UNIVERSIDADE DO MINHO

Health Care System

Mestrado Integrado de Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio
(2ºSemestre/2017-2018)

Número	Nome do(s) Autor(es)
a78468	João Vieira
a78821	José Martins
a77049	Miguel Quaresma
a77689	Simão Barbosa

Braga, Portugal
19 de Abril de 2018

1 Resumo

O objetivo deste projeto é criar um sistema de representação de conhecimento e raciocínio na caraterização de um universo na área da prestação de cuidados de saúde. É pretendido nesta fase de trabalho que se represente diferentes tipos de conhecimento, tanto positivo como negativo, e que cubra também casos de conhecimento imperfeito, utilizando valores nulos de todos os tipos estudados. Pretende-se também que se manipule invariantes que restringem tanto a inserção como a remoção de conhecimento do sistema, que se lide com o problema da evolução de conhecimento na base de representação do conhecimento tomando as decisões necessárias para tal, e, para além disto, que seja desenvolvido um sistema de inferência que possa implementar mecanismos de raciocínio inerentes a estes sistemas.

Conteúdo

1 Resumo **2**

2 Introdução **4**

3 Descrição do Trabalho e Análise de Resultados **5**
3.1 Base de Conhecimento 5
3.2 Funcionalidades 5
3.3 Representação de Conhecimento 6
3.3.1 Conhecimento positivo e negativo 6
3.3.2 Conhecimento imperfeito 6
3.3.3 Invariantes 7
3.3.4 Funções Auxiliares 10
3.4 Evolução com aprendizagem 11
3.5 Demonstrador de conjunções e disjunções 12

4 Conclusões e Sugestões **13**

2 Introdução

Health Care System é um Sistema de Representação de Conhecimento e Raciocínio usado na caraterização de um universo na área da prestação de cuidados de saúde desenvolvido usando a linguagem de programação em lógica PROLOG. A representação de conhecimento é bastante importante visto permitir-nos obter respostas a perguntas, não só de conhecimento positivo mas também de conhecimento negativo e desconhecido. Para além disso, por forma a vir a ser possivel a existência de inteligência artificial é essencial que a Base de Conhecimento possa evoluir/aprender, ou seja, por vezes um conhecimento já adquirido pode estar incorreto e é substituido por um conhecimento mais recente. Será apresentado como tal é possivel de seguida.

3 Descrição do Trabalho e Análise de Resultados

3.1 Base de Conhecimento

A base de conhecimento do sistema desenvolvido é essencial à representação do conhecimento e raciocínio, como tal, tendo em conta o sistema em questão, foram desenvolvidos os seguintes predicados:

- utente: #IdUt, Nome, Idade, Morada → {V,F}
- prestador: #IdPrest, Nome, Especialidade, Instituição → {V,F}
- cuidado: Data, #IdUt, #IdPrest, Descrição, Custo → {V,F}
- instituicao: #IdIt, Nome, Tipo, Cidade → {V,F}

Para além disso foram contruídos regras da negação dos predicados de modo a que quando não exista o facto negativo de uma demonstração seja possível obter resposta, assumindo o pressuposto do mundo fechado, sendo que caso o facto exista será assumida a negação forte. Estes predicados são os seguintes:

```
% utente: #IdUt, Nome, Idade, Morada -> {V,F}
-utente(IdUt, N, I, M):- nao(utente(IdUt,N,I,M)),
                         nao(excecao(utente(IdUt,N,I,M))).

% prestador: #IdPrest, Nome, Especialidade, Instituição -> {V,F}
-prestador(IdPrest, N, E, I):- nao(prestador(IdPrest,N,E,I)),
                               nao(excecao(prestador(IdPrest,N,E,I))).

% cuidado: Data, #IdUt, #IdPrest, Descrição, Custo -> {V,F}
-cuidado(D, IdUt, IdPrest, Desc, C):- nao(cuidado(D,IdUt,IdPrest,Desc,C)),
                                      nao(excecao(cuidado(D,IdUt,IdPrest,Desc,C))).

% instituicao: #IdIt, Nome, Tipo, Cidade -> {V,F}
-instituicao(IdIt, N, T, C):- nao(instituicao(IdIt,N,T,C)),
                         nao(excecao(instituicao(IdIt,N,T,C))).
```

