Universidade do Minho

Trabalho Prático I

Mestrado Integrado em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio.

2º Semestre — 2018/2019

a77523 - Alexandre Martins
a74814 - João Bernardo Freitas
a74806 - João Amorim
ae4560 - Tiago Gomes

.

17 de Março 2019
Braga

Resumo

O presente trabalho tem como propósito desenvolver um sistema de representação de conhecimento e raciocínio com capacidade de caracterizar um universo de discurso na área de prestação de cuidados de saúde, recorrendo à programação lógica em *Prolog*.

De forma a abordar os desafios propostos, desenvolvemos uma base de conhecimento com os componentes que determinamos apropriados para tal. Após a introdução do projecto, é feita uma descrição detalhada da solução apresentada para ir de encontro ao objectivo proposto.

Por último, é feita uma reflexão crítica sobre os resultados que foram obtidos.

Contents

1 Introdução

A programação em lógica assume-se como um tipo bastante especifico de programação, distinta das linguagens de programação mais comuns. Além de ser simples de utilizar, é normalmente descrita como uma linguagem declarativa, apesar de ter algumas características procedimentais. Desta forma, em vez de serem especificados os caminhos para atingir um certo objectivo, é especificada a situação, sendo o objectivo esperado o de que o interpretador encontre a solução.

Para este primeiro desafio prático, é solicitado o desenvolvimento de um sistema de representação de conhecimento e raciocínio capaz de caracterizar o discurso na área da prestação de cuidados de saúde. Fazendo uso das características da linguagem PROLOG, são implementadas funcionalidades para o registo e remoção de conhecimento, assim como funcionalidades que dão resposta não só às alíneas propostas como a algumas outras adicionais.

2 Preliminares

Para o desenvolvimento do problema proposto, a aprendizagem efectuada pelo grupo no decorrer das aulas foi crucial para o desenvolvimento de todo o projecto. Para além disso, uma análise cuidada e prévia do enunciado proposto, assim como uma devida distribuição das tarefas a realizar por cada elemento do grupo, permitiram que toda a resolução do projecto decorresse sem qualquer tipo de problema.

3 Descrição do Trabalho e Análise dos Resultados

Sendo o objectivo deste trabalho o de criar um sistema de representação de conhecimento e raciocínio que caracterize a prestação de cuidados de saúde, é em seguida apresentada a base do conhecimento que consideramos necessária, seguida de uma descrição e análise do trabalho elaborado.

3.1 Base do Conhecimento

Tal como já referido, a base do conhecimento que caracteriza este sistema de prestação de cuidados de saúde é o seguinte:

3.1.1 Utentes

Um utente é caracterizado pelo seu identificador único, nome, idade e cidade.

```
% Extenso do predicado utente: #IdUt, Nome, Idade, Cidade -> {V,F}
```

```
utente( 0, 'Elias',   28, 'Barcelos').
utente( 1, 'Sebastiao', 18, 'Porto').
utente( 2, 'Martim',  32, 'Braga').
utente( 3, 'Lucia',   55, 'Guimaraes').
utente( 4, 'Alexandre', 76, 'Lisboa').
utente( 5, 'Juliana', 22, 'Porto').
utente( 6, 'Joao',    8,  'Famalicao').
utente( 7, 'Antonio', 16, 'Viseu').
utente( 8, 'Isabela', 36, 'Vila do Conde').
utente( 9, 'Artur',   51, 'Setubal').
utente(10, 'Alice',   86, 'Povoa de Varzim').
```

3.1.2 Serviços

Um serviço é caracterizado pelo seu identificador único, descrição do serviço, identificador da instituição onde o serviço é prestado e cidade onde o mesmo é prestado. Mais à frente será possível observar que é criado todo o conhecimento referente às instituições, o que nos permite obter informações com mais detalhe sobre as mesmas, sendo por isso apenas utilizado um identificador da instituição.

```
% Extensao do predicado servico: #IdServ, Descrio,  Instituio , Cidade
    -> {V, F}

servico( 0, 'Dermatologia', 2, 'Porto').
servico( 1, 'Psiquiatria', 1, 'Porto').
servico( 2, 'Neurologia', 0, 'Braga').
servico( 3, 'Ortopedia', 3, 'Lisboa').
servico( 4, 'Podologia', 4, 'Barcelos').
servico( 5, 'Cardiologia', 5, 'Braga').
servico( 6, 'Oftalmologia', 6, 'Faro').
servico( 7, 'Pediatria', 7, 'Famalicao').
servico( 8, 'Psicologia', 7, 'Evora').
```

