UNIVERSIDADE DO MINHO

Programação em lógica estendida e Conhecimento imperfeito

Mestrado integrado em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio

(2ºSemestre/2017-2018)

a77320 Carlos Pedrosa
a78938 David Sousa
a78869 Manuel Sousa

Universidade do Minho,
Departamento de Informática,
4710-057 Braga, Portugal

abril, 2018

Resumo

O presente trabalho tem como principal objetivo aprimorar a utilização da linguagem de programação em lógica *PROLOG*, no âmbito da representação de conhecimento imperfeito, recorrente à utilização de valores nulos e da criação de mecanismos de raciocínio adequados. Deste modo, pretendemos descrever os esforços efetuados no sentido a atingir o objetivo proposto. De facto, depois de feita a introdução do projeto, é descrita detalhadamente a solução apresentada para dar resposta aos diversos exercícios. Por último, em jeito de conclusão apresentar-se-á uma reflexão crítica sobre os resultados que foram obtidos.

Índice

Índice de Figuras

1. Introdução

A linguagem de programação *PROLOG* é baseada na lógica matemática, nesse sentido, permite expressar programas na forma lógica e usar processos de inferência para produzir resultados. De facto, é uma linguagem de programação declarativa que difere da semântica imperativa e funcional das habituais linguagens. Para além deste facto, a escolha de representação de conhecimento é uma tarefa bastante árdua, uma vez que, é necessário garantir a construção de uma base de conhecimento, estável e consistente.

Em suma, o principal objetivo é a introdução da programação em lógica aos alunos, através de um exercício que consiste na estruturação de conhecimento dinâmico envolvendo uma área de prestação de cuidados de saúde.

2. Preliminares

Para um melhor entendimento do projeto apresentado, é necessário que o leitor apresente conhecimentos na área de programação em lógica, mais concretamente, de assuntos que façam referência a conhecimento imperfeito, positivo e negativo. Nesse sentido, a aprendizagem efetuada pelo grupo no decorrer das aulas foi crucial para o desenvolvimento de todo o projeto. Para além disso, uma análise cuidada e prévia do enunciado proposto permitiu identificar qual seria o melhor método a seguir, de modo a dar resposta aos diversos elementos apresentados.

3. Descrição do trabalho e análise dos resultados

A par com os valores de verdade, verdadeiro ou falso, surge agora o desconhecido, tornando assim mais ampla a capacidade de representação do conhecimento. Para além disso, surge ainda a possibilidade de existir conhecimento negativo, positivo e imperfeito (incerto, impreciso e interdito). É na conjugação destes factos que pretendemos debruçar a nossa atenção pelo que nesta secção pretendemos exemplificar todos os passos dados no sentido a cumprir o objetivo enunciado. Pretendemos dividir o presente relatório, segundo o molde de interrogações fornecidas pelo docente, ainda que estas, não sejam totalmente independentes.

3.1. Pressupostos adotados

Para a realização deste trabalho adotamos alguns pressupostos. Um destes foi a adoção do mundo fechado. De facto, consideramos que não fazia sentido que qualquer um dos predicados que não fosse representado tivesse o valor desconhecido. Sempre que é perguntado à base de conhecimento, conhecimento referente a um predicado que esta não possua a resposta será "falsa".

```
% Adoção do pressuposto do mundo fechado -> Utente
-utente(ID, Nome, Sexo, Idade, Morada) :- nao(utente(ID, Nome, Sexo, Idade, Morada)),
                                          nao(excecao(utente(ID, Nome, Sexo, Idade, Morada))).
```

Figura 1 - Adoção do mundo fechado para o utente

```
% Adoção do pressuposto do mundo fechado -> Prestador
-prestador(ID, Nome, Especialidade, Instituicao) :- nao(prestador(ID, Nome, Especialidade, Instituicao)),
                                                    nao(excecao(prestador(ID, Nome, Especialidade, Instituicao))).
```

Figura 2 - Adoção do mundo fechado para o prestador

```
% Adoção do pressuposto do mundo fechado -> Cuidado
-cuidado(Data, IdUt, IdPrest, Descricao, Custo) :- nao(cuidado(Data, IdUt, IdPrest, Descricao, Custo)),
                                                   nao(excecao(cuidado(Data, IdUt, IdPrest, Descricao, Custo))).
```

