Universidade do Minho
Mestrado Integrado em Engenharia Informática

Unidade Curricular de Sistemas de Representação de Conhecimento e Raciocínio

Ano Letivo de 2017/2018

1.º Exercício de Grupo

(Programação em Lógica e Invariantes)

Diogo Emanuel da Silva Nogueira (a78957)

Fábio Quintas Gonçalves (a78793)

Sarah Tifany da Silva (a76867)

Grupo 8

Março, 2018

Resumo

O presente relatório aglomera todo o trabalho e posterior preparação empregue na resolução deste primeiro exercício prático de grupo.

Nele, expõe-se todo o procedimento por detrás da realização alusiva a este exercício, os seus auxílios prévios, bem como a sua proposta de resolução devidamente fundamentada e sustentada pelos resultados/cálculos intermédios efetuados pertinentes para o efeito.

Índice

1. Introdução

O tema principal deste trabalho prático foca-se na aplicação de todo o conhecimento passado até então relativamente à Programação em Lógica e Invariantes. Assim, neste relatório pretende-se demonstrar todos os resultados provenientes do desenvolvimento de um Sistema de Representação de Conhecimento e Raciocínio que seja capaz de caracterizar todo um universo de discurso na área da prestação de cuidados de saúde, passando pelas instituições, utentes e os diversos cuidados médicos efetuados pelos demais.

Projeta-se assim, com o desenvolver deste exercício de grupo, toda uma solidificação no que diz respeito às capacidades de se representar um conjunto tão diversificado de informações, com o uso da linguagem de programação em lógica PROLOG.

Pondo em prática todo este conhecimento adquirido, irão desenvolver-se os predicados e invariantes, respeitando-se as necessidades de demonstração das várias funcionalidades requeridas para o sucesso deste Sistema.

2. Preliminares

Este trabalho prático teve desde início como suporte todo o conhecimento absorvido à medida que se foi praticando a linguagem de programação em lógica PROLOG e também o desenvolvimento das várias invariantes. Por dizer respeito a um tema do quotidiano transporta uma bagagem de sabedoria prévia, o que acaba por auxiliar na compreensão e posterior resposta/satisfação das múltiplas necessidades.

Com uma leitura atenta do enunciado deste primeiro exercício e usando também como sustento a análise de trabalhos desenvolvidos neste âmbito não só nas aulas práticas, mas também fora delas, foi mais do que possível desenvolver um trabalho consistente, funcional e que acima de tudo respeitasse o desejado.

Conhecendo-se este tema e tendo-se presente toda a ideia de como funciona a prestação dos vários cuidados de saúde associados ao seu utente e ao seu devido prestador, é perfeitamente viável compreender como todo o Sistema funciona e consequentemente como resolvê-lo.

Para este fim, foram criadas três bases de conhecimento que inicialmente foram estabelecidas e nas quais se vai basear todo um cenário dirigido a esta área:

- `utente: (#IDU, Nome, Idade, Cidade) -> {V,F}`

O Utente é uma das personagens intervenientes no Sistema e tem associado a si um ID e naturalmente um nome, idade e cidade.

- `prestador: (#IDP, Nome, Instituicao) -> {V,F}`

O Prestador é a segunda personagem interveniente no Sistema e possui também um ID e um nome. A única singularidade está na associação à instituição a que o mesmo se encontra vinculado de modo a se poder

estabelecer uma conexão entre o Prestador e o local onde o mesmo opera.

- *cuidado: (Data, #IDU, #IDP, Custo) -> {V,F}*

O Cuidado de Saúde trata de fazer toda uma união dos dois conhecimentos anteriores. Ao construirmos esta "ponte" e também com a informação da data, eficientemente se consegue saber quando e com que custo um determinado cuidado foi efetuado a um dado Utente.

Após esta abordagem geral é importante deixar esclarecido que tanto o ID do Utente como o ID do Prestador devem ser únicos, identificando cada um deles. Esta particularidade implica que não possa existir Utentes e Prestadores com o mesmo ID.

Contudo, e falando agora do Cuidado de Saúde, existe a possibilidade de um cuidado possuir um conjunto de ID's semelhantes, já que esta base de conhecimento é uma associação entre o Utente e o Prestador. Por outras palavras, quer-se dizer que um Utente pode ser submetido ao mesmo Cuidado de Saúde, quantas vezes se quiser.

3. Descrição do Trabalho e Análise dos Resultados

Nesta secção irá ser discutida e devidamente fundamentada a resolução deste primeiro exercício de grupo. Vamos explicitar toda a construção, etapa por etapa, deste caso prático, usando para efeito o excerto dos vários predicados e invariantes, no sentido de se produzir uma análise de resultados atenta e explícita.

Para uma maior perceção de como todo este caso foi desenvolvido, fez-se uma repartição desta secção principal em subseções, que nada mais são que as várias etapas que desde início foram pensadas e moldadas para irem de encontro com este relatório.

Além das funcionalidades e características do Sistema inicialmente solicitadas, quer ao nível das capacidades de representação de conhecimento quer ao nível das faculdades de raciocínio, vamos também apresentar os predicados extras que surgiram como ideia para este caso prático e que de algum modo valorizam ainda mais todo o nosso Sistema.

