:-dynamic(nu/1).

nao_nu(X):- nao(X).
nu(25).
nu(15).
nu(10).

[nu(A), nu(B), A>B]--->[C is A-B, retract(nu(A)), assert(nu(C))].
[nu(A)]--->[print(A), parar].

demo:-Con--->Ac, ver(Con), ex(Ac).

ver([]).
ver([X|Y]):-call(X),ver(Y).

ex([]):-demo.
ex([parar]).
ex([X|Y]):-call(X), ex(Y).
```

Neste ultimo exemplo foi necessário definir dois novos operadores em que o argumento xfx define a forma como é construída a árvore de prova. Neste caso em que temos x antes e depois de f significa que os ramos esquerdo e direito da árvore vão estar equilibrados.

Questão: ?- demo.
5
yes

Neste programa temos a função demo que inicia a execução do programa. Fazendo a pergunta acima recebemos como resposta o valor do máximo divisor comum.

Listas

As listas foram introduzidas nas aulas e aplicadas em vários programas, deste modo são apresentados ao longo desta secção algumas das caracteristicas das listas e como podem ser usadas. Quando se faz um programa em prolog usando listas é importante ter em conta a sua estrutura e a sua representação nesta linguagem. Uma lista é normalmente representada do seguinte modo, [H|T], em que H (abreviatura de "head" ou cabeça), ou seja, do primeiro elemento da lista e depois do | (simbolo de separação) tem-se o resto da lista ou T (de "tail" ou cauda) que por sua vez é também uma lista.

Exemplo npertence: Um dos primeiros programas realizados foi o npertence que verifica se um valor pertence a uma determinada lista dada como argumento.

```
npertence([],X,0).
npertence([X|T],X,N):-npertence(T,X,N1), N is N1+1.
npertence([_|T],X,N):-npertence(T,X,N).
```

O primeiro termo deste programa define qual a resposta quando se recebe uma lista vazia como argumento, o segundo termo determina o procedimento de quando o número está presente na primeira posição da lista e implementa a contagem da variável que define o número de vezes que o número em questão aparece na lista. O último termo define o que o programa faz quando o elemento que se procura não se encontra na primeira posição da lista. Ainda neste termo podemos observar que o primeiro elemento da lista foi substituido por um simbolo "_" que significa que esse elemento não é relevante, sendo por isso desnecessário ter em conta o seu valor enquanto se executa a árvore de prova.

Questão 1: O elemento 4 pertence a lista [4,4,5,4,4,6,4,4], se sim quantas vezes?

?- npertence([4,4,5,4,4,6,4,4],4,N).
N=6 ?
yes

Questão 2: O elemento 4 pertence a lista [], se sim quantas vezes?

?- npertence([],4,N).
N=0 ?
yes

Questão 3: O elemento 4 pertence a lista [1,2,3,5], se sim quantas vezes?

?- npertence([1,2,3,5],4,N).

N=0 ?

Yes

É ainda possível verificar se as operações estão a ser executadas correctamente, utilizando-se a função trace.

Questão 4: O elemento 4 pertence a lista [4,2,4], se sim quantas vezes?

?- trace.

?- npertence([4,2,4],4,N).

```
1      1 Call: npertence([4,2,4],4,_572) ?
2      2 Call: npertence([2,4],4,_1247) ?
3      3 Call: npertence([4],4,_1247) ?
4      4 Call: npertence([],4,_2585) ?
4      4 Exit: npertence([],4,0) ?
5      4 Call: _1247 is 0+1 ?
5      4 Exit: 1 is 0+1 ?
3      3 Exit: npertence([4],4,1) ?
2      2 Exit: npertence([2,4],4,1) ?
6      2 Call: _572 is 1+1 ?
6      2 Exit: 2 is 1+1 ?
1      1 Exit: npertence([4,2,4],4,2) ?
```

N=2 ?

Yes

Exemplo permuta: Dadas duas listas, este verifica se uma resulta da permutação da outra, dada apenas uma lista este encontra as suas permutações.

```
retirar(X,[X|R],R).
retirar(X,[Y|R],[Y|R1]):-retirar(X,R,R1).
permutacao([],[]).
permutacao([X|R],Z):- permutacao(R,Resto),retirar(X,Z,Resto).
```

Questão 1: [2,3,1] é permutação de [1,2,3]?

?-permutação([1,2,3],[2,3,1]).

Yes

Questão 2: Qual o conjunto de solução das permutações de [1,2,3]?

?-bagof(L,permutação([1,2,3],L),S).

S=[[1,2,3],[2,1,3],[2,3,1],[1,3,2],[3,1,2],[3,2,1]] ?

Yes

Exemplo retirartodos: Dada uma lista, retira o elemento sempre que aparecer.

```
retirartodos(X,[],[]).
retirartodos(X,[X|Y],Z):-!,retirartodos(X,Y,Z).
retirartodos(X,[Y|R],[Y|Z]):-retirartodos(X,R,Z).
```

Questão 1: Retira o 2 da lista [1,2,3,2,2]?

?-retirartodos(2,[1,2,3,2,2],L).

L=[1,3] ?
Yes

Questão 2: Retira o 5 da lista [1,2,3,2,2]?
?-retirartodos(5,[1,2,3,2,2],L).
L= [1,2,3,2,2]?
Yes

Exemplo ordenação: Ordena a lista.

```
qs([],[]).
qs([X|R],S):-particao(X,R,R1,R2),qs(R1,S1),qs(R2,S2),
             concatenar(S1,[X|S2],S).

concatenar([],S,S).
concatenar([X|R],S,[X|S2]):-concatenar(R,S,S2).

particao(X,[],[],[]).
particao(X,[Y|B],[Y|R1],R2):-Y<X,particao(X,B,R1,R2).
particao(X,[Y|B],R1,[Y|R2]):-Y>=X,particao(X,B,R1,R2).
```

Questão 1: Ordenar lista [2,1,4,3]?
?-qs([2,1,4,3],S).
S= [1,2,3,4]?
Yes

Questão 2: Verifica se a ordenação da lista [2,1,4,3] é [1,2,3,4]?
?-qs([2,1,4,3],[1,2,3,4]).
Yes

Questão 3: Verifica se a ordenação da lista [2,1,4,3] é [1,3,2,4]?
?-qs([2,1,4,3],[1,3,2,4]).
No

Conclusão

A realização deste trabalho permitiu consolidar os conhecimentos adquiridos no decurso do presente semestre. Tendo em conta que o objectivo principal deste trabalho era, acima de tudo, obrigar a revisão de toda a matéria exposta nas aulas, podemos concluir que conseguimos atingi-lo. Ao longo da elaboração deste trabalho, tivemos contacto com a programação declarativa pela primeira vez, e apercebemo-nos das suas vantagens e desvantagens. O contacto com esta nova forma de programar, abriu-nos horizontes para um novo tipo de raciocínio, que nos permitiu alargar a nossa realidade e adicionar novas potencialidades ao nosso conhecimento.

Esperamos num futuro próximo poder aplicar estes novos conhecimentos adquiridos, para solucionar problemas da vida real.

Bibliografia

i. Analide, C., & Neves, J. (1996). *Representação de Informação Incompleta.*

ii. Analide, C., & Neves, J. (1997). *Estruturas Hierárquicas com Herança.*

iii. Neves, J. (1984). *A Logic Interpreter to handle time and negation in logic data bases.*

iv. Neves, J. (2011). *Evolutionary Intelligence in Agent based Modelling.*

v. Neves, J., & Machado, J. (1992). *Lógica Computacional Volume I e II.*

vi. Neves, J., & Machado, J. (2001). *Representação Conhecimento.*