3.2 Funcionalidades

As regras de uma base de conhecimento conferem-lhe a utilidade necessária ao seu funcionamento visto que, se não for possível efetuar questões e obter respostas às mesmas, a utilidade da base de conhecimento é nula. Em PROLOG estas funcionalidades/regras são implementadas tendo por base as caraterísticas desta linguagem:

- Algoritmo de Resolução: mais propriamente o *Modus Tollens* ({A se B,¬A}⊢ ¬B) no qual uma questão é verdade se adicionando a negação da mesma à base de conhecimento origina uma inconsistência
- Clausulado de Horn: todas as regras são uma cláusula de Horn **i.e.** o qual admite apenas 1 termo positivo(conclusão), as fórmulas são bem formadas e fechadas, quantificadas universalmente (**e.g**. $\forall A, B$ avo(A,B)⊢neto(B,A))
- Mecanismo de *backtracking*: na presença de uma solução inválida(falsa) esta meta-heurística continua à procura de "outro caminho"de modo a que a regra seja verdadeira.

Adicionalmente assumem-se vários pressupostos de entre os quais:

- Pressuposto dos Nomes Únicos: duas constantes designam duas entidades diferentes
- Pressuposto do Domínio Fechado: não há mais objetos no universo para além dos designados por constantes

Será contudo abandonado o pressuposto do Mundo Fechado, introduzindo a Negação Forte, referida anteriormente. A introdução da Negação Forte permite a representação de conhecimento negativo, algo que usando o pressuposto do Mundo Fechado não seria possível. Por fim o sistema de representação de conhecimento e raciocínio desenvolvido deve respeitar as "leis"da lógica recorrendo para isso a invariantes que garantem que certas propriedades são respeitadas. Estes (invariantes) garantem a inexistência de inconsistências bem como a preservação do "significado do conhecimento"e são usados na evolução e involução da base de conhecimento.

3.3 Representação de Conhecimento

3.3.1 Conhecimento positivo e negativo

Neste segundo exercício, um dos objetivos do mesmo e obrigatoriamente necessário para testar toda a base de conhecimentos pretendida com o trabalho é pela representação de conhecimento dos diferentes tipos possíveis.
No que diz respeito a conhecimento positivo é possível observar na nossa implementação dados como:

- Representar o utente com o id 1, chamado João, que tem 23 anos e é da cidade do Porto.

  ```
  utente(1,joao,23,porto).
  ```

- Representar o prestador de serviços José, cuja sua especialidade é a Pediatria e que pertence ao Hospital de Braga, cujo número 1 o representa no sistema.

  ```
  prestador(1,jose,pediatria,hospital_Braga).
  ```

- Representar o cuidado efetuado no dia 23 de Maio de 2016 pelo prestador 1 ao utente 1 cujo custo do serviço pela consulta foi de 20 euros.

  ```
  cuidado(2016-05-23,1,1,consulta,20).
  ```

- Representar a instituição com o id 1, do tipo Hospital localizada na cidade de Braga cujo seu nome é Hospital de Braga.

  ```
  instituicao(1,hospital_Braga,hospital,braga).
  ```

3.3.2 Conhecimento imperfeito

Já no que diz respeito a conhecimento imperfeito, foram representados pelo grupo todos os tipos da utilização de valores nulos estudados.
Quanto aos valores nulos do tipo incerto (desconhecido) foram desenvolvidos e implementados sob a forma de conhecimento os seguintes enunciados criados pelo grupo:

- A utente Madalena tem 54 anos de idade e detém o id 22, no entanto é desconhecida a sua morada.

  ```
  utente(22,madalena,54,xpto1).
  excecao(utente(IdU,N,I,M)):- utente(IdU,N,I,xpto1).
  ```

- É sabido que existe um prestador com o id 23, ortopedia como sua especialidade e pertencente ao Hospital de Braga, ainda que não se conheça o seu nome.

- O utente com o id 3 recebeu um cuidado do prestador 3 no dia 23 de Maio de 2017 tendo como custo 20 euros, desconhece-se a descrição do serviço prestado.

- A Clinica Fernandes encontra-se representada na base de conhecimento pelo número 24, não sendo conhecida a cidade em que esta clínica está inserida.