3.1. Base de Conhecimento Inicial

A base de conhecimento inicial é um passo elementar e imprescindível para que se possa numa fase posterior efetuar os vários testes de toda a informação sem existir a obrigação/necessidade de inserir conhecimento. A nossa base de conhecimento foi construída tendo-se em mente as várias funcionalidades pedidas para o Sistema em si, de modo a se poder também visualizar eventuais erros que pudessem surgir no decorrer de todo o exercício.

Naturalmente, existiram cuidados específicos que foram de acordo com as demais particularidades dos diferentes predicados:

- A existência de vários Utentes com o mesmo nome, idade e cidade pensada na listagem dos vários Utentes por critérios de seleção;
- A introdução de Cuidados de Saúde associados à mesma instituição, Utente e Prestador, de modo a se visualizar mais consistentemente a identificação dos vários cuidados efetuados por estes três parâmetros;
- A existência de vários Cuidados de Saúde efetuados na mesma data possibilitando-se a identificação dos mesmos pelas diversas datas existentes;
- Entre outros…

Subsequentemente encontram-se os excertos onde podemos efetivamente comprovar estas particularidades impostas nas várias informações, bem como toda a estrutura daquilo que é cada uma das bases de conhecimento construídas para o efeito. Neste ponto do relatório, apenas destacaremos 4 informações de cada uma das bases, impelindo o excerto completo para a secção 5 (Anexos) com vista a evitar a saturação de informação.

```
% Base de Conhecimento Utente (IDU,Nome,Idade,Cidade).
utente(6472,'Diogo Nogueira',20,'Fafe').
utente(1211,'Diogo Nogueira',21,'Porto')
utente(8374,'Tifany Silva',24,'Porto').
utente(3463,'Fabio Fontes',19,'Braga').
utente(2321,'Mariana Lino',20,'Guimaraes').
```

```
% Base de Conhecimento Prestador (IDP,Nome,Instituicao).
prestador(4343,'Navarro Silva','Dermatologia','Hospital Privado de
Guimaraes').
prestador(3947,'Margarida Antunes','Dentista','Hospital Privado de
Guimaraes').
prestador(3643,'Nuno Costa','Nutricao','Classaude Porto').
```

```
prestador(7364,'Ana Maria','Hidroterapia','Clinica Fisiatrica de Fafe').
prestador(3642,'Tomas Casimiro','Clínica Geral','Classaude Porto').

% Base de Conhecimento Cuidado de Saúde (Data,IDU,IDP,Custo).
cuidado(date(2018,11,12),6472,4343,'Carie',12).
cuidado(date(2017,10,25),8374,7324,'Consulta de rotina aparelho',20).
cuidado(date(2016,09,15),3463,7864,'Queimadura na cara',15).
cuidado(date(2018,11,12),6472,3742,'Consulta',50).
cuidado(date(2015,01,25),9232,8675,'Consulta agendada/rotina',55).
```

3.2. Predicados de evolução e retrocesso de conhecimento

Os predicados de evolução e retrocesso são os predicados base para a possibilidade de se interferir com as bases de conhecimento anteriormente discutidas. Estes predicados garantem toda a solidez das bases de conhecimento ao recorrer ao uso dos invariantes que têm como objetivo garantir que não ocorra a inserção e remoção de conhecimento indesejado.

```
% Inserção de Conhecimento.
insercao(Termo) :- assert(Termo).
insercao(Termo) :- retract(Termo),!,fail.

% Remoção de Conhecimento.
remocao(Termo) :- retract(Termo).
remocao(Termo) :- assert(Termo),!,fail.

% Teste.
teste([]).
teste([R|LR]) :- R,teste(LR).

% Controlo da inserção de Conhecimento.
evolucao(Termo) :- findall(Invariante,+Termo::Invariante,Lista),
                   insercao(Termo),
                   teste(Lista).

% Controlo da remoção de Conhecimento.
retrocesso(Termo) :- Termo,
                     findall(Invariante,+Termo::Invariante,Lista),
                     remocao(Termo),
```

```
teste(Lista).
```

Tanto no momento da inserção como no momento da remoção de conhecimento, estes predicados testam os tais invariantes (que iremos debater na seção seguinte), impedindo qualquer ação imposta, caso um invariante não se verifique como esperado.

3.3. Invariantes

Conforme abordado na secção anterior, os invariantes são completamente indispensáveis para um correto funcionamento de todo o nosso Sistema. Apenas com a sua introdução é que somos capazes de efetuar o controle da informação que entra ou sai, criando um intermediário necessário para que os predicados de evolução e retrocesso funcionem como ambicionado.

Para esta gestão de informação se puder processar da maneira mais adequada e lógica face ao nosso Sistema, foram criados invariantes associados tanto à inserção como à remoção de conhecimento.