3.1.3 Consultas

Uma consulta é caracterizada pela data, sendo esta definida como data(DD, MM, AA), identificador do utente relativo à consulta, identificador do serviço a que corresponde a consulta, identificador do médico que a realiza e o seu custo.

```
% Extensao do predicado consulta: Data, #IdUt, #IdServ, IdMed, Custo ->
    {V, F}

consulta(data(20, 2, 2019), 0, 4, 0, 20).
consulta(data(20, 2, 2019), 1, 0, 7, 70).
```

```
consulta(data(21, 3, 2019), 1, 7, 2, 35).
consulta(data(21, 3, 2019), 2, 7, 1, 50).
consulta(data(07, 4, 2019), 3, 5, 4, 100).
consulta(data(07, 4, 2019), 4, 3, 5, 45).
consulta(data(10, 4, 2019), 4, 6, 3, 25).
consulta(data(10, 4, 2019), 5, 1, 6, 40).
consulta(data(10, 4, 2019), 6, 2, 1, 50).
consulta(data(13, 4, 2019), 6, 1, 0, 40).
consulta(data(13, 4, 2019), 7, 6, 4, 25).
consulta(data(19, 4, 2019), 8, 7, 2, 35).
consulta(data(01, 5, 2019), 8, 0, 5, 70).
consulta(data(01, 5, 2019), 9, 3, 7, 45).
consulta(data(06, 6, 2019), 10, 2, 5, 50).
consulta(data(12, 5, 2019), 10, 0, 4, 70).
consulta(data(12, 5, 2019), 10, 4, 1, 20).
consulta(data(13, 5, 2019), 2, 8, 6, 50).
```

3.2 Predicados Auxiliares

De seguida são apresentados predicados auxiliares, indispensáveis para a correcta construção dos predicados que são necessários para dar resposta às questões do enunciado.

3.2.1 Extensão do predicado soluções.

Predicado essencial para o correcto funcionamento de todo o sistema. Utiliza um predicado nativo da linguagem PROLOG, findall, que procura todas as ocorrências de F em Q, devolvendo o resultado numa lista S.

```
% Extenso do predicado solues : F, Q, S -> {V, F}

solucoes(F, Q, S) :- findall(F, Q, S).
```

3.2.2 Extensão do predicado comprimento.

Predicado utilizado em conjunto com o predicado soluções, que dada uma lista S, determina o seu tamanho N. Tal como o predicado soluções usa um predicado nativo da linguagem PROLOG, length.

```
% Extensao do predicado comprimento: S, N -> {V, F}

comprimento(S, N) :- length(S, N).
```

3.2.3 Extensão do predicado sum

```
% Extensao do predicado sum: Lista, R -> {V, F}
% Somatorio de uma lista

sum([], 0).
sum([H|T], R) :- sum(T, L), R is H + L.
```

3.2.4 Extensão do predicado semReps.

```
% Extenso do predicado apagaReps: L, R -> {V, F}
% Apaga diversos elementos repetidos numa lista.

semReps([], []).
semReps([H|T], [H|L]) :- apaga(H, T, X),
                         semReps(X, L).
```

3.2.5 Extensão do predicado apagaT.

```
% Extenso do predicado apagaT: X, L, R -> {V, F}
% Apaga todas as ocorrncias repetidas de um elemento numa lista.

apaga(X, [], []).
apaga(X,[X|L1],L2) :- apaga(X,L1,L2).
apaga(X,[Y|L1],[Y|L2]) :- apaga(X,L1,L2).
```

3.2.6 Extensão do predicado concatenar.

```
% Extenso do predicado concatenar : L1,L2,R -> {V,F}

concatenar([],L,L).
concatenar([X|L1],L2,[X|L3]) :- concatenar(L1,L2,L3).
```

3.2.7 Extensão do predicado concListList.

```
% Extenso do predicado concListList: LLs, L -> {V, F}
% Utilizando o predicado auxiliar concatenar, concatena listas dentro de
    uma lista.

concListList([], []).
concListList([H|T], L) :- concListList(T, L1),
                          concatenar(H, L1, L).
```

3.2.8 Extensão do predicado insere.

```
% Extensao do predicado insere: Termo -> {V, F}

insere(Termo) :- assert(Termo).
insere(Termo) :- retract(Termo), !, fail.
```

3.2.9 Extensão do predicado remove.

```
% Extensao do predicado remove: Termo -> {V, F}

remove(Termo) :- retract(Termo).
remove(Termo) :- assert(Termo), !, fail.
```

3.2.10 Extensão do predicado teste.

```
% Extensao do predicado teste: Lista -> {V, F}

teste([]).
teste([H | T]) :- H, teste(T).
```

3.2.11 Extensão do predicado evolução

A evolução do sistema é caracterizada pelo aumento do conhecimento, sendo este o objetivo deste predicado. O predicado insere tem como objectivo a inserção de conhecimento no sistema enquanto que o predicado teste se encarrega de realizar o teste de acordo com os invariantes relativos ao Termo fornecido. É possível verificar que este processo insere primeiro o conhecimento e só depois é que realiza o teste à consistência do sistema. Foi também definido que um invariante relativo à inserção de um termo é precedido pelo símbolo '+'.

```
% Extensao do predicado evolucao: Termo -> {V, F}
evolucao(Termo) :- solucoes(Inv, +Termo::Inv, S),
                   insere(Termo),
                   teste(S).
```

3.2.12 Extensão do predicado involução

A involução do sistema é caracterizada pela remoção do conhecimento, sendo este o objetivo deste predicado. O processo de involução é semelhante ao de evolução, com a diferença de que neste caso é efetuado um teste inicial ao próprio termo para não ser efetuada uma remoção de conhecimento que não

exista e a troca do predicado insere pelo remove. Foi também definido que um invariante relativo à remoção de um termo é precedido pelo símbolo '-'.

```
% Extensao do predicado involucao: Termo -> {V, F}
involucao(Termo) :- Termo,
                    solucoes(Inv, -Termo::Inv, S),
                    remove(Termo),
                    teste(S).
```