Nos valores nulos do tipo impreciso (desconhecido, mas de um conjunto determinado de hipóteses), foram tidos em conta as seguintes premissas:

- A Mariana é de Leiria e encontra-se representada na base de conhecimento pelo número 25, sabe-se que a sua idade encontra-se entre os 18 e os 34 anos.
- O prestador com o id 26 tem como especialidade Enfermagem e trabalha no Hospital de Coimbra, não se sabe se o seu nome é Rute ou Rita.

```
excecao(prestador(26,rute,enfermagem,hospital_Coimbra)).
excecao(prestador(26,rita,enfermagem,hospital_Coimbra)).
```

- A consulta realizada no dia 2 de Agosto de 2017 pelo prestador 4 ao utente 2 teve um custo no intervalo [10,24] euros.
- Não se sabe se a Clínica Armandes com o id 27 é uma clínica de Lisboa ou de Setúbal.

Por último, faltam apenas os valores nulos do tipo interdito, que foram representados tendo em conta os enunciados:

- O utente representado pelo número 28 tem como seu nome Zulmira e tem 45 anos, no entanto, ninguém pode conhecer a sua morada por vontade da mesma.
- O Centro Carandá tem ao seu dispor um prestador com especialidade em Medicina Geral e cujo seu número no sistema é o 26, este Centro não revela o nome do mesmo de forma a proteger os seus melhores prestadores.
- Calcula-se que o maior custo alguma vez praticado por um cuidado tenha sido realizado no dia 17 de Outubro de 2017 pelo prestador 4 ao utente 5 com a descrição "consulta"; face à polémica instalada o custo da mesma encontra-se completamente em segredo.

```
cuidado(2017-10-17,5,4,consulta,xpto7).
excecao(cuidado(D,IdU,IdP,Desc,C)):- cuidado(D,IdU,IdP,Desc,xpto7).
nulo(xpto7).
+cuidado(D,IdU,IdP,Desc,C)::
            (solucoes(C,(cuidado(2017-10-17,5,4,consulta,C),nao(nulo(C))),L),
             len(L,N),
             N==0).
```

- A Clínica Antunes abriu à pouco tempo e é representada na base de conhecimento pelo número 30, no entanto, a administração desta clínica prefere que pelo menos por enquanto a cidade onde esta se encontra seja interdita e impossível de conhecer.