Figura 3 - Adoção do mundo fechado para o cuidado

3.2. Resposta às alíneas propostas

- **Representar conhecimento positivo e negativo**

Neste trabalho é necessário distinguir conhecimento positivo do conhecimento negativo. Neste sentido, a introdução do conhecimento negativo é feita com o auxílio do operador "-". Este permite assim indicar o facto de algum utente, prestador ou cuidado não ser abrangido pela base de conhecimento.

```
% ------------------------ CONHECIMENTO NEGATIVO --------------------------- %

% Não pode ser um utente se for um prestador
% utente : N -> {V,F,D}
-utente(_, Nome, Sexo, Idade, Morada) :- prestador(ID, Nome, Especialidade, Instituicao).

% prestador : N -> {V,F,D}
-prestador(_, Nome, Especialidade, Instituicao) :- utente(ID, Nome, Sexo, Idade, Morada).

% É falsa a informação de que certos utentes/prestador tenham estado/trabalhem no centro de Saúde
-utente(11, miguel, 'M', 22, 'Mafra').
-utente(12, marta, 'F', 33, 'Oeiras').
-utente(13, fatima, 'F', 32, 'Faro').

-prestador(11, veronica, 'Imunoalergologia', 'Braganca').
-prestador(12, nicole, 'Ortopedia', 'Barcelos').
-prestador(13, judite, 'Pneumologia', 'Alentejo').

% O centro de saúde não funcionou no dia 27/03/2018 porque esteve fechado por causa da greve:
%     - É falsa a existência de cuidados para esse dia
-cuidado((2018,3,27), _, _, _, _).
```

Figura 4 - Inserção de conhecimento negativo

- **Representar casos de conhecimento imperfeito, pela utilização de valores nulos de todos os tipos estudados**

Existem três tipos de conhecimento imperfeito, conhecimento imperfeito incerto, impreciso e interdito. Assim, numa primeira instância representamos este tipo de conhecimento e posteriormente tratamos dos desafios que a sua inserção apresentava.

```
% Foi contratado um prestador em formação pelo que este ainda não possui especialidade
prestador(14, alberto, incEspecialidade1, 'Tomar').
excecao(prestador(ID, Nome, Especialidade, Instituicao)) :- prestador(ID, Nome, incEspecialidade1, Instituicao).
incEspecialidade(incEspecialidade1).
```

Figura 5 - Exemplo de conhecimento incerto

```
% O valor do cuidado X encontra-se entre 50 e 60 euros
excecao(cuidado((2018,3,29), 7, 6, 'Recolha de um eletrocardiograma', X)) :- X > 50, X < 60.
imprecisoCusto((2018,3,29),7,6).
```

Figura 6 - Exemplo de conhecimento impreciso

```
nuloInterditoMor(nuloMorada1).
excecao(utente(ID, Nome, Sexo, Idade, Morada)) :- utente(ID, Nome, Sexo, Idade, nuloMorada1).
utente(17, monica, 'F', 19, nuloMorada1).
```

Figura 7 - Exemplo de conhecimento interdito

- **Manipular invariantes que designem restrições à inserção e à remoção de conhecimento do sistema**

A manipulação de invariantes foi uma das tarefas mais desafiantes em todo o trabalho. Nesse sentido, os invariantes que construímos prendem-se essencialmente com a inserção de conhecimento.

O seguinte invariante, não permite adicionar conhecimento perfeito, quando já existe conhecimento perfeito acerca desse predicado. Assim, este apenas utiliza o predicado "não" para identificar que não se trata de conhecimento imperfeito, uma vez que, a inserção destes casos requer a utilização de "nulos" que os permite identificar. Para além disto, deverá ser possível adicionar idade/morada certa quando existe incerta. Este procedimento, foi adotado para cada um dos predicados contemplados no trabalho anterior.

```
+utente( ID, _, _, _, _ ) :: ( integer(ID),
                               findall( ID, (utente(ID, Nome, Sexo, Idade, Morada),
                               nao(incIdade(Idade)),nao(incMorada(Morada)),
                               nao(nuloInterditoIda(Idade)) , nao(nuloInterditoMor(Morada))  ), S),
                               comprimento( S, N ), N == 1 ).
```

Figura 8 - Invariante relativo a conhecimento perfeito

```
+utente( ID, _, _, Idade, _ ) :: ( nao(incIdade(Idade)), integer(ID),
                                   findall( ID, (utente(ID, Nome, Sexo, IdadeX, Morada), incIdade(IdadeX),nao(incMorada(Morada)),
                                   nao(nuloInterditoIda(IdadeX)) , nao(nuloInterditoMor(Morada))  ), S),
                                   comprimento( S, N ), N == 0 ).
```