3.3 Inserção e Remoção de Conhecimento - Invariantes

A inserção e remoção de conhecimento é definida com base em invariantes. Estes invariantes especificam um conjunto de restrições que devem ser sempre verdadeiras após qualquer inserção ou remoção de conhecimento. Podemos ver a inserção de conhecimento como uma evolução do sistema, enquanto que uma remoção de conhecimento pode ser vista como uma involução do sistema, tendo esta evolução e involução sido já explicitadas na secção dos predicados auxiliares. É de salientar que ambas as operações, de inserção e remoção, são sempre efectuadas, sendo a consistência do sistema verificada após a execução de cada uma, com recurso aos invariantes. Caso alguma operação não cumpra com as restrições dos invariantes, é causada uma anomalia no sistema e este volta ao seu estado anterior.

3.3.1 Invariante Estrutural - Inserir Utente.

De forma a não permitir a inserção de utentes repetidos (com o mesmo *id*), bem como a definição de restrições para o *id* e idade, desenvolvemos o seguinte invariante estrutural:

```
% Invariante Estrutural: No permitir a insercao de utentes repetidos
    (com o mesmo ID)
% Id e Idade inteiros
% Idade entre 0 e 120

+utente(Id, _, Idade, _) :: (
    integer(Id),
    integer(Idade),
    Idade >= 0,
    Idade =< 120,
    solucoes(Id, utente(Id, _, _, _), S),
    comprimento(S, N),
    N == 1
).
```

3.3.2 Invariante Referencial - Remover Utente

Para a remoção de um utente, é necessário garantir que não existem consultas a si associadas, tendo para isso sido desenvolvido o seguinte invariante:

```
% Invariante Referencial: Nao permitir a remocao de um utente enquanto
    existirem consultas
% associadas ao mesmo
-utente(IdUt, _, _, _) :: (
   solucoes(IdUt, consulta(_, IdUt, _, _, _), S1),
   comprimento(S1, N1),
   N1 == 0
).
```

3.3.3 Invariante Estrutural - Inserir Serviço

Para não permitir a inserção de serviços que já existam (com o mesmo ID) ou identificadores de serviço não inteiros, não permitir a inserção de tipos de serviço que já existam, ou seja, cuja descrição, em conjunto com a instituição mais a cidade já existam, serviços cuja instituição não exista e ainda serviços cuja cidade inserida não corresponda à instituição escolhida, foi construído o seguinte invariante:

```
% Invariante Estrutural: Nao permitir a insercao de servicos que ja
    existam (com o mesmo ID),
% tipos de servico que j existam, ou seja, cuja descricao + instituicao
    + cidade j existam,
% servicos cuja instituicao nao exista e ainda servicos cuja cidade
    inserido nao corresponde
% a instituicao.
% Id deve ser inteiro

+servico(Id, Desc, IdInst, Cid) :: (
    integer(Id), integer(IdInst),
    solucoes(Id, servico(Id, _, _, _), S1),
    solucoes((Desc,IdInst,Cid), servico(_, Desc, IdInst, Cid), S2),
    solucoes((IdInst, Cid), instituicao(IdInst, _, Cid), S3),
    comprimento(S1, N1),
    comprimento(S2, N2),
    comprimento(S3, N3),
    N1 == 1,
    N2 == 1,
    N3 == 1
).
```

3.3.4 Invariante Referencial - Remover Serviço

De forma a garantir que o serviço não é removido enquanto existirem consultas associadas ao mesmo, foi desenvolvido o seguinte invariante referencial:

```
% Invariante Referencial: Nao permitir a remocao de servico enquando
    existirem consultas
% associadas ao mesmo

-servico(IdServ, _, _, _) :: (
   solucoes(IdServ, consulta(_, _, IdServ, _, _), S1),
   comprimento(S1, N1),
   N1 == 0
).
```

3.3.5 Invariante Estrutural - Inserir Consulta

O invariante que diz respeito à inserção de uma consulta é sem dúvida o mais complexo. Este invariante garante em primeiro lugar que todos os identificadores são inteiros e que o custo referente a uma consulta é um número maior ou igual a 0. De seguida, verifica se os identificadores de utente, serviço e médico existem, sendo que se não existirem, a consulta não será adicionada. Por último, é feita a verificação da existência de uma consulta igual aquela que se pretende adicionar, sendo que se já existir uma em que todos os parâmetros são iguais, a consulta não é adicionada. Esta última restrição poderá acrescentar alguma inconsistência ao sistema, mas consideramos que no mundo real não existem 2 consultas no mesmo dia, para o mesmo utente, serviço e médico e com um custo igual.

```
% Invariante Estrutural: No permitir a existencia de consultas repetidas
% Nao permitir a insercao de uma consulta cujo utente, servico ou medico
    nao existam
% O valor do primeiro argumento deve ser uma data e o custo deve ser do
    tipo number
% e ser maior ou igual a zero

+consulta(Data, IdUt, IdServ, IdMed, Custo) :: (
   integer(IdUt), integer(IdServ), integer(IdMed),
   number(Custo), Custo >= 0,
   solucoes(IdUt, utente(IdUt, _, _, _), S1),
   solucoes(IdServ, servico(IdServ, _, _, _), S2),
   solucoes(IdMed, medico(IdMed, _, _, _), S3),
   solucoes((Data, IdUt, IdServ, IdMed, Custo),
            consulta(Data, IdUt, IdServ, IdMed, Custo), S4),
   comprimento(S1, N1),
   comprimento(S2, N2),
   comprimento(S3, N3),
   comprimento(S4, N4),
```

```
   N1 == 1, N2 == 1, N3 == 1, N4 ==1
).
```