3.3.3 Invariantes

Com a implementação de certos invariantes considerados como importantes para o contexto em causa, a base de dados de conhecimento em PROLOG fica assim mais coerente e menos suscetível a falhas. Foram atualizados os invariantes do trabalho anterior, devido à adição de conhecimento negativo e desconhecido à Base de Conhecimento. Como tal, decidimos que ao realizar a evolução de algo só será permitida a mesma desde que não exista repetido, não exista a negação da mesma e não exista a presença de exceções que indica a presença de conhecimento desconhecido do mesmo. Tendo isto em conta, foram alterados alguns invariantes do trabalho anterior e implementados novos invariantes que:

```
%Impede a evolução do utente(positivo) caso já exista conhecimento positivo, negativo
ou desconhecido com o mesmo Id
+utente(Id,Nome,Idade,Morada)::(solucoes((X,Y,Z),utente(Id,X,Y,Z),S1),
                                solucoes((X,Y,Z),-utente(Id,X,Y,Z),S2),
                                solucoes((X,Y,Z),excecao(utente(Id,X,Y,Z)),S3),
                                len(S1,N1),
                                len(S2,N2),
                                len(S3,N3),
                                N is N1 + N2 + N3,
```

```
                                N=<1).

%Impede a evolução do prestador(positivo) caso já exista conhecimento positivo, negativo
ou desconhecido com o mesmo Id
+prestador(Id,Nome,Esp,Inst)::(solucoes((X,Y,Z),prestador(Id,X,Y,Z),S1),
                               solucoes((X,Y,Z),-prestador(Id,X,Y,Z),S2),
                               solucoes((X,Y,Z),excecao(prestador(Id,X,Y,Z)),S3),
                               len(S1,N1),
                               len(S2,N2),
                               len(S3,N3),
                               N is N1 + N2 + N3,
                               N=<1).

%Prestador pertence a instituição válida
+prestador(Id,Nome,Esp,Inst)::
            (solucoes((IdI,TipoI,CidadeI),instituicao(IdI,Inst,TipoI,CidadeI),S),
            len(S,N),
            N>=1).

%Impede a evolução do cuidado(positivo) caso já exista conhecimento positivo, negativo
ou desconhecido
+cuidado(Data,IdU,IdP,Desc,Custo)::
        (solucoes((Data,IdU,IdP,Desc,Custo),cuidado(Data,IdU,IdP,Desc,Custo),S1),
        solucoes((Data,IdU,IdP,Desc,Custo),-cuidado(Data,IdU,IdP,Desc,Custo),S2),
        solucoes((Data,IdU,IdP,Desc,Custo),excecao(cuidado(Data,IdU,IdP,Desc,Custo)),S3),
        len(S1,N1),
        len(S2,N2),
        len(S3,N3),
        N is N1 + N2 + N3,
        N=<1).

%Utente existe
+cuidado(Data,IdU,IdP,Desc,Custo)::(solucoes((X,Y,Z),utente(IdU,X,Y,Z),S),
                                    len(S,N),
                                    N>=1).

%Prestador existe
+cuidado(Data,IdU,IdP,Desc,Custo)::(solucoes((X,Y,Z),prestador(IdP,X,Y,Z),S),
                                    len(S,N),
                                    N>=1).

%Impede a evolução da instituição(positivo) caso já exista conhecimento positivo, negativo
ou desconhecido com o mesmo Id
+instituicao(Id,Nome,Tipo,Cidade)::
(solucoes((Nome,Tipo,Cidade),instituicao(Id,Nome,Tipo,Cidade),S1),
        solucoes((Nome,Tipo,Cidade),-instituicao(Id,Nome,Tipo,Cidade),S2),
        solucoes((Nome,Tipo,Cidade),excecao(instituicao(Id,Nome,Tipo,Cidade)),S3),
        len(S1,N1),
        len(S2,N2),
        len(S3,N3),
        N is N1 + N2 + N3,
        N=<1).

%Não existem cuidados referentes a utente
-utente(Id,Nome,Idade,Morada)::
                    (solucoes((Data,IdP,Desc,Custo),cuidado(Data,Id,IdP,Desc,Custo),S),
                    len(S,N),
```

```
                        N==0).

%Não existem cuidados referentes a prestador
-prestador(Id,Nome,Esp,Inst)::(solucoes((Data,IdU,Desc,Custo),cuidado(Data,IdU,Id,Desc,Custo),S),
                                len(S,N),
                                N==0).

%Não existem prestadores registados nesta instituição
-instituicao(Id,Nome,Tipo,Cidade)::(solucoes((IdP,NomeP,Esp),prestador(IdP,NomeP,Esp,Nome),S),
                                    len(S,N),
                                    N==0).