Figura 9 - Invariante relativo à inserção de conhecimento incerto

Para além disto, não deverá ser possível a existência de conhecimento contraditório. Ora, este facto, é controlado através do uso de invariantes. De facto, a inclusão de conhecimento negativo nesta fase podia causar inconsistência.

```
% Invariantes que impedem a inserção de conhecimento contraditório
+(-utente(ID,Nome,Sexo,Idade,Morada)) :: (findall( (ID), utente(ID,NomeX,SexoX,IdadeX,MoradaX),S),
                                          comprimento(S,N), N == 0).

+(-prestador(ID, Nome, Especialidade, Instituicao)) :: (findall( (ID), prestador(ID, NomeX, EspecialidadeX, InstituicaoX),S),
                                          comprimento(S,N), N == 0).

+(-cuidado( Data, IdUt, IdPrest, Descricao, Custo )) :: (findall( (Data,IdUt,IdPrest,Descricao,Custo), cuidado( Data, IdUt, IdPrest, DescricaoX, CustoX ),S),
                                          comprimento(S,N), N == 0).
```

Figura 10 - Invariantes relativos à inserção de conhecimento contraditório

```
+(-cuidado(Data,IdUt,IdPrest,_,_)) :: nao(imprecisoCusto(Data,IdUt,IdPrest)).
```

Figura 11 - Invariante que impede a inserção de negativo existindo cuidados imprecisos

Adicionamos ainda um invariante que impede a inserção de exceções caso exista conhecimento perfeito relativo ao predicado que pretendemos inserir.

```
% Não se pode adicionar um excecao se for para conhecimento perfeito.
+excecao(T) :: nao(T).
```

Figura 12 - Invariante relativo à exceção

- **Lidar com a problemática da evolução do conhecimento, criando os procedimentos adequados**

Sendo que o conhecimento agora também pode ser imperfeito e negativo surge a necessidade da construção de novos predicados de evolução e consequentemente de involução. Construímos para cada tipo de conhecimento uma evolução/involucão diferente. Estas são distinguidas pelo nome que possuem.

- **Perfeito positivo:**

Relativamente ao conhecimento perfeito, a evolução deste teve de ser alterada. Efetivamente, consideramos que se existisse um conhecimento incerto relativo ao conhecimento a ser inserido este devia ser removido e substiuído pela informação que já

possuimos. Se existe um incerto e eventualmente surge uma informação que é perfeita, e que é certa, é necessário atualizar a base de conhecimento neste sentido. Este facto foi contemplado na contrução do predicado "evolucaoPerfeitoX", sendo X um dos seguintes nomes: Idade/Morada/Espec/Custo.

```
evolucaoPerfeitoIdade(utente(ID,Nome,Sexo,Idade,Morada)) :-
                        nao(existeIdadeIncerto(utente(ID,Nome,Sexo,Idade,Morada))),
                        existeIdadeIncerto2(utente(ID,Nome,Sexo,Idade,Morada),L),
                        involucaoIdInc(utente(ID,Nome,Sexo,L,Morada)),
                        evolucao(utente(ID,Nome,Sexo,Idade,Morada)).

evolucaoPerfeitoIdade(utente(ID,Nome,Sexo,Idade,Morada)) :-
                      existeIdadeIncerto(utente(ID,Nome,Sexo,Idade,Morada)),
                      evolucao(utente(ID,Nome,Sexo,Idade,Morada)).
```

Verificar a existência de conhecimento incerto antes de inserir

Figura 13 - Evolução de conhecimento perfeito positivo (exemplo idade)

- **Perfeito Negativo**

No que trata ao conhecimento negativo apenas construímos um predicado que pudesse inserir este tipo de conhecimento na base de conhecimento (não faz sentido a existência de predicado imperfeito relativamente a este). Ora este facto foi inserido graças à evolução que se apresenta na figura seguinte[1].

```
% Evolução do CONHECIMENTO PERFEITO NEGATIVO
evolucaoNeg(T) :- findall(I,+(-T)::I,Li),
                  teste(Li),
                  assert(-T).

involucaoNeg(T) :- findall(I,+(-T)::I,Li),
                   teste(Li),
                   retract(-T).
```

Figura 14 – Evolução e respetiva involução de conhecimento perfeito negativo

[1] Uma vez que o docente impossibilita o uso do operando cut(!), optamos por construir um predicado que o evite. No entanto, para o efeito, esta testa antes de inserir.