3.3.6 Invariante Referencial - Remover Consulta

Para remover uma consulta se esta existir, foi desenvolvido o seguinte invariante referencial:

```
% Invariante Referencial: Remover uma consulta se esta existir
-consulta(Data, IdUt, IdServ, IdMed, Custo) :: (
            solucoes((Data, IdUt, IdServ, IdMed, Custo),
            consulta(Data, IdServ, IdServ, IdMed, Custo), S1),
            comprimento(S1, N1),
            N1 = 0
).
```

3.4 Capacidades do sistema desenvolvido.

3.4.1 Registar utentes, serviços e consultas.

Esta alínea diz respeito à evolução do sistema, ou seja, à inserção de novo conhecimento, sendo para isto utilizado o conjunto de predicados e invariantes já descritos para o efeito da mesma. Nesse sentido, para a inserção de utentes, serviços e consultas no sistema, utilizamos o predicado *evolução*, que insere e verifica se não existem irregularidades quando confrontado com os invariantes que foram definidos. Caso sejam encontradas irregularidades, o estado anterior à inserção é restabelecido.

3.4.2 Remover utentes, serviços e consultas.

De forma a realizar o proposto, e seguindo um raciocínio semelhante ao da alínea anterior, usamos desta vez o predicado *involução*. Este permite, através do teste dos invariantes desenvolvidos, remover conhecimento do sistema.

3.4.3 Identificar as instituições prestadoras de serviços.

Para identificar as instituições prestadoras de serviços, foi desenvolvido o predicado que se observa de seguida. Visto que foi adicionado o conhecimento relativo às instituições e que estas não podem ser repetidas, basta apresentar todas as instituições presentes no sistema. Se as instituições fossem, tal como é apresentado no enunciado, apenas parte do conhecimento dos Serviços, seria necessário eliminar todas as ocorrências repetidas das mesmas.

```
% Extensao do Predicado identifica_instituicoes: Lista -> {V,F}
identifica_instituicoes(L) :-
   solucoes((Id, Nome, Cidade), instituicao(Id, Nome, Cidade), X),
```

3.4.4 Identificar utentes/serviços/consultas por critério de selecção

Para abordar esta alínea desenvolvemos um conjunto de extensões de predicados que nos permite, através do critério de selecção escolhido, obter um utente, serviço ou consulta. De notar, que tanto nesta alínea como em algumas das próximas, a identificação de, por exemplo, um utente, poderia ser feita em apenas um predicado que recebesse como parâmetros todos os critérios de selecção. Por uma questão de facilidade de leitura e de percepção, decidimos optar antes por fazer uma separação. Esta alínea é dividida em 3 partes, sendo estas:

- Identificação de Utente
- Identificação de Serviço
- Identificação de Consulta

Para a Identificação de Utente por identificador único, nome, idade ou cidade temos:

```
%------- Identificar Utentes -------
% Identificar utentes por critrios de  seleo ;
% ID
% Nome
% Idade
% Cidade

% Extensao do Predicado idUtenteID: ID, Lista -> {V,F}
idUtenteID(ID, L) :- solucoes((ID, Nome, Idade, Cidade), utente(ID,
    Nome, Idade, Cidade), L).

% Extensao do Predicado idUtenteNome: Nome, Lista -> {V,F}
idUtenteNome(Nome, L) :- solucoes((ID, Nome, Idade, Cidade), utente(ID,
    Nome, Idade, Cidade), L).

% Extensao do Predicado idUtenteIdade: Idade, Lista -> {V,F}
idUtenteIdade(Idade, L) :- solucoes((ID, Nome, Idade, Cidade),
    utente(ID, Nome, Idade, Cidade), L).

% Extensao do Predicado idUtenteCidade: Cidade, Lista -> {V,F}
idUtenteCidade(Cidade, L) :- solucoes((ID, Nome, Idade, Cidade),
    utente(ID, Nome, Idade, Cidade), L).
```

Para a Identificação de Serviço por identificador único, descrição, identificador de instituição ou cidade temos:

```
%------- Identificar Servios -------
% Identificar servicos por criterios de selecao;
% ID
% Descricao
```

```
% Instituicao
% Cidade

% Extenso do Predicado idUServicoID: ID, Lista -> {V,F}
idServicoID(ID, L) :- solucoes((ID, Descricao, Instituicao, Cidade),
                      servico(ID, Descricao, Instituicao, Cidade), L).

% Extenso do Predicado idServicoDesc: Descricao, Lista -> {V,F}
idServicoDesc(Descricao, L) :- solucoes((ID, Descricao, Instituicao,
    Cidade),
                           servico(ID, Descricao, Instituicao, Cidade),
                               L).

% Extenso do Predicado idServicoInst: Instituicao, Lista -> {V,F}
idServicoInst(Instituicao, L) :- solucoes((ID, Descricao, Instituicao,
    Cidade),
                             servico(ID, Descricao, Instituicao,
                                 Cidade), L).

% Extenso do Predicado idServicoCid: Cidade, Lista -> {V,F}
idServicoCid(Cidade, L) :- solucoes((ID, Descricao, Instituicao, Cidade),
                          servico(ID, Descricao, Instituicao, Cidade),
                              L).
```