1) Invariantes associados à inserção de conhecimento:

No que diz respeito ao conjunto de invariantes associados à inserção criamos um conjunto que impede a inserção de conhecimento repetido, tanto por parte do Utente como do Prestador. Não se procedeu de igual modo para o Cuidado de Saúde pois consideramos possível a inserção de cuidados pertencentes ao mesmo Utente e Prestador.

```
% Não permitir a inserção de conhecimento repetido relativo ao Utente.
+utente(IdUt,Nome,Idade,Morada) ::
(findall(IdUt,(utente(IdUt,_,_,_)),L),
                                comprimento(L,C), C =< 1).
```

```
% Não permitir a inserção de conhecimento repetido relativo ao
Prestador.
+prestador(IdP,Nome,Especialidade,Instituicao) ::
(findall(IdP,prestador(IdP,_,_,_),L),
                                comprimento(L,C), C =< 1).
```

Adicionalmente, também foi necessário produzir um invariante que não possibilitasse a inserção de informação impossível de se introduzir na base de conhecimento em questão. No fundo, garantir que ao introduzir um Cuidado de Saúde, este pertencesse a um Utente e Prestador existentes na base de conhecimento.

```
% Não permite a inserção de conhecimento que não exista na Base de
Conhecimento.
+cuidado(D,IDU,IDP,C) :: (findall(IDU, utente(IDU,_,_,_), S),
                          comprimento(S,N), N == 1).
+cuidado(D,IDU,IDP,C) :: (findall(IDP, prestador(IDP,_,_,_), S),
                          comprimento(S,N), N == 1).
```

2) Invariantes associados à remoção de conhecimento:

Para a remoção de conhecimento apenas foi oportuno desenvolver invariantes que perante toda a lógica do funcionamento dos cuidados de saúde, não permitissem a remoção de Utentes e Prestadores quando estes se encontrassem associados a um determinado Cuidado de Saúde.

Não foi necessário criar invariantes que, igualmente ao que foi feito para a inserção, controlassem a remoção de informação inexistente em cada uma das bases de conhecimento. Ao colocarmos a cláusula `Termo` em "primeiro plano" no predicado `retrocesso` abordado na secção 3.2 (Predicados de evolução e retrocesso de conhecimento), já estamos a garantir isso. Basicamente, fazer o `retrocesso` já implica que o termo exista. Caso não exista, já não é feita a remoção.

```
% Não permite a remoção de Utentes quando existe Cuidados de Saúde
associados aos mesmos.
-utente(IDU,Nome,Idade,Morada) :: (findall(IDU,cuidado(_,IDU,_,_,_),L),
                                   comprimento(L,X),
                                   X == 0).

% Não permite a remoção de Prestadores quando existe Cuidados de Saúde
associados aos mesmos.
-prestador(IDP,Nome,Especialidade,Instituicao) ::
(findall(IDP,cuidado(_,_,IDP,_,_),L),
                                                comprimento(L,X),
                                                X == 0).
```

3.4. Predicados de registo e remoção de conhecimento

Os predicados de registo e remoção de conhecimento são os predicados que permitem a inserção e remoção de um Utente, Prestador e também Cuidado de Saúde, pela recorrência aos predicados de evolução e retrocesso criados na secção 3.2 (Predicados de evolução e retrocesso de conhecimento). Assim sendo, são equivalentes, diferenciando-se apenas na maneira concreta e percetível como estão definidos.

1) Predicados de registo de Utente, Prestador e Cuidado de Saúde:

Recebem os parâmetros de um dos predicados (Utente, Prestador ou Cuidado de Saúde) existentes, transportando para o predicado `evolucao(Termo)` de modo a garantir a inserção na base de conhecimento de forma verificada e controlada pelos invariantes previamente discutidos.

```
% Extensão do predicado registar: Termo -> {V,F}
registar(T) :- evolucao(T).
```

```
% Extensão do predicado registarUtente: IdUt, Nome, Idade, Morada ->
{V,F}
registarUtente(IdUt,Nome,Idade,Morada) :-
        evolucao(utente(IdUt,Nome,Idade,Morada)).

% Extensão do predicado registarPrestador: IdP, Nome, Especialidade,
Instituicao -> {V,F}
registarPrestador(IdP,Nome,Especialidade,Instituicao) :-
        evolucao(prestador(IdUt,Nome,Idade,Morada)).

% Extensão do predicado registarCuidado: Data, IdUt, IdP, Descricao,
Custo  -> {V,F}
registarCuidado(Data,IdUt,IdP,Descricao,Custo) :-
        evolucao(cuidado(Data,IdUt,IdP,Descricao,Custo)).
```

2) Predicados de remoção de Utente, Prestador e Cuidado de Saúde:

Recebem também os parâmetros de um dos predicados (Utente, Prestador ou Cuidado de Saúde) existentes, mas neste caso passam estes parâmetros para o predicado `retrocesso(Termo)` que se encarregará de efetuar a remoção, também de forma segura e controlada.

```
% Extensão do predicado remover: Termo -> {V,F}
remover(T) :- retrocesso(T).

% Extensão do predicado removerUtente: IdUt, Nome, Idade, Morada ->
{V,F}
removerUtente(IdUt,Nome,Idade,Morada) :-
        retrocesso(utente(IdUt,Nome,Idade,Morada)).

% Extensão do predicado removerPrestador: IdP, Nome, Especialidade,
Instituicao -> {V,F}
removerPrestador(IdP,Nome,Especialidade,Instituicao) :-
        retrocesso(prestador(IdUt,Nome,Idade,Morada)).

% Extensão do predicado removerCuidado: Data, IdUt, IdP, Descricao,
Custo  -> {V,F}
removerCuidado(Data,IdUt,IdP,Descricao,Custo) :-
        retrocesso(cuidado(Data,IdUt,IdP,Descricao,Custo)).
```