- **Conhecimento Incerto**

Uma vez que é necessária a inserção de conhecimento incerto relativa a cada predicado, decidimos implementar uma evolução que trate de todos os casos referentes a esta. A única inserção de conhecimento incerto que exigiu cuidado redobrado foi na adesão do conhecimento relativo ao custo. De facto, este pode ser incerto ou impreciso e é necessária atenção relativa a este caso. Consideramos que só pode ser acrescentado conhecimento incerto se não existir impreciso. Efetivamente, como já existe lá conhecimento impreciso, algum facto já é conhecido.

```
% ------------------------- Prestador
% Especialidade
evolucaoEspecInc(prestador(ID,Nome,Especialidade,Instituicao)) :-
          evolucao(prestador(ID,Nome,Especialidade,Instituicao)),
          assert( ( (excecao(prestador(Id,N,E,I))) :- prestador(Id,N,Especialidade,I) ) ),
          assert( incEspecialidade(Especialidade)).

involucaoEspecInc(prestador(ID,Nome,Especialidade,Instituicao)) :-
            involucao(prestador(ID,Nome,Especialidade,Instituicao)),
            retract( ( (excecao(prestador(Id,N,E,I))) :- prestador(Id,N,Especialidade,I) ) ),
            retract( incEspecialidade(Especialidade)).
```

Figura 15 – Evolução e respetiva involução de conhecimento incerto (exemplo instituição)

- **Conhecimento Impreciso**

De forma a não tornar o trabalho demasiado complexo, consideramos que apenas o custo poderá ser impreciso. No entanto, os restantos predicados imprecisos seriam semelhantes a este. O conhecimento impreciso relativamente ao custo pode ser de quatro tipos. Pode-se conhecer vários valores(lista), pode-se conhecer uma margem de valores(entre), pode-se conhecer até um certo valor(até) ou a partir de um valor(Apartir). Para além disso, consideramos que a inserção de conhecimento impreciso substitui o conhecimento incerto(de um mesmo predicado) existente na base de conhecimento.

```
% Custo [Lista]
evolucaoCustImp(cuidado(Data, IdUt,IdPrest, Descricao, Custo), L,lista) :-
              existeCustoImpreciso(Data,IdUt,IdPrest,Descricao,Custo),
              nao(existeCustoIncerto(cuidado(Data, IdUt,IdPrest, Descricao, Custo))),
              existeCustoIncerto2(cuidado(Data, IdUt,IdPrest, Descricao, Custo),Aux),
              involucaoCustInc(cuidado(Data,IdUt, IdPrest, Descricao, Aux)),
              auxImpEvolucao(cuidado(Data,IdUt, IdPrest, Descricao, Custo), L),
              assert(imprecisoCusto(Data,IdUt,IdPrest)).

evolucaoCustImp(cuidado(Data, IdUt,IdPrest, Descricao, Custo), L,lista) :-
                existeCustoImpreciso(Data,IdUt,IdPrest,Descricao,Custo),
                existeCustoIncerto(cuidado(Data, IdUt,IdPrest, Descricao, Custo)),
                auxImpEvolucao(cuidado(Data,IdUt, IdPrest, Descricao, Custo), L),
                assert(imprecisoCusto(Data,IdUt,IdPrest)).
```

Figura 16 - Evolução de conhecimento impreciso (exemplo custo lista)

```
involucaoCustImp(cuidado(Data, IdUt,IdPrest, Descricao, Custo), L,lista) :-
                auxImpInvolucao(cuidado(Data,IdUt, IdPrest, Descricao, Custo), L),
                retract(imprecisoCusto(Data,IdUt,IdPrest)).
```

Figura 17 - Involução de conhecimento impreciso (exemplo custo lista)

```
% Custo [Menor,Maior]
evolucaoCustImp(cuidado(Data, IdUt,IdPrest, Descricao, Custo), Menor,Maior,entre) :-
             findall(I,+(excecao(cuidado(Data,IdUt, IdPrest, Descricao, Custo)))::I,Li),
             teste(Li),
             nao(existeCustoIncerto(cuidado(Data, IdUt,IdPrest, Descricao, Custo))),
             existeCustoIncerto2(cuidado(Data, IdUt,IdPrest, Descricao, Custo),Aux),
             involucaoCustInc(cuidado(Data,IdUt, IdPrest, Descricao, Aux)),
             assert( (excecao(cuidado(Data,IdUt, IdPrest, Descricao, Custo)) :- Custo>Menor, Custo<Maior )),
             assert(imprecisoCusto(Data,IdUt,IdPrest)).

evolucaoCustImp(cuidado(Data, IdUt,IdPrest, Descricao, Custo), Menor,Maior,entre) :-
               findall(I,+(excecao(cuidado(Data,IdUt, IdPrest, Descricao, Custo)))::I,Li),
               teste(Li),
               existeCustoIncerto(cuidado(Data, IdUt,IdPrest, Descricao, Custo)),
               assert( (excecao(cuidado(Data,IdUt, IdPrest, Descricao, Custo)) :- Custo>Menor, Custo<Maior )),
               assert(imprecisoCusto(Data,IdUt,IdPrest)).

 involucaoCustImp(cuidado(Data, IdUt,IdPrest, Descricao, Custo), Menor,Maior,entre) :-
               findall(I,-(excecao(cuidado(Data,IdUt, IdPrest, Descricao, Custo)))::I,Li),
               teste(Li),
               retract( (excecao(cuidado(Data,IdUt, IdPrest, Descricao, Custo)) :- Custo>Menor, Custo<Maior )),
               retract(imprecisoCusto(Data,IdUt,IdPrest)).
```