%Impede a evolução do utente(negativo) caso já exista conhecimento positivo, negativo
ou desconhecido com o mesmo Id
+(-utente(Id,Nome,Idade,Morada))::(solucoes((X,Y,Z),utente(Id,X,Y,Z),S1),
                                   solucoes((X,Y,Z),-utente(Id,X,Y,Z),S2),
                                   solucoes((X,Y,Z),excecao(utente(Id,X,Y,Z)),S3),
                                   len(S1,N1),
                                   len(S2,N2),
                                   len(S3,N3),
                                   N is N1 + N2 + N3,
                                   N=<1).

%Impede a evolução do prestador(negativo) caso já exista conhecimento positivo, negativo
ou desconhecido com o mesmo Id
+(-prestador(Id,Nome,Esp,Inst))::(solucoes((X,Y,Z),prestador(Id,X,Y,Z),S1),
                                  solucoes((X,Y,Z),-prestador(Id,X,Y,Z),S2),
                                  solucoes((X,Y,Z),excecao(prestador(Id,X,Y,Z)),S3),
                                  len(S1,N1),
                                  len(S2,N2),
                                  len(S3,N3),
                                  N is N1 + N2 + N3,
                                  N=<1).

%Impede a evolução do cuidado(negativo) caso já exista conhecimento positivo, negativo
ou desconhecido
+(-cuidado(Data,IdU,IdP,Desc,Custo))::
           (solucoes((Data,IdU,IdP,Desc,Custo),cuidado(Data,IdU,IdP,Desc,Custo),S1),
            solucoes((Data,IdU,IdP,Desc,Custo),-cuidado(Data,IdU,IdP,Desc,Custo),S2),
            solucoes((Data,IdU,IdP,Desc,Custo),excecao(cuidado(Data,IdU,IdP,Desc,Custo)),S3),
            len(S1,N1),
            len(S2,N2),
            len(S3,N3),
            N is N1 + N2 + N3,
            N=<1).

%Impede a evolução do instituicao(negativo) caso já exista conhecimento positivo, negativo
ou desconhecido com o mesmo Id
+(-instituicao(Id,Nome,Tipo,Cidade))::
            (solucoes((Nome,Tipo,Cidade),instituicao(Id,Nome,Tipo,Cidade),S1),
            solucoes((Nome,Tipo,Cidade),-instituicao(Id,Nome,Tipo,Cidade),S2),
            solucoes((Nome,Tipo,Cidade),excecao(instituicao(Id,Nome,Tipo,Cidade),S3)),
            len(S1,N1),
            len(S2,N2),
            len(S3,N3),
            N is N1 + N2 + N3,
            N=<1).
```

```
%Impede a evolução de exceção utente(desconhecido) caso já exista conhecimento positivo,
negativo com o mesmo Id
+excecao(utente(Id,Nome,Idade,Morada))::(solucoes((X,Y,Z),utente(Id,X,Y,Z),S1),
                                         solucoes((X,Y,Z),-utente(Id,X,Y,Z),S2),
                                         len(S1,N1),
                                         len(S2,N2),
                                         N is N1 + N2,
                                         N=<1).

%Impede a evolução de exceção prestador(desconhecido) caso já exista conhecimento positivo,
negativo com o mesmo Id
+excecao(prestador(Id,Nome,Esp,Inst))::(solucoes((X,Y,Z),prestador(Id,X,Y,Z),S1),
                                        solucoes((X,Y,Z),-prestador(Id,X,Y,Z),S2),
                                        len(S1,N1),
                                        len(S2,N2),
                                        N is N1 + N2,
                                        N=<1).

%Impede a evolução de exceção cuidado(desconhecido) caso já exista conhecimento positivo
ou negativo
+excecao(cuidado(Data,IdU,IdP,Desc,Custo))::
               (solucoes((Data,IdU,IdP,Desc,Custo),cuidado(Data,IdU,IdP,Desc,Custo),S1),
               solucoes((Data,IdU,IdP,Desc,Custo),-cuidado(Data,IdU,IdP,Desc,Custo),S2),
               len(S1,N1),
               len(S2,N2),
               N is N1 + N2,
               N=<1).

%Impede a evolução de exceção instituicao(desconhecido) caso já exista conhecimento positivo
ou negativo com o mesmo Id
+excecao(instituicao(Id,Nome,Tipo,Cidade))::
               (solucoes((Nome,Tipo,Cidade),instituicao(Id,Nome,Tipo,Cidade),S1),
               solucoes((Nome,Tipo,Cidade),-instituicao(Id,Nome,Tipo,Cidade),S2),
               len(S1,N1),
               len(S2,N2),
               N is N1 + N2,
               N=<1).
```