Para a Identificação de Consultas por data, identificador de utente, identificador de serviço, identificador de médico ou custo temos:

```
%------- Identificar Consultas -------
% Identificar consultas por criterios de selecao;
% Data
% idUtente
% IdServico
% IdMedico
% Custo

% Extenso do Predicado idConsultaData: Data, Lista -> {V,F}
idConsultaData(Data, L) :- solucoes((Data, IdUt, IdServ, IdMed, Custo),
    consulta(Data, IdUt, IdServ, IdMed, Custo), L).

% Extenso do Predicado idConsultaIDUt: IdUt, Lista -> {V,F}
idConsultaIDUt(IdUt, L) :- solucoes((Data, IdUt, IdServ, IdMed, Custo),
    consulta(Data, IdUt, IdServ, IdMed, Custo), L).

% Extenso do Predicado idConsultaIDServ: IdServ, Lista -> {V,F}
idConsultaIDServ(IdServ, L) :- solucoes((Data, IdUt, IdServ, IdMed,
    Custo), consulta(Data, IdUt, IdServ, IdMed, Custo), L).

% Extenso do Predicado idConsultaIDMed: IdMed, Lista -> {V,F}
```

```
idConsultaIDMed(IdMed, L) :- solucoes((Data, IdUt, IdServ, IdMed,
    Custo), consulta(Data, IdUt, IdServ, IdMed, Custo), L).

% Extenso do Predicado idConsultaCusto: Custo, Lista -> {V,F}
idConsultaCusto(Custo, L) :- solucoes((Data, IdUt, IdServ, IdMed,
    Custo), consulta(Data, IdUt, IdServ, IdMed, Custo), L).
```

3.4.5 Identificar serviços prestados por instituição/cidade/data/custo.

Para a realização desta alínea, foi utilizado o predicado auxiliar semReps, que garante que não são mostrados serviços repetidos. Isto apenas acontece quando pretendemos identificar serviços por Data ou por Custo, visto que para os restantes, o invariante estrutural da inserção de serviços não permite a adição de serviços repetidos, ou seja, cuja descrição já esteja associada a uma instituição, instituição esta que está automaticamente associada a apenas uma cidade. Em suma, para uma instituição ou cidade, podemos ter vários serviços, mas caso se tente adicionar um serviço cuja descrição, em conjunto com a instituição e cidade, já existam, o invariante não permitirá a sua adição, mantendo assim o conhecimento sobre os serviços sem duplicados. De notar então que os predicados de identificação de serviços por instituição e por cidade correspondem aos predicados desenvolvidos na alínea anterior. De qualquer das maneiras, novos predicados são criados nesta alínea, com o objetivo de completar o pedido:

```
%------- Servicos por Instituicao -------
% Extenso do Predicado servicosPorInst: IdInstituicao, Lista -> {V,F}

servicosPorInst(IdInst, L) :- solucoes((IdServ, Descricao, Instituicao) ,
                              (servico(IdServ, Descricao, IdInst, Cidade),
                             instituicao(IdInst, Instituicao, _)), S).

%------- Servicos por Cidade -------
% Extenso do Predicado servicosPorCid: Cidade, Lista -> {V,F}

servicosPorCid(Cidade, L) :- solucoes((IdServ, Descricao, Cidade) ,
                              servico(IdServ, Descricao, _, Cidade), S).


%------- Servicos por Data -------
% Extenso do Predicado servicosPorData: Data, Lista -> {V,F}

servicosPorData(Data, L) :- solucoes((IdServ, Descricao, Data) ,
                            (servico(IdServ, Descricao, _, _),
                            consulta(Data, _, IdServ,_, _)), S),
                            semReps(S,L).


%------- Servicos por Custo -------
% Extenso do Predicado servicosPorCusto: Custo, Lista -> {V,F}
```

```
servicosPorCusto(Custo, L) :- solucoes((IdServ, Descricao, Custo) ,
                              (servico(IdServ, Descricao, _, _),
                             consulta(_, _, IdServ,_, Custo)), S),
                             semReps(S,L).
```

3.4.6 Identificar utentes de um serviço/instituição

Para tal desenvolvemos um predicado auxiliar, que utiliza um predicado construído na identificação de utentes dado o seu identificador e que devolve a lista dos utentes com as suas informações, dada uma lista com vários *IdUtente*:

```
%--- Predicado Auxiliar ---
listaInfoUtentes([], []).
listaInfoUtentes([IdUt|T], [L|Ls]) :- idUtenteID(IdUt, L),
                                      listaInfoUtentes(T, Ls).
```