Figura 18 - Evolução e involução de conhecimento impreciso (exemplo custo entre)

- **Conhecimento Interdito**

Relativamente ao conhecimento interdito, uma vez que se trata da inserção de conhecimento dinâmico a adesão deste na base de conhecimento implica, consequentemente, inserir um invariante. O nulo jamais poderá ser alterado. Apenas podem existir nulos interditos para a idade ou para a morada. Mais uma vez realçar que verificamos a existência de conhecimento incerto relativamente ao conhecimento que queremos inserir.

```
% Evolução do CONHECIMENTO INTERDITO
% Idade
evolucaoIdadeInt(utente(ID,Nome, Sexo, Idade, Morada)) :-
                existeIdadeIncerto(utente(ID,Nome, Sexo, Idade, Morada)),
                evolucao((utente(ID,Nome,Sexo,Idade,Morada)) ),
                assert((excecao(utente(I,N,S,I,M)) :- utente(I,N,S,Idade,M)) ),
                assert(nuloInterditoIda(Idade)),
                assert((+utente(Id,No,Se,Ida,Mo) :: (findall( (Id,No,Se,I,Mo),
                (utente(ID,Nome,Sexo,I,Morada),nao(nuloInterditoIda(I))), S),
                comprimento(S,N),
                N==0))
                ).
```

Inserção do respetivo invariante

Figura 19 - Evolução conhecimento interdito (exemplo idade)

- **Em suma:**

De modo a sintetizar as decisões que tomamos optamos por construir uma tabela. Esta apenas indica a forma como lidamos com os diversos tipos de conhecimento, na base de conhecimento, por nós idealizada, pelo que as justificações para as opções que foram tomadas encontram-se explicadas nos tópicos já apresentados.

Base de conhecimento	**Conhecimento a inserir**	**O que acontece**
Perfeito	Perfeito(repetido)	Utilização de um invariante que impede a inserção de conhecimento perfeito repetido
Perfeito	Incerto/Impreciso/Interdito	Utilização de um invariante que impede a inserção de conhecimento perfeito com parâmetros
Incerto	Incerto	Utilização de um invariante que impede a inserção de incerto se já existir incerto para esse predicado
	Perfeito	Tratamento deste caso, na evolução de conhecimento perfeito (se existir incerto deverá ser substituído pelo perfeito)
	Impreciso	Tratamento deste caso, na evolução de conhecimento impreciso (se existir incerto deverá ser substituído pelo impreciso)
	Interdito	Tratamento deste caso na evolução do conhecimento interdito (não é permitida a adesão de conhecimento interdito caso exista conhecimento incerto para esse predicado)

Interdito	Perfeito	Tratamento deste caso na evolução de conhecimento interdito (interdito não pode ser alterado, é adicionado um invariante)
	Impreciso	Tratamento deste caso, na evolução de conhecimento interdito (interdito não pode ser alterado, é adicionado um invariante)
	Incerto	Tratamento deste caso, na evolução de conhecimento interdito (impreciso não pode ser alterado, é adicionado um invariante)
Impreciso	Perfeito	Utilização de um invariante que impede a inserção de perfeito se já existir perfeito para esse predicado
	Incerto	Utilização de um invariante que impede a inserção de incerto se já existir impreciso para esse predicado
	Interdito	Utilização de um invariante que impede a inserção de interdito se já existir impreciso para esse predicado

Tabela 1 - Síntese de decisões tomadas

Pela análise da tabela é possível inferir que o grupo privilegiou a existência de conhecimento atualizado (por exemplo se existir conhecimento incerto e pretendermos inserir o mesmo predicado, no entanto, desta vez, este encontrar-se impreciso, deve substituir o incerto que se encontra na base de conhecimento).

