Sistemas de Representação de Conhecimento e Raciocínio

Exercício 1

Relatório de Desenvolvimento

Grupo 18

Cesário Miguel Pereira Perneta - A73883
Luís Miguel Bravo Ferraz - A70824
Tiago Miguel Fraga Santos - A74092
Rui Pedro Barbosa Rodrigues - A74572

17 de Março de 2018

Resumo

Neste relatório irá ser abordado a resolução e verificação do primeiro trabalho prático da Unidade Curricular de Sistemas de Representação de Conhecimento e Raciocínio. Será possível encontrar uma detalhada explicação dos predicados elaborados, bem como a apresentação do código usado na representação dos predicados fundamentais, que visam dar resposta aos requisitos propostos neste trabalho prático.

Conteúdo

Capítulo 1

Introdução

Como primeiro exercício desta unidade curricular, temos como objectivo desenvolver um sistema de representação de conhecimento e raciocínio que tenha a capacidade de caracterizar um universo de discurso na área da prestação de cuidados de saúde. Para tal usou-se o método sugerido no enunciado para a caracterização de conhecimento, onde temos **utente**, **prestador** e **cuidado**, os quais foram usados para a demonstração das várias funcionalidades. Em baixo, é possivel visualizar a forma como está definida a nossa **base de conhecimento**.

1.1 Base de Conhecimento

Utentes:

```
utente: #IdUt, Nome, Idade, Morada -> {V,F}

utente(1,'Tiago',23,morada('Rua 1','Esposende')).
utente(2,'Cesario',21,morada('Rua 2','Valença')).
utente(3,'Luis',22,morada('Rua 3','Viana')).
utente(4,'Rui',21,morada('Rua 4','Braga')).
```

Prestadores:

```
prestador: #IdPrest, Nome, Especialidade,Instituição -> {V,F}

prestador(1,'Nelon Puga','Desporto',instituicao('Clinica do Dragao', morada('Rua das Antas','Porto'
prestador(2,'Joaquim','Cardiologia',instituicao('Clinica do Dragao', morada('Rua das Antas','Porto'
```

Cuidados Médicos:

```
cuidado: Data, #IdUt, #IdPrest, Descrição, Custo -> {V,F}

cuidado(data(10,02,2018),1,9,'Consulta Rotina',30).
cuidado(data(10,02,2018),2,9,'Consulta Rotina',30).
```

Extra - Enuncidado

Além do enunciado proposto, decidimos considerear estes dois predicados extra, de forma a oferecer mais dinamismo e, principalmente, organização na nossa base de conhecimento de forma a que a seleção de informação fosse feita de uma forma mais rica e sistemática.

Instituições:

```
instituição: Data, #IdUt, #IdPrest, Descrição, Custo -> {V,F}

instituicao('Clinica do Dragao', morada('Rua das Antas','Porto')).
instituicao('Hospital Sao Joao', morada('Rua da Circunvalacao','Porto')).
```

Moradas:

```
morada: Rua,Localidade -> {V,F}
```

Datas:

```
data: Dia, Mes, Ano -> {V,F}

data(D, M, A) :- pertence(M, [1,3,5,7,8,10,12]), D >= 1, D =< 31.
data(D, M, A) :- pertence(M, [4,6,9,11]), D >= 1, D =< 30.
data(D, 2, A) :- A mod 4 =\= 0, D >= 1, D =< 28.
data(D, 2, A) :- A mod 4 =:= 0, D >= 1, D =< 29.
```

Capítulo 2

Desenvolvimento

Neste capitulo será explicado os predicados utilizados na resolução dos requisitos propostos. Apresentar-se-à também o código usado na representação dos predicados fundamentais, e ainda uma breve explicação acerca dos predicados auxiliares utilizados.

2.1 Registar utentes, prestadores e cuidados médicos.

Registo na base de conhecimento os predicados utentes, prestadores e cuidados médicos.