Utilizando este predicado auxiliar desenvolvemos um conjunto de predicados para dar resposta ao pedido, apresentando no primeiro todos os utentes que tenham recorrido a um determinado serviço e no segundo todos os utentes que tenham recorrido a uma determinada instituição.

```
%------- Utentes de um Servico -------

% Extensao do predicado utentesPorInst: IdServico -> Lista {V, F}
% Identifica os utentes de um servico

utentesPorServ(IdServ, L) :- solucoes(IdU, (servico(IdServ, _, _, _),
                                        consulta(Data, IdU, IdServ,
                                            IdMed, Custo)), X),
                                        listaInfoUtentes(X,S),
                                        semReps(S,D),
                                        concListList(D, L).



%------- Utentes de uma Instituicao -------

% Extensao do predicado utentesPorInst: IdInstituicao -> Lista {V, F}
% Identifica os utentes de uma instituicao

utentesPorInst(IdInst, L) :- solucoes(IdU, (servico(IdServ, _, IdInst,
    _),
                                          instituicao(IdInst, _, _),
                                             consulta(Data, IdU, IdServ,
                                                 IdMed, Custo)), X),
                                             listaInfoUtentes(X,S),
                                            semReps(S,D),
```

```
                                                concListList(D, L).
```

3.4.7 Identificar serviços realizados por utente/instituição/cidade/médico.

Seguindo a mesma lógica das alíneas anteriores, desenvolvemos um conjunto de extensões de predicado que achamos adequado. O objetivo aqui seria apresentar todos os serviços que já tenham sido realizados e que digam respeito a um utente, instituição, cidade ou médico.

```
%------- Servicos por Instituicao -------
% Extenso  do Predicado servicosRealizUt: IdUtente, Lista -> {V,F}

servicosRealizUt(IdUt, L) :- solucoes((Data, IdServ, Descricao, Nome) ,
                             (servico(IdServ, Descricao, _, _),
                              utente(IdUt, Nome, _, _),
                              consulta(Data, IdUt, IdServ, _, _)), L).


% Extenso  do Predicado servicosRealizInst: IdInst, Lista -> {V,F}

servicosRealizInst(IdInst, L) :- solucoes((Data, IdServ, Descricao,
                                           Instituicao),
                                 (servico(IdServ, Descricao, IdInst, _),
                                  instituicao(IdInst, Instituicao, _),
                                  consulta(Data, IdUt, IdServ, _, _)), L).

% Extenso  do Predicado servicosRealizCid: Cidade, Lista -> {V,F}

servicosRealizCid(Cidade, L) :- solucoes((Data, IdServ, Descricao,
                                          Cidade),
                                 (servico(IdServ, Descricao, _, Cidade),
                                 consulta(Data, IdUt, IdServ, _, _)), L).

% Extenso  do Predicado servicosRealizMed: IdMed, Lista -> {V,F}

servicosRealizMed(IdMed, L) :- solucoes((Data, IdServ, Descricao, Nome) ,
                               (servico(IdServ, Descricao, _, Cidade),
                                medico(IdMed, Nome, _, _),
                                consulta(Data, IdUt, IdServ, IdMed, _)),
                                    L).
```

3.4.8 Custo total das consultas por utente/serviço/instituição/médico/data

No início, fazemos questão de expor a razão pela qual optamos construir um predicado para cada critério de seleção. Para esta alínea, resolvemos demonstrar a resolução do problema com um único predicado, dando assim outras opções de resolução deste tipo de problemas. O objetivo para este último predicado é

o de apresentar o custo total das consultas por identificador de utente, identificador de servico, identificador de instituicao, identificador de médico ou data, utilizando o predicado auxiliar sum, que obtêm a soma dos custos de todas as consultas referentes ao critério de seleção escolhido.

```
% Extenso  do Predicado custoToal: IdUtente, IdServico, Instituicao,
    IdMed, Data, Lista -> {V,F}
custoTotal(IdU, IdServ, IdInst, IdMed, Data, C) :-
   solucoes(Custo,
            (servico(IdServ, _, IdInst, _),
             consulta(Data, IdU, IdServ, IdMed, Custo)),
            R),
   sum(R, C).
```

3.5 Funcionalidades e Conhecimento Adicional

Uma vez que o achamos oportuno introduzir algumas funcionalidades fora do espectro pré-definido, com vista a melhorar e completar o funcionamento do projecto, achamos viável adicionar algum conhecimento referente às instituições prestadoras de serviços bem como o conhecimento sobre médicos que realizam as consultas.

3.5.1 Conhecimento Adicional

Uma instituição é caracterizada pelo seu identificador único, Nome e Cidade.

```
% Extensao do predicado instituicao: #Id, Nome, Cidade -> {V, F}

instituicao(0, 'Hospital de Braga',      'Braga').
instituicao(1, 'Hospital Sao Joao',      'Porto').
instituicao(2, 'Hospital de Santa Maria', 'Porto').
instituicao(3, 'Hospital de Santa Maria', 'Lisboa').
instituicao(4, 'Hospital Santa Maria Maior', 'Barcelos').
instituicao(5, 'Hospital Escala',        'Braga').
instituicao(6, 'Hospital de Faro',       'Faro').
instituicao(7, 'Hospital de Famalicao',  'Famalicao').
```