3.3.4 Funções Auxiliares

O desenvolvimento do sistema em causa envolveu, por vezes, o uso de regras que partilhavam certas operações entre si, como tal estas operações foram degeneradas em regras individuais por forma a reduzir a quantidade de código necessária. As operações referidas encontram-se descritas de seguida:

- Encontra todos os predicados(Questão) que sejam satisfeitos ao efetuar o *backtracking* tendo Formato em conta
  solucoes : Formato, Questao, Soluçoes → {V,F}
- Função para somar uma lista
  inserir: Termo → {V,F}
- Remover conhecimento
  remover: Termo → {V,F}
- Regra de teste dos invariantes correspondentes
  test: Lista → {V,F}
- remove se existe se não existir passa ao seguinte da lista de factos
  removeL: Lista → {V,F}

3.4 Evolução com aprendizagem

De modo a aprender conhecimento devemos ao adicionar algo garantir que se existir conhecimento positivo, negativo ou desconhecido que gere incoerência na Base de Conhecimento seja removido antes de adicionar o conhecimento mais recente. Contudo achamos que caso seja desconhecido interdito o conhecimento não deve ser possível trocar e como tal, só é possivel nos restantes casos. Para tal criamos a função *evolucaoLearn*. Por exemplo, o evolucaoLearn para o predicado utente é da seguinte forma:

```
evolucaoLearn(utente(IdU,Nome,Idade,Morada)):-
                        solucoes(utente(IdU,N,I,M),utente(IdU,N,I,M),L1),
                        naoNuloL(L1),
                        solucoes(-utente(IdU,N2,I2,M2),-utente(IdU,N2,I2,M2),L2),
                        solucoes(excecao(utente(IdU,N3,I3,M3)),excecao(utente(IdU,N3,I3,M3)),L3),
                        removeL(L1),
                        removeL(L2),
                        removeL(L3),
                        inserir(utente(IdU,Nome,Idade,Morada)).

evolucaoLearn(-utente(IdU,Nome,Idade,Morada)):-
                        solucoes(utente(IdU,N,I,M),utente(IdU,N,I,M),L1),
                        naoNuloL(L1),
                        solucoes(-utente(IdU,N2,I2,M2),-utente(IdU,N2,I2,M2),L2),
                        solucoes(excecao(utente(IdU,N3,I3,M3)),excecao(utente(IdU,N3,I3,M3)),L3),
                        removeL(L1),
                        removeL(L2),
                        removeL(L3),
                        inserir(-utente(IdU,Nome,Idade,Morada)).

evolucaoLearn(excecao(utente(IdU,Nome,Idade,Morada))):-
                        solucoes(utente(IdU,N,I,M),utente(IdU,N,I,M),L1),
                        naoNuloL(L1),
                        solucoes(-utente(IdU,N2,I2,M2),-utente(IdU,N2,I2,M2),L2),
                        removeL(L1),
                        removeL(L2),
                        inserir(excecao(utente(IdU,Nome,Idade,Morada))).
```

Sendo naoNuloL usado de modo a garantir que não se remove factos de conhecimento interdito:

```
naoNulo(X,Y,Z):-
    nao(nulo(X)),
    nao(nulo(Y)),
    nao(nulo(Z)).

naoNuloL([]).
naoNuloL([utente(IdU,N,I,M)|T]):-naoNulo(N,I,M), naoNuloL(T).
naoNuloL([-utente(IdU,N,I,M)|T]):-naoNulo(N,I,M), naoNuloL(T).
naoNuloL([excecao(utente(IdU,N,I,M))|T]):-naoNulo(N,I,M), naoNuloL(T).
```

Para os restantes predicados o evolucaoLearn é bastante parecido trocando na maioria apenas o utente pelo predicado referente, sendo que no naoNuloL serão adicionadas 3 linhas bastante parecidas com as 3 ultimas acima indicadas em que apenas troca utente pelo predicado e caso possua mais parâmetros do que 3 usa-se naoNulo mais do que uma vez ou nao(nulo(P)) de forma "direta". De modo a adicionar conhecimento interdito adicionamos a possibilidade de adicionar nulo(P), garantindo que não há duplicação:

```
evolucaoLearn(nulo(T)):-
        nao(nulo(T)),
        inserir(nulo(T)).
```

Também pra permitir conhecimento interdito bem como incerto e impreciso adicionamos mais duas funções, contudo as mesmas não garantem que não há repetição e como tal o uso das mesmas deve ser realizado com muito cuidado e algo que deveria futuramente corrigido. A primeira permite a inserção de regras de exceção e a segunda permite a inserção de invariantes.

```
%WARNING - permite duplicação, usar com muito cuidado, permite a inserçao de regras de exceção
%evolucaoLearnExc: excecao(P),Q -> {V,F}
evolucaoLearnExc(excecao(P),Q):-
        assert((excecao(P):-Q)).

%WARNING - permite duplicação, usar com muito cuidado, permite a inserçao de invariantes
%evolucaoLearnI: Termo, Invariante -> {V,F}
evolucaoLearnI(T,I):-
        assert(T::I).
```