2.1.1 Predicado Principal

```
resgistar T -> {V,F}
```

O registo de utentes,prestadores e cuidados prestados é efetuado a partir do predicado registar.

```
registar( Termo ) :-
    solucoes( Invariante,+Termo::Invariante,Lista),
                insercao(Termo),
                    teste(Lista).
```

Este predicado tem em conta os invariantes criados, de modo a evitar a inserção de conhecimento repetido, e de modo a manter a consistência da nossa base de conhecimento. Os invariantes utilizados para o efeito são os seguintes:

2.1.2 Invariantes

```
+utente(ID,N,I,M) ::
        (solucoes( (ID),
            (utente(ID,Ns,Is,Ms)),S),
                comprimento(S,N), N==1).
```

Este invariante está definido de forma a não permitir informação repetida de utentes na nossa base de conhecimento. Pode estar repetido através do ID, ou seja, cada utente tem um ID único.

```
+prestador(ID,N,E,I) ::
        (solucoes( (ID),
            (prestador(ID,Ns,Es,Is)), S),
                comprimento(S,N),N==1).
```

Este invariante está definido de forma a não permitir informação repetida de prestadores na nossa base de conhecimento. Pode estar repetido através do ID, ou seja, cada prestador tem um ID único.

```
+cuidado(Dt,IDU,IDP,D,C) ::
        (solucoes( (Dt,IDU,IDP),
            (cuidado(Dt,IDU,IDP,Ds,Cs)),S),
                comprimento(S,N),N==1).
```

Este invariante está definido de forma a não permitir informação repetida de cuidados de saude na nossa base de conhecimento. Pode estar repetido através da data, do IDU e do IDP em simultâneo, ou seja, cada cuidado de saude tem um trio data, IDU e IDP único.

```
+cuidado(Dt,IDU,IDP,D,C) ::
        (solucoes( (IDU),
            (utente(IDU,Ns,Is,Ms)),S),
                comprimento(S,N),N==0).

+cuidado(Dt,IDU,IDP,D,C) ::
        (solucoes( (IDP),
            (prestador(IDP,Ns,Es,Is)),S),
                comprimento(S,N),N==0).
```

Estes dois invariantes protegem a nossa base de conhecimento, de maneira a não ser inserido conhecimento por falta de conhecimento, ou seja, nao é possivel adicionar um cuidado de saude com um determinado IDU ou IDP caso estes nao constem no predicado utente ou prestador, respetivamente.

```
+instituicao(N,M) ::
    (solucoes( (N),
        instituicao(N,M) ,S),
            N==1).
```

Este invariante está definido de forma a não permitir informação repetida de instituições na nossa base de conhecimento. Pode estar repetido através do nome, ou seja, cada instituição tem um nome único.

2.1.3 Predicados Auxiliares

Nesta secção apresentamos os predicados auxiliares usados no predicado registar.

```
solucoes T,Q,S -> {V,F}
```

Predicado que guarda numa lista as soluções da questão efetuada,com o formato especificado.

```
insercao T -> {V,F}

insercao(T) :- assert(T).
insercao(T) :- retract(T),!,fail.
```

Este predicado trata de inserir na base de conhecimento em caso de sucesso, senão remove o conhecimento inserido em caso de insucesso.

```
teste L -> {V,F}

teste([]).
teste([I|L]) :- I, teste(L).
```

Testa se existe algum invariante que não seja satisfeito, se falhar em algum dos invariantes , a inserção falha e assim não ocorre evolução da nossa base de conhecimento. Se todos os invariantes forem satisfeitos, a nossa base de conhecimento é evoluída com sucesso.

```
comprimento Lista, R -> {V,F}

comprimento([],0).
comprimento([X|L],R) :- comprimento(L,N) , R is 1+N.
```

Este predicado calcula o numero de elementos de uma lista.

2.2 Remover utentes, prestadores e cuidados médicos.

Remover na base de conhecimento os predicados utentes, prestadores e cuidados médicos.

2.2.1 Predicado Principal

```
remover T -> {V,F}
```

A remoção de utentes,prestadores e cuidados prestados é efetuado a partir do predicado remover.

```
remover( Termo ) :-
    solucoes( Invariante,+Termo::Invariante,Lista),
                remocao(Termo),
                    teste(Lista).
```

Este predicado tem em conta os invariantes criados, de modo a evitar a remoção de conhecimento critico, e de modo a manter a consistência da nossa base de conhecimento. Os invariantes utilizados para o efeito são os seguintes:

2.2.2 Invariantes

```
-utente(ID,N,I,M) ::
    (solucoes( (ID),
        (cuidado(Dts,ID,IDPs,Ds,Cs)) , S),
            comprimento(S,N),N>0).
```

Este invariante está definido de forma a não permitir remoção de utentes, quando estes pertencem ao predicado do conhecimento de cuidados medicos.

```
-prestador(ID,N,E,I) ::
    (solucoes( (ID),
        (cuidado(Dts,IDUs,ID,Ds,Cs)), S),
            comprimento(S,N),N==0).
```