Um médico é caracterizado pelo seu identificador único, nome, idade e identificador da instituição a que pertence.

```
% Extensao do predicado medico: #Id, Nome, Idade, #IdInstituicao -> {V,
    F}

medico(0, 'Alberto', 35, 3).
medico(1, 'Ana',     28, 2).
medico(2, 'Roberto', 58, 7).
medico(3, 'Josefina', 42, 5).
medico(4, 'Helena',  27, 0).
medico(5, 'Nuno',    37, 1).
medico(6, 'Margarida', 48, 6).
medico(6, 'Margarida', 31, 4).
```

Para ir de encontro ao conhecimento agora adicionado, desenvolvemos um conjunto de invariantes de forma a manter a consistência do sistema, garantindo que a inserção de conhecimento sobre instituições e médicos é feita de forma correta.

3.5.2 Invariante Estrutural - Inserir Instituição

De forma a não permitir a inserção de instituições com o mesmo *id*, e para que este identificador seja inteiro, desenvolveu-se o seguinte invariante:

```
% Invariante Estrutural: No permitir a insercao de instituicoes
    repetidas (com o mesmo ID)
% Id inteiros

+instituicao(IdInst, _, _) :: (
    integer(IdInst),
    solucoes(IdInst, instituicao(IdInst, _, _), S),
    comprimento(S, N),
    N == 1
).
```

3.5.3 Invariante Referencial - Remover Instituição

Para remover uma instituição, é preciso garantir que não existem médicos ou serviços associadas à mesma, sendo então desenvolvido o seguinte invariante:

```
% Invariante Referencial: Nao permitir a remocao de uma instituicao se
    existirem
% servicos ou medicos associadas a mesma

-instituicao(IdInst, _, _) :: (
   solucoes(IdInst, servico(_, _, IdInst, _), S1),
   solucoes(IdInst, medico(_, _, _, IdInst), S2),
   comprimento(S1, N1),
   comprimento(S2, N1),
   N1 == 0
).
```

3.5.4 Invariante Estrutural - Inserir Médico

De forma a garantir que o identificador do médico e a sua idade são inteiros, que o identificador do médico ainda não existe e que o identificador da instituição corresponda a uma instituição presente no sistema, desenvolveu-se o seguinte invariante:

```
% Invariante Estrutural: No permitir a insercao de instituicoes
    repetidas (com o mesmo ID)
% Id inteiros

+medico(Id, _, Idade, IdInst) :: (
   integer(Id), integer(Idade),
   Idade >= 0, Idade =< 120,
   solucoes(Id, medico(Id, _, _, _), S1),
   solucoes(IdInst, instituicao(IdInst, _, _), S2),
   comprimento(S1, N1),
   comprimento(S2, N2),
   N1 == 1, N2 == 1
```

```
).
```

3.5.5 Invariante Referencial - Remover Médico

Para remover um médico casa este exista no sistema foi desenvolvido o seguinte invariante:

```
% Invariante Referencial: Remover um medico se este existir

-medico(Id, Nome, Idade, IdInst) :: (
   solucoes((Id, Nome, Idade, IdInst),
             medico(Id, Nome, Idade, IdInst), S1),
   comprimento(S1, N1),
   N1 = 0
).
```

3.5.6 Funcionalidades Extra

Ao longo da execução deste trabalho, é possível observar que tanto Instituição como Médico são utilizados na construção dos predicados base pedidos no enunciado. Apesar de as Instituições serem algo que já nos é dado, foi criado todo o conhecimento sobre as mesmas, o que faz com que possamos desenvolver predicados mais eficazes e que nos apresentam mais informação, do que um simples nome de Instituição. De forma semelhante, o conhecimento do Médico foi adicionado aos predicados base, passando assim a ser possível identificar coisas como serviços e consultas por médico. De seguida, temos os predicados que servem para a identificação dos Médicos dado um critério de selecção.

Para identificar médicos através do *id*, nome, idade, Instituição foi desenvolvido o seguinte conjunto de extensões de predicado:

```
% Extenso do Predicado idMedicoID: ID, Lista -> {V,F}
idMedicoID(ID, L) :- solucoes((ID, Nome, Idade, IdInst), medico(ID,
    Nome, Idade, IdInst), L).

% Extenso do Predicado idMedicoIdade: Nome, Lista -> {V,F}
idMedicoNome(Nome, L) :- solucoes((ID, Nome, Idade, IdInst), medico(ID,
    Nome, Idade, IdInst), L).

% Extenso do Predicado idMedicoIdade: Idade, Lista -> {V,F}
idMedicoIdade(Idade, L) :- solucoes((ID, Nome, Idade, IdInst),
    medico(ID, Nome, Idade, IdInst), L).

% Extenso do Predicado idMedicoInstituicao: Morada, Lista -> {V,F}
idMedicoInstituicao(Instituicao, L) :- solucoes((ID, Nome, Idade,
    IdInst), medico(ID, Nome, Idade, IdInst), L).
```

Conclusão e Sugestões

A nosso ver, tanto o objetivo fundamental do trabalho como a adição de conhecimento e predicados extra, foram cumpridos. Neste sentido, achamos que este trabalho foi importante para a consolidação dos conceitos relativos ao *PROLOG* e às variantes que este apresenta, bem como da relevância do mesmo na abordagem de problemas/desafios.