- **Desenvolver um sistema de inferência capaz de implementar os mecanismos de raciocínio inerentes a estes sistemas**

Como já tínhamos referido neste projeto, utilizamos o predicado "demo" que para além de responder a uma pergunta com os valores de verdade habituais, acrescenta ainda um novo, o desconhecido. De modo a refinarmos este predicado, implementamos a possibilidade de ainda ser possível o teste de vários predicados através de uma lista. Assim, foi necessário definir o resultado que poderia ocorrem entre desconhecido **e/ou** os valores de verdade já existentes criando assim uma nova definição de **conjunção/disjunção**.

```
% Extensao do meta-predicado demoLista : Questoes, Resposta -> {V,F}
demoListaC([Q|[]], R) :- demo(Q,R).
demoListaC([Q|Qs],R) :-  demo(Q,R1),
                         demoListaC(Qs,R2),
                         conjuncaoDesc(R1,R2,R).

demoListaD([Q|[]], R) :- demo(Q,R).
demoListaD([Q|Qs],R)  :- demo(Q,R1),
                         demoListaD(Qs,R2),
                         disjuDesc(R1,R2,R).
```

Figura 20 -Alteração do predicado demo para o tratamento individual de conjunções e disjunções

```
% Extensao do predicado conjuncao: Q1, L2, R -> {V,F}
conjuncaoDesc(desconhecido, verdadeiro, desconhecido).
conjuncaoDesc(desconhecido, desconhecido, desconhecido).
conjuncaoDesc(desconhecido, falso, falso).
conjuncaoDesc(verdadeiro,verdadeiro,verdadeiro).
conjuncaoDesc(verdadeiro,desconhecido,desconhecido).
conjuncaoDesc(verdadeiro,falso,falso).
conjuncaoDesc(falso,verdadeiro,falso).
conjuncaoDesc(falso,desconhecido,falso).
conjuncaoDesc(falso,falso,falso).
```

Figura 21 - Adoções da conjunção do desconhecido com os valores de verdade já existentes

```
disjuDesc(verdadeiro, verdadeiro, verdadeiro).
disjuDesc(verdadeiro,desconhecido,verdadeiro).
disjuDesc(verdadeiro,falso,verdadeiro).
disjuDesc(desconhecido,verdadeiro,verdadeiro).
disjuDesc(desconhecido, desconhecido, desconhecido).
disjuDesc(desconhecido, falso, desconhecido).
disjuDesc(falso,verdadeiro,falso).
disjuDesc(falso,desconhecido,desconhecido).
disjuDesc(falso,falso,falso).
```

Figura 22 - Adoções da disjunção do desconhecido com os valores de verdade já existentes

```
demoFinal([Q],R) :- demo(Q,R).
demoFinal([Q, e | Ls], R) :- demo(Q,R1),
                             demoFinal(Ls,R2),
                             conjuncaoDesc(R1,R2,R).

demoFinal([Q],R) :- demo(Q,R).
demoFinal([Q, ou | Ls], R) :- demo(Q,R1),
                              demoFinal(Ls,R2),
                              disjuDesc(R1,R2,R).
```

Figura 23 - demo final que permite a utilização de conjunções e disjunções

4. Conclusões e sugestões

De forma geral consideramos que o objetivo do trabalho foi concluído. Nesse sentido, o projeto apresentado foi crucial para a consolidação de conceitos relativos ao conhecimento imperfeito e às variantes que este apresenta. Os desafios que foram surgindo prendiam-se com gestão da base de conhecimento da melhor forma, uma vez que foi necessário ter atenção em todos os casos que a poderiam tornar inconsistente. No entanto, consideramos que tomamos as decisões mais acertadas e que nos pareceram ser as mais coerentes no decorrer de todo o exercício prático.

Em suma, este trabalho formatou a nossa maneira de pensar aquando da resolução de um problema, tornando-nos mais objetivos e racionais. Foi assim possível um pensamento mais crítico relativamente a predicados que introduzem um conhecimento mais abrangente.

5. Referências

- Ivan Bratko, "PROLOG: Programming for Artificial Intelligence", 3rd Edition, Addison-Wesley Longman Publishing Co., Inc., 2000;

- Hélder Coelho, "A Inteligência Artificial em 25 lições", Fundação Calouste Gulbenkian, 1995;

- “Sugestões para a Redacção de Relatórios Técnicos, Cesar Analide, Paulo Novais, José Neves.