Este invariante está definido de forma a não permitir remoção de prestadores, quando estes pertencem ao predicado do conhecimento de cuidados medicos.

```
-cuidado(Dt,IDU,IDP,D,C) ::
    (solucoes( (Dt,IDU,IDP,D,C),
        (cuidado(Dt,IDU,IDP,D,C)) ,S),
            comprimento(S,N), N==0).
```

Este invariante está definido de forma a não permitir remoção de cuidados, quando estes nao existem na base de conhecimento.

```
-instituicao(N,M) ::
    (solucoes( (N,M),
        prestador(_,_,_,(N,M)) , S),
            comprimento(S,N),N==0).
```

Este invariante está definido de forma a não permitir remoção de instituições, quando estes pertecem ao predicado do conhecimento de prestadores.

2.2.3 Predicados Auxiliares

Nesta secção apresentamos os novos predicados auxiliares usados no predicado remover.

```
remocao T -> {V,F}

remocao(T) :- retract(T).
remocao(T) :- assert(T),!,fail.
```

2.3 Identificar utentes por critérios de seleção

Na terceiro requisito do exercício é pedido que seja possível identificar os utentes pelos seus critérios de seleção, ou seja, pelo seu **ID**, **nome**, a sua **idade** ou **morada**. Através do predicado *soluções* a resposta bastante simples de concretizar.

```
ponto_tres : Critério,R -> {V,F}
```

A conclusão deste predicado é instruida com critério pelo qual vai ser feita a seleção dos utentes, e uma lista aonde se irá colocar o resultado dessa mesma seleção.

```
ponto_tres(ids,R) :-
    solucoes((ID),utente(ID,Ns,Is,Ms),R).

ponto_tres(nomes,R) :-
    solucoes((N),utente(IDs,N,Is,Ms),R).

ponto_tres(idades,R) :-
    solucoes((I),utente(IDs,Ns,I,Ms),R).

ponto_tres(moradas,R) :-
    solucoes((M),utente(IDs,Ns,Is,M),R).
```

Irá colocar todos os **ID's**, **nomes**, **idades** ou **moradas** presentes nos *utentes* em R.

2.4 Identificar as instituições prestadoras de cuidados de saúde

Identificar as instituições prestadoras de serviços de saude que ja tenham ou nao prestado cuidados de saude, permitindo fornecer ao utilizador uma noção das instituições a que pode recorrer.

```
ponto_quatro : Critério,R -> {V,F}
```

Neste predicado os critérios instruidos na conclusão especificam se queremos todas as instituições presentes na base de conhecimento, ou apenas aquelas que ja forneceram cuidados de saude.

```
ponto_quatro(todas,R) :-
    solucoes( (N) ,
        instituicao(N,M), R).

ponto_quatro(cuidados,R) :-
    solucoes( (N),
        (cuidado(_,_,IDP,_,_),prestador(IDP,_,_,instituicao(N,_)))S),
            unicos(S,R).
```

No primeiro caso, é uma pesquisa simples com o auxilio do predicado *soluções* colocando em **R**, a lista com o **nome** de todas as instituições presentes na nossa base de conhecimento. Achamos suficiente caracterizar as instituições apenas com o nome visto que ja fornece ao utilizador um nivel pormenorizado de informação. No segundo caso, a pesquisa do predicado *soluções* é mais complexa. Neste caso é feita uma compatibilidade interna de todos os cuidados de saude e todos os prestadores que contenham o mesmo **IDP**. A todos os casos que obedeçam a este critério, o nome da instituição dos prestadores é guardado numa lista auxiliar, sendo posteriormente essa lista filtrada através do predicado *unicos* (predicado que será definido mais à frente), colocando assim o resultado final sem repetições no argumento **R**.

2.5 Identificar cuidados de saúde prestados por instituição/cidade/datas

Este ponto do enunciado do enunciado têm como função filtrar os cuidados de saúde dado uma instituição, uma cidade ou uma data.

```
ponto_cinco : Critério,X,R -> {V,F}
```