3.5. Predicados que listam conhecimento

1) Identificar Utentes por critérios de seleção:

Para se proceder à identificação dos Utentes recorreu-se apenas a 3 critérios de seleção, deixando-se de parte a identificação via ID já que desde início foi convencionado que o ID era um valor único e de identificação pessoal.

```
% Extensão do predicado findUtentesN: Nome, R -> {V,F}
findUtentesN(Nome,R) :-
        findall((IdU,Nome,Idade,Morada),utente(IdU,Nome,Idade,Morada),R)
.
% Extensão do predicado findUtentesI: Idade, R -> {V,F}
findUtentesI(Idade,R) :-
        findall((IdU,Nome,Idade,Morada),utente(IdU,Nome,Idade,Morada),R)
.
% Extensão do predicado findUtentesM: Morada, R -> {V,F}
findUtentesM(Morada,R) :-
        findall((IdU,Nome,Idade,Morada),utente(IdU,Nome,Idade,Morada),R)
.
```

Em cada um destes predicados apenas foi necessário aplicar o predicado `findall` (já predefinido no Prolog), que trata de filtrar a informação da base de conhecimento em questão, agrupando o conjunto de parâmetros dos Utentes que satisfazem o predicado `utente(IdU,Nome,Idade,Morada)`e que possuem um dado parâmetro de seleção: nome, idade ou morada.

2) Identificar as instituições prestadoras de Cuidados de Saúde:

Igualmente ao que se sucede no ponto anterior, é necessário a recorrência ao predicado `findall` por fim a se adquirir todo o conjunto de instituições existentes na base de conhecimento que estão associadas aos vários prestadores de serviço. De forma mais técnica, equivale a dizer que estamos a adquirir todos os parâmetros `Instituicao` que satisfazem o predicado `(_,_,_,Instituicao)`.

```
% Extensão do predicado findInstituicoes: R -> {V,F}.
findInstituicoes(R) :-
findall(Instituicao,prestador(_,_,_,Instituicao),L),
            removeDuplicados(L,R).
```

Tal como se espera, este conjunto de instituições irá conter informação repetida já que diferentes Prestadores podem operar no mesmo centro médico. Para contornar isto, "transporta-se" a lista anteriormente obtida ao predicado `removeDuplicados` que trata de eliminar os tais componentes duplicados.

3) Identificar Cuidados de Saúde prestados por instituição/cidades/datas:

Para a resolução deste predicado, segue-se o raciocínio base usado até então, mas, necessitando-se de um pouco mais de ponderação devido à complexidade acentuada nestas identificações.

Para se listar os vários Cuidados de Saúde prestados por instituição, foi necessário implementar um predicado auxiliar, uma vez que a instituição onde um determinado Cuidado de Saúde ocorre não é um parâmetro da base de conhecimento `cuidado`:

- Através do predicado `findall`, obtemos o conjunto de todos os ID's dos Prestadores que satisfazem o predicado `prestador(IDP,_,_,Inst)`, em que `Inst` é a instituição da qual se pretende obter os cuidados de saúde, ou seja, os ID's de todos os Prestadores que trabalham na Instituição passada como parâmetro.
- Após isso, passa-se essa mesma lista ao predicado auxiliar `cuiInstAux`, que trata de obter os parâmetros `(Data,IDU,IDP,D,C)`, ID a ID, com o auxílio do predicado auxiliar `concat` para os unir.

```
% Extensão do predicado cuidadoPorInst: Inst, R -> {V,F}
cuidadoPorInst(Inst,R) :- findall((IDP),prestador(IDP,_,_,Inst) , L),
                          cuiInstAux(L,R).

% Extensão do predicado cuiInstAux: L,R -> {V,F}
cuiInstAux([],[]).
cuiInstAux([IDP|T],R) :-
findall((Data,IDU,IDP,D,C),cuidado(Data,IDU,IDP,D,C),L1),
                          concat(L1,L2,R),
                          cuiInstAux(T,L2).
```

Para a listagem dos diferentes Cuidados de Saúde por cidade o procedimento é o mesmo. A única desigualdade está na obtenção dos ID's referentes aos Utentes e não aos Prestadores, uma vez que a cidade é um parâmetro que se encontra declarado no base de conhecimento do Utente.

```
% Extensão do predicado cuidadoPorCid: Cid, R -> {V,F}
cuidadoPorCid(Cid,R) :- findall((IDU),utente(IDU,_,_,Cid) , L),
                        cuiCidAux(L,R).

% Extensão do predicado cuiCidAux: L, R -> {V,F}
cuiCidAux([],[]).
cuiCidAux([IDU|T],R) :-
findall((Data,IDU,IDP,D,C),cuidado(Data,IDU,IDP,D,C),L1),
                          concat(L1,L2,R),
                          cuiCidAux(T,L2).
```

Para se listar os vários Cuidados de Saúde prestados por datas evita-se a criação de um predicado auxiliar já que a data é um parâmetro do predicado `cuidado`. Apenas se necessita de obter a lista dos vários cuidados que coincidem com a data que é passada ao predicado `cuidadoPorData`.

```
% Extensão do predicado cuidadoPorData: Data, R -> {V,F}
cuidadoPorData(Data,R) :-
findall((Data,IU,IP,D,C),(cuidado(Data,IU,IP,D,C)),R).
```

4) Identificar os Utentes de um prestador/especialidade/instituição:

A identificação de Utentes por parte destes critérios não se distancia dos demais predicados construídos até então.