Anexos

I. Anexo 1 – Predicados auxiliares

```
% Invariante que resulta da adição de um custo incerto quando já existe imprecisos
existeCustoImpreciso(Data, IdUt, IdPrest, Descricao, Custo) :-  (findall((D,IdU,IdP), (imprecisoCusto(Data,IdUt,IdPrest)), L ),
                                                                comprimento(L,N),
                                                                N==0).

%Só pode existir um incerto para um determinado idade porque não vamos permitir a inserção de conhecimento repetido
existeIdadeIncerto(utente(ID,Nome, Sexo, Idade, Morada)) :- (findall( (Idade), (utente(ID,NomeX, SexoX, IdadeX, MoradaX), incIdade(IdadeX)),L),
                                                           comprimento(L,N),
                                                           N==0).

%Só pode existir um incerto para um determinada morada, porque não vamos permitir a inserção de conhecimento repetido
existeMoradaIncerto(utente(ID,Nome, Sexo, Idade, Morada)) :- (findall( (MoradaX), (utente(ID,NomeX, SexoX, IdadeX, MoradaX), incMorada(MoradaX)),L),
                                                           comprimento(L,N),
                                                           N==0).

%Só pode existir um incerto para um determinado valor, porque não vamos permitir a inserção de conhecimento repetido
existeCustoIncerto(cuidado(Data, IdUt,IdPrest, Descricao, Custo)) :- (findall(Custo,(cuidado(Data, IdUt,IdPrest, Descricao, CustoX), incCusto(CustoX)),L  ),
                                                                   comprimento(L,N),
                                                                   N==0).
```

```
existeCustoIncerto2(cuidado(Data, IdUt,IdPrest, Descricao, Custo), L) :- (findall( (CustoX), (cuidado(Data, IdUt,IdPrest, DescricaoX, CustoX),
                                                                          incCusto(CustoX)),[L|Ls])).
existeIdadeIncerto2(utente(ID,Nome, Sexo, Idade, Morada),L) :- (findall( (IdadeX ), (utente(ID,NomeX, SexoX, IdadeX, MoradaX),
                                                                          incIdade(IdadeX)),[L|Ls])).
existeMoradaIncerto2(utente(ID,Nome, Sexo, Idade, Morada),L) :- (findall( (MoradaX ), (utente(ID,NomeX, SexoX, IdadeX, MoradaX),
                                                                          incMorada(MoradaX)),[L|Ls])).
existeEspecIncerto2(prestador(ID, Nome, Especialidade, Instituicao),L) :- (findall( (EspecialidadeX), (prestador(ID,NomeX,EspecialidadeX,InstituicaoX),
                                                                          incEspecialidade(EspecialidadeX)),[L|Ls])).
```

```
% Evolução do CONHECIMENTO IMPRECISO
% [Auxiliares]
auxImpEvolucao(cuidado(D,IDu,IDp,Des,Cust),[Q|[]]) :- evolucao(excecao(cuidado(D,IDu,IDp,Des,Q))).
auxImpEvolucao(cuidado(D,IDu,IDp,Des,Cust),[Q|Qs]) :- evolucao(excecao(cuidado(D,IDu,IDp,Des,Q))),
                                                      auxImpEvolucao(cuidado(D,IDu,IDp,Des,Cust),Qs).

auxImpInvolucao(cuidado(D,IDu,IDp,Des,Cust),[Q|[]]) :- involucao(excecao(cuidado(D,IDu,IDp,Des,Q))).
auxImpInvolucao(cuidado(D,IDu,IDp,Des,Cust),[Q|Qs]) :- involucao(excecao(cuidado(D,IDu,IDp,Des,Q))),
                                                       auxImpInvolucao(cuidado(D,IDu,IDp,Des,Cust),Qs).
```