No predicado acima indicado têm como critérios possíveis uma *instituicao*, uma *cidade* ou uma *data*, sendo X, respetivamente, o **nome**, a **localidade**, ou a **data**.

```
ponto_cinco(inst,X,R) :-
    solucoes( (Dt,IDU,IDP,D,C) ,
        (prestador(IDP,_,_,instituicao(X,_)) , cuidado(Dt,IDU,IDP,D,C) ), R).



ponto_cinco(cidade,X,R) :-
    solucoes( (Dt,IDU,IDP,D,C) ,
        (prestador(IDP,_,_,instituicao(_,morada(_,X))) , cuidado(Dt,IDU,IDP,D,C) ), R).

% *** outra_versao -> que nao faz tanto sentido ***

% ponto_cinco(cidade,X,R) :-
    solucoes( (Dt,IDU,IDP,D,C) ,
        (utente(IDU,_,_,morada(_,X)) , cuidado(Dt,IDU,IDP,D,C) ), R).



ponto_cinco(data,X,R) :-
    solucoes( (X,IDU,IDP,D,C) ,
        cuidado(X,IDU,IDP,D,C) , R).
```

Como é possível observar, o seu funcionamento é semelhante ao predicado anterior. No primeiro argumento do predicado soluções colocamos o que prentiamos que fosse o nosso resultado e no segundo argumento fizemos as compatibilidades necessárias para obter a lista com os resultados. Neste caso, decidimos que os argumentos minimos para oferecer ao utilizador informação suficiente sobre os cuidados de saude seriam todos os argumentos que definem o predicado *cuidado*. A lista com o resultado obtido é colocada no argumento **R** da conclusão.

2.6 Identificar os utentes de um prestador/especialidade/instituição

Este ponto tem como função executar a filtragem de todos os utentes que tenham efetuado cuidados de saude com um prestador, numa especialidade ou numa instituição.

```
ponto_seis : Critério,X,R -> {V,F}
```

A conclusão deste predicado, tem como critérios possiveis, o *prest*, a *esp* e a *inst*, que correspondem, respetivamente, ao **ID** do prestador, ao **nome** da especialidade e ao **nome** da instituição. Decidimos instruir a conclusão com o *ID* do prestador e o nome da instituição, pois são os identificadores unicos dos prestadores e das instituições na nossa base de conhecimento, ou seja, nao existe dois prestadores com o mesmo *id*, nem duas instituições com o mesmo nome.

```
ponto_seis(prest,X,R) :-
    solucoes( (IDU,N),
        ( cuidado(_,IDU,X,_,_),utente(IDU,N,_,_)), S),
            unicos(S,R).



ponto_seis(esp,X,R) :-
    solucoes( (IDU,N),
        ( prestador(IDP,_,X,_), cuidado(_,IDU,IDP,_,_), utente(IDU,N,_,_)), S),
            unicos(S,R).



ponto_seis(inst,X,R) :-
    solucoes( (IDU,N),
        ( cuidado(_,IDU,_,_,instituicao(X,_)), utente(IDU,N,_,_)), S),
            unicos(S,R).
```

Na resposta a este ponto, que é colocada no argumento **R**, decidimos que cada utente ia ser caracterizado pelo seu **nome** e **ID**. Apesar do ID ser o identificador unico dos utentes na nossa base de conhecimento, achamos que fornecer apenas essa informação ao utilizador podia ser curto, portanto acrescentamos o nome de cada utente ao seu identificador.

2.7 Identificar cuidados de saúde realizados por utente/instituição/prestador

Este ponto, tem como objetivo listar todos os cuidados de saúde que tenham sido realizados por um utente, uma instituição ou por um prestador.

```
ponto_sete : Criterio,X,R -> {V,F}
```

A conclusão deste predicado, tem como critérios possiveis o *utente*, a *inst* ou o *prest*, que correspondem, respetivamente, ao **ID** do utente, **nome** da instituição ou ao **ID** do prestador. Como foi dito anteriormente, tanto o id do utente, o id do prestador, e o nome da instituição, são respetivamente, os identificadores unicos dos seus predicados, não existindo estes identificadores repetidos na nossa base de conhecimento, é portanto, os argumentos necessários para instruir a conclusão deste predicado.

```
ponto_sete(utente,X,R) :-
    solucoes( (Dt,X,IDP,D,C),
        cuidado(Dt,X,IDP,D,C), R).
```

```
ponto_sete(inst,X,R) :-
    solucoes( (Dt,IDU,IDP,D,C),
        ( prestador(IDP,_,_,instituicao(X,_)), cuidado(Dt,IDU,IDP,D,C)),R).



ponto_sete(prest,X,R) :-
    solucoes( (Dt,IDU,X,D,C),
        cuidado(Dt,IDU,X,D,C), R).
```

Uma vez mais, o funcionamento do predicado soluções é igual aos descritos em cima, e o resultado colocado em **R** é a lista de todos os cuidados pretendidos, sendo que cada cuidado é definido por todos os seus argumentos.