Posto isto, achamos que este novo conhecimento adquirido é crucial para abordagem de situações que requerem rigor e restrição no âmbito da criação e desnvolvimento de sistemas de representação de conhecimento e raciocínio, sendo que nos permite restringir e otimizar o funcionamento do mesmo sistema de forma objectiva e consistente.

Para o futuro, consideramos que seria possível expandir o sistema, tratando a informação sobre vários outros aspectos referentes à área de cuidados de saúde, como outro tipo de funcionários, tratamento de receitas, etc.

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;

Anexos

```
| ?- listing(utente).
utente(0, 'Elias', 28, 'Barcelos').
utente(1, 'Sebastiao', 18, 'Porto').
utente(2, 'Martim', 32, 'Braga').
utente(3, 'Lucia', 55, 'Guimaraes').
utente(4, 'Alexandre', 76, 'Lisboa').
utente(5, 'Juliana', 22, 'Porto').
utente(6, 'Joao', 8, 'Famalicao').
utente(7, 'Antonio', 16, 'Viseu').
utente(8, 'Isabela', 36, 'Vila do Conde').
utente(9, 'Artur', 51, 'Setubal').
utente(10, 'Alice', 86, 'Povoa de Varzim').

yes
| ?- evolucao(utente(10,'Ricardo',14,'Povoa de Varzim')).
no
| ?- evolucao(utente(11,'Ricardo',14,'Povoa de Varzim')).
yes
| ?- listing(utente).
utente(0, 'Elias', 28, 'Barcelos').
utente(1, 'Sebastiao', 18, 'Porto').
utente(2, 'Martim', 32, 'Braga').
utente(3, 'Lucia', 55, 'Guimaraes').
utente(4, 'Alexandre', 76, 'Lisboa').
utente(5, 'Juliana', 22, 'Porto').
utente(6, 'Joao', 8, 'Famalicao').
utente(7, 'Antonio', 16, 'Viseu').
utente(8, 'Isabela', 36, 'Vila do Conde').
utente(9, 'Artur', 51, 'Setubal').
utente(10, 'Alice', 86, 'Povoa de Varzim').
utente(11, 'Ricardo', 14, 'Povoa de Varzim').

yes
```

Figure 1: Exemplo de Inserção de Utente

```
| ?- involucao(utente(12,'Ricardo',14,'Povoa de Varzim')).
no
| ?- involucao(utente(11,'Ricardo',14,'Povoa de Varzim')).
yes
| ?- listing(utente).
utente(0, 'Elias', 28, 'Barcelos').
utente(1, 'Sebastiao', 18, 'Porto').
utente(2, 'Martim', 32, 'Braga').
utente(3, 'Lucia', 55, 'Guimaraes').
utente(4, 'Alexandre', 76, 'Lisboa').
utente(5, 'Juliana', 22, 'Porto').
utente(6, 'Joao', 8, 'Famalicao').
utente(7, 'Antonio', 16, 'Viseu').
utente(8, 'Isabela', 36, 'Vila do Conde').
utente(9, 'Artur', 51, 'Setubal').
utente(10, 'Alice', 86, 'Povoa de Varzim').

yes
| ?-
```

Figure 2: Exemplo de Remoção de Utente

```
| ?- idUtenteID(4,L).
L = [(4,'Alexandre',76,'Lisboa')] ?
yes
```

Figure 3: Exemplo de procura de Utente pelo ID

```
| ?- identifica_instituicoes(L).
L = [(0,'Hospital de Braga','Braga'),(1,'Hospital Sao Joao','Porto'),(2,'Hospital de Santa Maria','Porto'),(3,'Hospital de Santa Maria','Li
sboa'),(4,'Hospital Santa Maria Maior','Barcelos'),(5,'Hospital Escala','Braga'),(6,'Hospital de Faro','Faro'),(7,'Hospital de Famalicao','
Famalicao')] ?
yes
```

Figure 4: Identificar instituições prestadoras de serviços

```
| ?- listaInfoUtentes([1,2,3,4],L).
L = [[(1,'Sebastiao',18,'Porto')],[(2,'Martim',32,'Braga')],[(3,'Lucia',55,'Guimaraes')],[(4,'Alexandre',76,'Lisboa')]] ?
yes
```

Figure 5: Funcionalidade listaInfoUtentes

```
| ?- servicosPorCid('Porto',L).
L = [(0,'Dermatologia','Porto'),(1,'Psiquiatria','Porto')] ?
yes
```

Figure 6: Exemplo de listagem dos serviços disponíveis por cidade

```
| ?- servicosRealizCid('Porto',L).
L = [(data(20,2,2019),0,'Dermatologia','Porto'),(data(1,5,2019),0,'Dermatologia','Porto'),(data(12,5,2019),0,'Dermatologia','Porto'),(data
10,4,2019),1,'Psiquiatria','Porto'),(data(13,4,2019),1,'Psiquiatria','Porto')] ?
yes
```

Figure 7: Identificar serviços realizados por cidade

```
| ?- custoTotal(1,A,B,C,D,F).
F = 105 ?
yes
```

Figure 8: Cálculo do custo total dos cuidados de saúde por Id de utente