- Quando estamos a identificar Utentes por Prestador, necessitamos de recorrer ao predicado `cuidado` de modo a obter os ID's dos Utentes que cumpram o predicado `cuidado(_,IDU,IDP,_,_)`, sendo que o `IDP` é o ID passado como parâmetro.
- Para finalizar, evoca-se o predicado auxiliar `utePresAux` que trata de conseguir os parâmetros `(IDU,Nome,I,C)` de cada um desses ID's obtidos, fazendo uso do predicado `concat` para a concatenação do conjunto de tuplos de Utentes encontrados.

```
% Extensão do predicado utentePorPres: IDP, R -> {V,F}
utentePorPres(IDP,R) :- findall((IDU),cuidado(_,IDU,IDP,_,_),L),
                        utePresAux(L,R).

% Extensão do predicado utePresAux: L, R -> {V,F}
utePresAux([],[]).
utePresAux([IDU|T],R) :-
findall((IDU,Nome,I,C),utente(IDU,Nome,I,C),L1),
                         concat(L1,L2,R),
                         utePresAux(T,L2).
```

Em contrapartida, quando estamos a identificar Utentes por especialidade há a necessidade de recorrer ao predicado auxiliar `uteEspAux` e de seguida ao predicado auxiliar `utePresAux` que anteriormente tinha sido definido:

- Inicialmente, obtemos a lista de todos os ID's de Prestadores que obedecem ao predicado `(IDP,_,Esp,_)`, onde `Esp` é a especialidade que queremos associar;
- Numa segunda fase, tratamos de obter a lista de todos os ID's de Utentes que obedecem ao predicado `(_,IDU,IDP,_,_)`, onde `IDP` é cada um dos ID's anteriores;

- Por fim, com o predicado auxiliar `utePresAux` anterior, retiram-se todos os parâmetros dos Utentes cujo ID esteja na lista anteriormente obtida.

Para identificar Utentes por instituições é exatamente igual a identificar por especialidade sendo que a única coisa que muda é o parâmetro passado ao predicado principal.

```
% Extensão do predicado utentePorEsp: Esp, R -> {V,F}
utentePorEsp(Esp,R) :- findall((IDP),prestador(IDP,_,Esp,_),L),
                       uteEspAux(L,X),
                       utePresAux(X,R).


% Extensão do predicado utentePorIns: Ins, R -> {V,F}
utentePorIns(Ins,R) :- findall((IDP),prestador(IDP,_,_,Ins),L),
                       uteEspAux(L,X),
                       utePresAux(X,R).

% Extensão do predicado uteEspAux: L, R -> {V,F}
uteEspAux([],[]).
uteEspAux([IDP|T],R) :- findall((IDU),cuidado(_,IDU,IDP,_,_),L1),
                        concat(L1,L2,R),
                        uteEspAux(T,L2).
```

5) Identificar Cuidados de Saúde realizados por utente/instituição/prestador:

Pelo que se pode perceber até agora, é necessário recorrer a predicados auxiliares quando não é possível a obtenção direta daquilo que se pretende, ou seja, quando o parâmetro que estamos a passar ao predicado "principal" não é um parâmetro da base de conhecimento da qual queremos obter informação.

Nesta funcionalidade, apenas podemos estabelecer esta conexão direta nos cuidados efetuados por Utente e Prestador já que cada Cuidado de Saúde tem como parâmetros o ID do Utente e ID do Prestador.

```
% Extensão do predicado cuidadosPorUt: IDU, R -> {V,F}
cuidadosPorUt(IDU,R) :-
findall((Data,IDU,IDP,X,Y),cuidado(Data,IDU,IDP,X,Y),R).

% Extensão do predicado cuidadosPorPres: IDU, R -> {V,F}
cuidadosPorPres(IDP,R) :-
findall((Data,IDU,IDP,X,Y),cuidado(Data,IDU,IDP,X,Y),R).
```

Para o conjunto dos Cuidados de Saúde realizados por instituição existiu a necessidade de se recorrer a um predicado auxiliar uma vez que a instituição é um parâmetro do predicado `prestador` e não do predicado `cuidado`. Tal como se procedeu até agora:

- Primeiro obtém-se a lista de ID's dos prestadores que satisfazem o predicado `prestador(IDP,_,_,Inst)`, onde `Inst` é a instituição em causa;
- Por fim, recorre-se ao predicado `cuiInstAux` (que já tinha sido mencionado em alíneas anteriores), e pega-se nos parâmetros do predicado `cuidado` através da associação de cada um dos ID's listados inicialmente.

```
% Extensão do predicado cuidadosPorInst: Ins, R -> {V,F}
cuidadosPorInst(Inst,R) :- findall((IDP),prestador(IDP,_,_,Inst),L),
                           cuiInstAux(L,R).

% Extensão do predicado cuiInstAux: L, R -> {V,F}
cuiInstAux([],[]).
cuiInstAux([IDP|T],R) :-
findall((Data,IDU,IDP,D,C),cuidado(Data,IDU,IDP,D,C),L1),
                        concat(L1,L2,R),
                        cuiInstAux(T,L2).
```

6) **Determinar todas as instituições/prestadores a que um utente já recorreu:**

Aqui, os passos são análogos, tanto para a listagem das instituições como dos Prestadores:

- Obtém-se a lista dos ID's de todos os prestadores que satisfazem o predicado `cuidado(_,IDU,IDP,_,_)`;
- Recorre-se a um predicado auxiliar que trata de obter a lista das instituições associadas a cada um dos ID's acima para as instituições e recorre-se a um predicado auxiliar que trata de obter os parâmetros base de um Prestador associados também a cada um dos ID's de cima no caso dos prestadores.

```
% Extensão do predicado instituicoesPorUt: IDU, R -> {V,F}
instituicoesPorUt(IDU,R) :- findall((IDP),cuidado(_,IDU,IDP,_,_),L),
                            instUtAux(L,X),
                            removeDups(X,R).

% Extensão do predicado instUtAux: IDP, R -> {V,F}
instUtAux([],[]).
instUtAux([IDP|T],R) :- findall((Ins),prestador(IDP,_,_,Ins),L1),
                        concat(L1,L2,R),
                        instUtAux(T,L2).

% Extensão do predicado prestadoresPorUt: IDU, R -> {V,F}
prestadoresPorUt(IDU,R) :- findall((IDP),cuidado(_,IDU,IDP,_,_),L),
                           presUtAux(L,X),
                           removeDups(X,R).

% Extensão do predicado presUtAux: IDP, R -> {V,F}
presUtAux([],[]).
presUtAux([IDP|T],R) :-
findall((IDP,N,E,Ins),prestador(IDP,N,E,Ins),L1),
                        concat(L1,L2,R),
                        presUtAux(T,L2).
```

Para se contornar a obtenção de conhecimento duplicado, faz-se partido do predicado `removeDups` tanto para a listagem das instituições como para a listagem de Prestadores.

7) Calcular o custo total dos cuidados de saúde por utente/especialidade/prestador/datas:

Na listagem de conhecimento desta alínea existiu uma abordagem distinta das anteriores. Apesar de se fazer igualmente uso do predicado `findall` para se obter o conjunto de informação desejado, foi adicionalmente necessário ir além desta simples listagem de conhecimento. Isto conseguiu-se, mais uma vez, recorrendo-se a um predicado auxiliar capaz de interpretar este conhecimento, ao efetuar todos os custos obtidos em cada uma destas 4 situações.

Para as primeiras 3 situações, o que se faz é gerar uma lista com os custos associados ao parâmetro passado.

- Numa primeira fase, percorre-se todos os cuidados existentes e seleciona-se apenas os que fazem correspondência com o parâmetro fornecido, obtendo-se apenas o parâmetro `Custo` de cada uma das correspondências;
- Por fim, faz-se uso do predicado `somaLista` para se unificar todos os custos.

```
% Extensão do predicado custoTotalUt: IDU, T -> {V,F}
custoTotalUt(IDU,T) :- findall((Custo),cuidado(_,IDU,_,_,Custo),L),
                       somaLista(L,T).

% Extensão do predicado custoTotalPres: IDP, T -> {V,F}
custoTotalPres(IDP,T) :- findall((Custo),cuidado(_,_,IDP,_,Custo),L),
                         somaLista(L,T).
```

```
% Extensão do predicado custoTotalData: Data, T -> {V,F}
custoTotalData(Data,T) :- findall((Custo),cuidado(Data,_,_,_,Custo),L),
                          somaLista(L,T).
```

No cálculo do custo total por especialidade surge uma particularidade. Contrariamente ao que acontece nos três predicados anteriores, não é possível ser obtida diretamente através do predicado `cuidado` a lista de todos os custos. Para se resolver isso gerou-se um predicado auxiliar `custoPrestador` que obtém o custo associado a um ID.

```
% Extensão do predicado custoTotalEsp: Esp, T -> {V,F}
custoTotalEsp(Esp,T) :- findall((IDP),prestador(IDP,_,Esp,_),L),
                        custoPrestador(L,X),
                        somaLista(X,T).

% Extensão do predicado presUtAux: IDP, R -> {V,F}
custoPrestador([],[]).
custoPrestador([IDP|T],R) :- findall((C),cuidado(,,IDP,_,C),L1),
                             concat(L1,L2,R),
                             custoPrestador(T,L2).
```

3.6. Predicados Extra

Adicionalmente aos predicados essenciais a este exercício prático, o grupo decidiu desenvolver alguns predicados extra, pensando na inclusão de novas funcionalidades de modo a pôr em prática todo o conhecimento adquirido e demonstrar destreza e capacidade em se ir além daquilo que inicialmente foi estabelecido ser feito.