2.8 Determinar todas as instituições/prestadores a que um utente já recorreu

Este ponto tem como objetivo filtrar todas as instituições ou prestadores que ja tenham prestado cuidados de saude a um utente.

```
custoTotal : Critério,Utente,R -> {V,F}
```

A conclusão deste predicado tem como critérios possiveis o *inst*, caso o utilizador pretenda listar as instituições ou o *prest*, caso pretenda listar os prestadores. O argumento **U** é o identificador único de um utente da nossa base de conhecimento, ou seja o seu **IDU**.

```
ponto_oito(inst,U,R) :-
    solucoes( (N) ,
        ( cuidado(_,U,IDP,_,_), prestador(IDP,_,_,instituicao(N,_))) ,S) ,
            unicos(S,R).


ponto_oito(prest,U,R) :-
    solucoes( (ID,N) ,
        ( cuidado(_,U,IDP,_,_), prestador(IDP,N,_,_) ,S)) ,
             unicos(S,R).
```

No argumento **R**, é colocada a lista pretendida sendo que cada instituição é definida pelo seu **nome**, e os prestadores são definidos pelo seu **ID** e pelo seu **nome**.

2.9 Calcular o custo total dos cuidados de saúde por utente/especialidade/prestador/datas

Neste ponto, é pretendido que seja feito a soma total dos custos de todos os cuidados de saude efetuados por um utente, uma especialidade, um prestador, ou numa determinada data.

```
ponto_nove : Critério,Valor,R -> {V,F}
```

A conslusão deste predicado tem como critérios o *utente*, a *esp*, o *prest* e a *data*. O argumento **X** é instruido com os respetivos identificadores unicos de cada predicado.

```
ponto_nove(utente,X,R) :-
    solucoes( (C),
        cuidado(Dt,X,IDP,D,C), S),
            somaTotal(S,R).



ponto_nove(esp,X,R) :-
    solucoes( (C),
        (prestador(IDP,_,X,_), cuidado(_,_,IDP,_,C)) , S),
            somaTotal(S,R).



ponto_nove(prest,X,R) :-
    solucoes( (C),
        cuidado(Dt,IDU,X,D,C), S),
            somaTotal(S,R).



ponto_nove(data,X,R) :-
    solucoes( (C),
        cuidado(X,IDU,IDP,D,C), S),
            somaTotal(S,R).
```

Após ser executada a filtragem através do predicado soluções a lista **S**, que no momento é uma lista de valores de custos, é fornecida ao prediado **somaTotal**, que calcula a soma de todos os valores e coloca esse resultado no argumento **R**.

2.10 Predicados Gerais

2.10.1 Predicado: nao

Este predicado serve para negar o valor de verdade de uma questão.

```
nao: Questão -> {V,F}

nao(Q) :- Q, !, fail.
nao(Q).
```

2.10.2 Predicado: pertence

Este predicado foi definido para saber se um dado valor faz parte de uma lista.

```
pertence: Valor, Lista -> {V,F}

pertence(H,[H|T]).
pertence(X,[H|T]) :-
    X \= H,
    pertence(X,T).
```

2.10.3 Predicado: unicos

Este predicado foi definido com o obetivo de filtrar valores repetidos numa lista. É utilizado em quase todos os pontos do enunciado.

```
unicos: Lista, R -> {V,F}

unicos([],[]).
unicos([H|T], R) :-
    pertence(H,T),
    unicos(T,R).
unicos([H|T], [H|R]) :-
    nao(pertence(H,T)),
    unicos(T,R).
```

2.10.4 Predicado: somaTotal

Este predicado foi definido com o objetivo de calcular a soma de todos os valores de uma lista. É utilizado no predicado do ponto nove.

```
somaTotal: Lista,Resultado -> {V,F}

somaTotal( [],0 ).
somaTotal( [X],X ).
somaTotal( [X|L],R ) :- somaTotal( L,R1 ), R is X+R1.
```

Capítulo 3

Conclusão

Este primeiro exercício ajudou bastante a ambientarmo-nos à linguagem de programacão lógica PROLOG, permitindo-nos caracterizar um universo de forma simples e estruturada.

Os principais obstáculos que encontramos no desenvonvimento do projeto, foi na forma de como íriamos estruturar o nosso pensamento usando programacão em lógica, sendo que após ultrapassarmos este problema, a resolução do exercício foi fácil.

Em suma, cumprimos todos os requisitos pedidos no enunciado, conseguido assim um trabalho sólido e competente.