II. Anexo 2 – Resultados obtidos

```
| ?- listing(cuidado).
cuidado((2018,3,8), 1, 1, 'Remocao de um pequeno nodo', 2500).
cuidado((2018,3,8), 1, 2, 'Consulta de Rotina', 30).
cuidado((2018,3,10), 2, 3, 'Incisao nas costas sobre um furunculo', 150).
cuidado((2018,3,10), 3, 3, 'Remocao de pontos negros', 20).
cuidado((2018,3,10), 4, 3, 'Tratamento de alergia no joelho direito', 30).
cuidado((2018,3,28), 5, 3, 'Tratamento de alergia no joelho esquerdo', incValor1).

yes
| ?- listing(excecao).
excecao(utente(A,B,C,_,D)) :-
        utente(A, B, C, incIdade1, D).
excecao(utente(A,B,C,D,_)) :-
        utente(A, B, C, D, incMorada1).
excecao(prestador(A,B,_,C)) :-
        prestador(A, B, incEspecialidade1, C).
excecao(cuidado(A,B,C,D,_)) :-
        cuidado(A, B, C, D, incValor1).
excecao(cuidado((2018,3,29),7,6,'Recolha de um eletrocardiograma',A)) :-
        A>50,
        A<60.
excecao(utente(A,B,C,_,D)) :-
        utente(A, B, C, nuloIdade1, D).
excecao(utente(A,B,C,D,_)) :-
        utente(A, B, C, D, nuloMorada1).

yes
| ?- evolucaoCustImp(cuidado((2018,3,28),5,3,'Tratamento de alergia no joelho esquerdo', impCusto10),[10,20,30],lista).
yes
| ?- listing(cuidado).
cuidado((2018,3,8), 1, 1, 'Remocao de um pequeno nodo', 2500).
cuidado((2018,3,8), 1, 2, 'Consulta de Rotina', 30).
cuidado((2018,3,10), 2, 3, 'Incisao nas costas sobre um furunculo', 150).
cuidado((2018,3,10), 3, 3, 'Remocao de pontos negros', 20).
cuidado((2018,3,10), 4, 3, 'Tratamento de alergia no joelho direito', 30).

yes
| ?- listing(excecao).
excecao(utente(A,B,C,_,D)) :-
        utente(A, B, C, incIdade1, D).
excecao(utente(A,B,C,D,_)) :-
        utente(A, B, C, D, incMorada1).
excecao(prestador(A,B,_,C)) :-
        prestador(A, B, incEspecialidade1, C).
excecao(cuidado((2018,3,29),7,6,'Recolha de um eletrocardiograma',A)) :-
        A>50,
        A<60.
excecao(utente(A,B,C,_,D)) :-
        utente(A, B, C, nuloIdade1, D).
excecao(utente(A,B,C,D,_)) :-
        utente(A, B, C, D, nuloMorada1).
excecao(cuidado((2018,3,28),5,3,'Tratamento de alergia no joelho esquerdo',10)).
excecao(cuidado((2018,3,28),5,3,'Tratamento de alergia no joelho esquerdo',20)).
excecao(cuidado((2018,3,28),5,3,'Tratamento de alergia no joelho esquerdo',30)).
```

```
| ?- listing(utente).
utente(1, ana, 'F', 15, 'Aveiro').
utente(2, bruno, 'M', 16, 'Maia').
utente(3, catarina, 'F', 18, 'Maia').
utente(4, diogo, 'M', 18, 'Porto').
utente(5, eduardo, 'M', 19, 'Coimbra').
utente(6, filipe, 'M', 20, 'Cascais').
utente(7, hugo, 'M', 25, 'Alpendorada').
utente(8, isabel, 'F', 30, 'Magrelos').
utente(9, joao, 'M', 38, 'Magrelos').
utente(10, luis, 'M', 42, 'Santo Tirso').
utente(14, margarida, 'F', incIdade1, 'Famalicao').
utente(15, patricia, 'F', 18, incMorada1).
utente(16, guilherme, 'M', nuloIdade1, 'Viana de Castelo').
utente(17, monica, 'F', 19, nuloMorada1).

yes
| ?- evolucaoPerfeitoIdade(utente(14,margarida,'F',22,'Famalicao')).
yes
| ?- listing(utente).
utente(1, ana, 'F', 15, 'Aveiro').
utente(2, bruno, 'M', 16, 'Maia').
utente(3, catarina, 'F', 18, 'Maia').
utente(4, diogo, 'M', 18, 'Porto').
utente(5, eduardo, 'M', 19, 'Coimbra').
utente(6, filipe, 'M', 20, 'Cascais').
utente(7, hugo, 'M', 25, 'Alpendorada').
utente(8, isabel, 'F', 30, 'Magrelos').
utente(9, joao, 'M', 38, 'Magrelos').
utente(10, luis, 'M', 42, 'Santo Tirso').
utente(15, patricia, 'F', 18, incMorada1).
utente(16, guilherme, 'M', nuloIdade1, 'Viana de Castelo').
utente(17, monica, 'F', 19, nuloMorada1).
utente(14, margarida, 'F', 22, 'Famalicao').

yes
| ?-
```

```
| ?- demoFinal([utente(1,ana,'F',15,'Aveiro'),e,utente(11,miguel,'M',22,'Mafra'
),ou,prestador(1,manuel,'Dermatologia','Santa Maria')],R).
R = falso ?
yes
| ?-
```