Enumeram-se e apresentam-se os predicados complementarmente desenvolvidos:

1) **Listar todas as moradas presentes na Base de Conhecimento, sem repetições:**

- Com uma simples utilização do predicado `findall`, conseguimos alcançar todos os parâmetros de `Morada` que satisfazem o predicado `utente(_,_,_,Morada)`.
- De modo a se ter uma lista de conhecimento correta, elimina-se as repetições existentes na lista obtida acima, pelo uso do predicado `removeDuplicados`.

```
% Extensão do predicado moradas: R -> {V,F}
moradas(R) :- findall((Morada),utente(_,_,_,Morada),L),
              removeDuplicados(L,R).
```

2) **Determinar a lista de todos os custos totais dos Utentes:**

- Obtém-se uma lista de todos os ID's dos Utentes existentes na base de conhecimento, naturalmente, como até agora foi feito de forma exaustiva e repetitiva, pelo uso do `findall`.
- Com o predicado `custoListaUlt` geramos uma nova lista, mas com os tuplos (Utente, Custo) em que Utente é o ID do Utente em questão e Custo é a soma dos custos totais de todos os cuidados de saúde efetuados pelo mesmo.

```
% Extensão do predicado custoTodosUt: R -> {V,F}
custoTodosUt(R) :- findall((IDU), utente(IDU,_,_,_), L1),
                   custoListaUt(L1,R).

% Extensão do predicado custoListaUt: L, R -> {V,F}
custoListaUt([],[]).
custoListaUt([IDU|T],R) :- findall(Custo, cuidado(_,IDU,_,_,Custo), L1),
                           somaLista(L1,X),
                           custoListaUt(T,L2),
                           concat([X], L2, R).
```

3) **Determinar o top 5 de Utentes que mais gastaram:**

Este predicado é um aglomerar de recorrências a vários predicados auxiliares:

- Os primeiros passos são exatamente iguais aos do predicado extra anterior: gerar uma lista de tuplos (Utente, Custo) para cada um dos Utentes que tenha efetuado um cuidado de saúde;
- Por fim, e com o auxílio de todos os outros predicados, consegue-se gerar uma lista ordenada, mas desta vez com os tuplos invertidos (Custo, Utente) pela recorrência ao predicado `reversePares`. Esta lista sofre uma delimitação de apenas os primeiros 5 tuplos através do predicado `top` criado para o efeito.

```
% Extensão do predicado top5utentes: R -> {V,F}
top5utentes(R) :- findall(IDU, utente(IDU,_,_,_),L1),
                  custoListaUt(L1,L2),
                  concatPares(L2,L1,L3),
                  sort(L3,L4),
                  reversePares(L4,L5),
                  top(L5,5,R).
```

3.7. Predicados Auxiliares

Como se foi comprovando ao longo de todo o relatório, existiu várias vezes a necessidade de gerar predicados auxiliares de modo a se conseguir concretizar os vários predicados principais deste nosso problema:

1) ***somaLista*:** encarrega-se de fazer o somatório de todos os elementos presentes numa determinada lista;
2) ***concat:*** produz uma lista que nada mais é que a concatenação de duas listas;

3) ***concatPares:*** tem um papel semelhante ao predicado *concat,* diferenciando-se apenas no facto de concatenar duas listas em pares:
4) ***comprimento:*** calcula o comprimento de uma lista;
5) ***removeDuplicados:*** efetua a remoção de todos os elementos que se encontrem repetidos numa dada lista;
6) ***pertence:*** verifica se uma determinada informação pertence a uma lista;
7) ***n_pertence:*** faz exatamente o inverso da anterior;
8) ***reverse:*** trata de modificar uma lista, produzindo o seu inverso;
9) ***reversePares:*** tem um papel semelhante ao predicando *reverse,* mas em vez de receber uma lista, recebe uma lista de tuplos produzindo a inversa desta;
10) ***top:*** delimita uma determinada lista para um valor passado como parâmetro.

Estes predicados encontram-se anexados na secção 5 (Anexos) criada para o efeito, tal como procedido para a base de conhecimento inicial.

4. Conclusão e Sugestões

Após toda a contextualização e posterior resolução deste primeiro exercício de grupo é importante deixar claro todas as dúvidas e aprendizagens conseguidas com este trabalho prático. Sendo o primeiro exercício da Unidade Curricular, evidentemente que se gerou toda uma necessidade de tentar compreender como era suposto ir ao encontro daquilo que inicialmente foi proposto resolver.

Uma vez que o objetivo base deste exercício passou sempre por pôr em prática todo o conhecimento acerca da linguagem de programação em lógica PROLOG e o desenvolvimento de invariantes, a grande ajuda para se elaborar um relatório sólido e que fizesse jus ao esperado, esteve em aplicar os conhecimentos das aulas teóricas e práticas, seguindo o padrão da resolução de todos os exercícios realizados até ao momento.