3.5 Demonstrador de conjunções e disjunções

De modo a demonstrar que um conjunto de disjunções e/ou de conjunções tem o valor verdadeiro, falso ou desconhecido é necessário criar um novo demonstrador visto que o *demo* apenas permite a demonstração de uma questão de cada vez. Implementou-se então o demonstrador *demoComp*:

```
demoComp((Q1,Q2),R):-
    demo(Q1,R1),
    demoComp(Q2,R2),
    conjuncao(R1,R2,R).
demoComp((Q1;Q2),R):-
    demo(Q1,R1),
    demoComp(Q2,R2),
    disjuncao(R1,R2,R).
demoComp(QS,R):- demo(QS,R).
```

Fica assim por definir as regras *conjuncao* e *disjuncao*. Consideramos que os valores de verdade consoante o caso deviam ser:

conjuncao	V	D	F
V	V	D	F
D	D	D	F
F	F	F	F

disjuncao	V	D	F
V	V	V	V
D	V	D	D
F	V	D	F

Tendo isso em conta as regras são:

```
conjuncao(R1,R2,verdadeiro):- R1==verdadeiro, R2==verdadeiro.
conjuncao(R1,R2,falso):- R1==falso.
conjuncao(R1,R2,falso):- R2==falso.
conjuncao(R1,R2,desconhecido):- R1==desconhecido, R2==verdadeiro.
conjuncao(R1,R2,desconhecido):- R1==verdadeiro, R2==desconhecido.
conjuncao(R1,R2,desconhecido):- R1==desconhecido, R2==desconhecido.

disjuncao(R1,R2,falso):- R1==falso, R2==falso.
disjuncao(R1,R2,verdadeiro):- R1==verdadeiro.
disjuncao(R1,R2,verdadeiro):- R2==verdadeiro.
disjuncao(R1,R2,desconhecido):- R1==desconhecido, R2==falso.
disjuncao(R1,R2,desconhecido):- R1==falso, R2==desconhecido.
disjuncao(R1,R2,desconhecido):- R1==desconhecido, R2==desconhecido.
```

4 Conclusões e Sugestões

Em jeito de conclusão, é da opinião do grupo de trabalho que o sistema de conhecimento obtido nesta fase do projeto está de acordo com os objetivos pretendidos neste exercício. Uma das principais dificuldades encontradas no desenvolvimento deste sistema passou pelo tratamento e distinção entre os diferentes tipos de conhecimento (positivo,negativo e imperfeito) que podem sofrer evoluções na base de conhecimento, no qual é necessário perceber quais os dados a remover desta base de conhecimento para não ser possível existir quaisquer tipos de incoerências na mesma. No que diz respeito a melhorias, a base de conhecimento seria mais completa caso:

- A função *evolucaoLearn()* responsável por introduzir e remover conhecimentos fosse capaz de lidar e manipular conhecimento imperfeito do tipo valor nulo impreciso que inclua intervalos de valores considerados como desconhecidos, por exemplo:

```
excecao(utente(25,mariana,I,leiria)):- I>=18, I=<20. % não suportado

excecao(utente(25,mariana,18,leiria)).              % suportado
excecao(utente(25,mariana,19,leiria)).
excecao(utente(25,mariana,20,leiria)).
```

- tanto a função *evolucaoLearnExc()* como a funcão *evolucaoLearnI()* que permitem respetivamente a inserção de regras de exceção e de invariantes na base de conhecimento, fossem métodos mais seguros e que não permitissem duplicação de conhecimento, pelo que devem ser utilizadas pelo utilizador com grande cautela e sensatez.

Referências

[1] MARTINS, José Carlos Lima, *Notas Teóricas*, José Carlos Lima Martins, 2018.

[2] VIEIRA, João Pedro Ferreira, *Notas Teóricas*, João Pedro Ferreira Vieira, 2018.

[3] QUARESMA, Miguel Miranda, *Notas Teóricas*, Miguel Miranda Quaresma, 2018.

[4] BARBOSA, Simão Paulo Leal, *Notas Teóricas*, Simão Paulo Leal Barbosa, 2018.