Com toda esta ideia e planeamento definidos, o grupo foi capaz de, sem grandes dificuldades, não só resolver todo o exercício como ainda incluir alguns extras sempre com o intuito de instruir de maneira mais sólida e profunda o funcionamento da linguagem de programação lógica em questão e que problemas podem ser resolvidos e estudados com a mesma. A única dificuldade esteve apenas em tentar relacionar as várias bases de conhecimento aquando da necessidade de obtenção dos vários parâmetros pretendidos. Contudo, o feedback final acerca deste primeiro exercício de grupo é claramente positivo e motivador para a realização dos futuros exercícios práticos.

5. Anexos

```
%-------------------- Base de Conhecimento Inicial --------------------%

% Base de Conhecimento Utente (IDU,Nome,Idade,Cidade).
utente(6472,'Diogo Nogueira',20,'Fafe').
utente(1211,'Diogo Nogueira',21,'Porto').
utente(8374,'Tifany Silva',24,'Porto').
utente(3463,'Fabio Fontes',19,'Braga').
utente(2321,'Mariana Lino',20,'Guimaraes').
utente(9232,'Francisco Silva',21,'Braga').
utente(2372,'Cristina Lopes',22,'Fafe').
utente(7363,'Tiago Nogueira',18,'Porto').
utente(6252,'Sara Costa',18,'Porto').
utente(2373,'Joana Maria',28,'Braga').
utente(2324,'Joao Miguel',21,'Guimaraes').

% Base de Conhecimento Prestador (IDP,Nome,Especialidade,Instituicao).
prestador(4343,'Navarro Silva','Dermatologia','Hospital Privado de
Guimaraes').
prestador(3947,'Margarida Antunes','Dentista','Hospital Privado de
Guimaraes').
prestador(3643,'Nuno Costa','Nutricao','Classaude Porto').
prestador(7364,'Ana Maria','Hidroterapia','Clinica Fisiatrica de Fafe').
prestador(3642,'Tomas Casimiro','Clínica Geral','Classaude Porto').
prestador(8675,'Maria Joana','Psiquiatria','Clinica Praxis Porto').
prestador(3742,'Joana Silva','Psicologia','Clinica Praxis Porto').
prestador(7864,'Ana Luisa','Dermatologia','Clinica Fisiatrica de Fafe').
prestador(7324,'Mariana Costa','Dentista','Clinica Fisiatrica de Fafe').
prestador(3127,'Joao Araujo','Dentista','Hospital Privado de
Guimaraes').

% Base de Conhecimento Cuidado de Saúde (Data,IDU,IDP,Custo).
cuidado(date(2018,11,12),6472,4343,'Carie',12).
cuidado(date(2017,10,25),8374,7324,'Consulta de rotina aparelho',20).
cuidado(date(2016,09,15),3463,7864,'Queimadura na cara',15).
cuidado(date(2018,11,12),6472,3742,'Consulta',50).
cuidado(date(2015,01,25),9232,8675,'Consulta agendada/rotina',55).
cuidado(date(2017,08,30),2372,3642,'Paciente com febre',15).
cuidado(date(2018,04,21),7363,7364,'Tratamento rotineiro',25).
cuidado(date(2016,06,08),6252,3643,'Consulta de iniciacao de tratamento
dietetico',60).
cuidado(date(2017,12,24),2373,3947,'Remocao dos dentes do siso',55).
cuidado(date(2018,12,12),6472,4343,'Tratamento de eczema',60).
```

```
%---------------------- Predicados Auxiliares ------------------------%

% Extensão do predicado custoTotal: L, R -> {V,F}
somaLista([],0).
somaLista([H|T],R)  :- somaLista(T,X),
                       R is X+H.

% Extensão do predicado concat: L1, L2, L3 -> {V,F}
concat([],L,L).
concat([H|T],L2,[H|L]) :- concat(T,L2,L).

% Extensão do predicado concatPares: L1, L2, L3 -> {V,F}
concatPares([],[],[]).
concatPares([X|L1],[Y|L2],[(X,Y)|R]) :- concatPares(L1,L2,R).


% Extensão do predicado comprimento : L, N ->{V,F}
comprimento([],0).
comprimento([X|L],R) :- comprimento(L,N), R is 1+N.

% Extensão do predicado removeDuplicados : L, L1 ->{V,F}
removeDuplicados([],[]).
removeDuplicados([X|L],[H|R]) :- pertence(X,L),
removeDuplicados(L,[H|R]).
removeDuplicados([X|L],[X|R]) :- n_pertence(X,L),removeDuplicados(L,R).

% Extensão do predicado pertence: L, X -> {V,F}
pertence(X,[X|_]).
pertence(X,[Y|T]) :- X\=Y, %for backtracking purposes...
             pertence(X,T).

n_pertence(X,L):- \+ pertence(X,L). % "\+" é o mesmo que "not"

% Extensão do predicado reverse: L1, R -> {V,F}
reverse([],[]).
reverse([H|T],L) :- reverse(T,P), concat(P,[H],L).

% Extensão do predicado reversePares: L1, R -> {V,F}
reversePares([],[]).
reversePares([(X,Y)|T],L) :- reverse(T,P), concat(P,[(X,Y)],L).

% Extensão do predicado reverse: L1, N, R -> {V,F}
top(X,0,[]).
top([X|T],N,[X|R]) :- N1 is N - 1, top(T,N1,R).
```