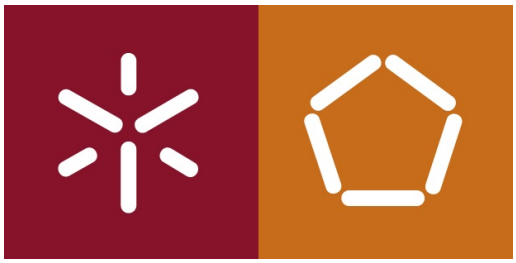

Universidade do Minho - Escola de Engenharia

Mestrado Integrado em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio

2017/2018 - 2ºsemestre

Exercício nº1 - Programação em Lógica e Invariantes

Autores :
Carlos Campos (A74745)

Diana Costa (A78985)

Marcos Pereira (A79116)

Vitor Castro (A77870)

Braga, 17 de Abril de 2018

Resumo

Perante a proposta de realizar um sistema de representação de conhecimento e raciocínio, com capacidade de caracterizar um universo de discurso na área da prestação de cuidados de saúde, houve um impasse inicial devido à necessidade de uma boa estruturação do problema. Tudo isto requeriu a escolha acertada de predicados e bom uso da linguagem de programação em lógica, *PROLOG*, de modo a que a resolução destes fosse a mais clara e simples possível.

Depois de algum tempo e trabalho, o resultado encontrado foi satisfatório, e os objetivos e respostas às questões do enunciado proposto cumpridos.

Conteúdo

1 Introdução **4**

2 Preliminares **5**

3 Descrição do trabalho e Análise de resultados **6**
3.1 Representação de Conhecimento 6
3.1.1 Utente 6
3.1.2 Prestador 6
3.1.3 Cuidado 7
3.1.4 Recibo 7
3.2 Registar Utentes, Prestadores, Cuidados de Saúde e Recibo 7
3.2.1 Invariantes 8
3.2.2 Definições iniciais 9
3.3 Remover Utentes, Prestadores, Cuidados de Saúde e Recibo 9
3.3.1 Invariantes 10
3.4 Identificar utentes por critérios de seleção 11
3.4.1 Utentes por Nome 11
3.4.2 Utentes por Idade 11
3.4.3 Utentes por Morada 11
3.5 Instituições prestadoras de cuidados de saúde 12
3.6 Cuidados prestados por instituição/cidade/datas 12
3.6.1 Instituição 12
3.6.2 Cidade 13
3.6.3 Datas 13
3.7 Identificação de utentes de um prestador/especialidade/instituição 13
3.7.1 Utentes de um Prestador 13
3.7.2 Utentes de uma Especialidade 14
3.7.3 Utente de uma Instituição 14
3.8 Cuidados de saúde realizados por utente/instituição/prestador 15
3.8.1 Utente 15
3.8.2 Instituição 15
3.8.3 Prestador 15
3.9 Instituições/prestadores a que um utente já recorreu 16
3.10 Cálculo do custo total dos cuidados de saúde por utente, especialidade, prestador e datas 16
3.10.1 Utente 16
3.10.2 Especialidade 17
3.10.3 Prestador 17
3.10.4 Datas 18
3.11 Predicados sobre recibo 18
3.11.1 Utentes que gastaram mais do que um determinado valor 18
3.11.2 Utentes mais frequentes 19
3.11.3 Gasto total registado de um utente 19
3.12 Outros predicados 20
3.12.1 Remover duplicados de uma lista 20
3.12.2 Número de vezes que um elemento aparece numa lista 20
3.12.3 Ordenar por ordem decrescente 20
3.12.4 Soluções 21
3.12.5 Comprimento 21
3.12.6 Testar 21
3.12.7 Somatório 21
3.12.8 Pertence 22

3.12.9 Predicado não 22
3.12.10 Insere ordenado 22
3.12.11 Concatena 22
3.13 Análise de Resultados 23

4 Conclusões e Sugestões **25**

1 Introdução

Este projeto surge no âmbito da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio, do Mestrado Integrado em Engenharia Informática da Universidade do Minho. Era requerido que, como primeiro exercício, fosse desenvolvido um sistema de representação de conhecimento e raciocínio com capacidade para caracterizar um universo de discurso na área da prestação de cuidados de saúde.

Como primeira instância, foi fornecida a seguinte caracterização de conhecimento,

- utente: #IdUt, Nome, Idade, Morada $\rightsquigarrow \{\mathbb{V}, \mathbb{F}\}$
- prestador: #IdPrest, Nome, Especialidade, Instituição $\rightsquigarrow \{\mathbb{V}, \mathbb{F}\}$
- cuidado: Data, #IdUt, #IdPrest, Descrição, Custo $\rightsquigarrow \{\mathbb{V}, \mathbb{F}\}$

à qual o grupo adicionou:

- recibo: #IdUt, NomeUt, #IdPrest, Especialidade, Data, Custo $\rightsquigarrow \{\mathbb{V}, \mathbb{F}\}$

Como segunda tarefa, era requerido que se definissem determinadas funcionalidades, como registar utentes, prestadores e cuidados de saúde, remover os mesmos, identificar utentes por critérios de seleção, calcular o custo todal dos cuidados de saúde por utente, especialidade, prestador ou datas, entre outros. A estes adicionaram-se mais algumas funcionalidades referentes ao "recibo", de forma a que o sistema representasse com mais pormenor características relativas a cuidados de saúde.

Em suma, a Secção 2 descreve os preliminares necessários ao projeto, enquanto que a Secção 3 descreve a resolução do trabalho, desde a representação do conhecimento(3.1), resolução das funcionalidades(3.2 a 3.11), e com a consequente análise de resultados(3.13) e resumo geral de predicados que foram usados com mais frequência ao longo do exercício(Secção 3.12).

O relatório termina com uma breve conclusão na Secção 4, onde é feito um balanço do trabalho realizado, tendo em conta as dificuldades ao longo do desenvolvimento do mesmo.

2 Preliminares

Para o desenvolvimento deste exercício prático não foi necessário o estudo de matérias adicionais, apenas da matéria lecionada, até então, na presente unidade curricular.

A leitura e compreensão deste relatório são facilitadas através de conhecimentos de programação em lógica com invariantes, consciência da sintaxe de *Prolog* e entendimento para representar conhecimento e construir mecanismos de raciocínio para dar solução aos problemas propostos.

3 Descrição do trabalho e Análise de resultados

3.1 Representação de Conhecimento

Nos seguintes tópicos são apresentados os predicados exigidos à representação do conhecimento pedida.

3.1.1 Utente

Um utente é caracterizado pelo seu ***IdUt*** (que é diferente para cada utente, identificando-o), ***Nome***, ***Idade*** e ***Morada***.

```
%-----------------------------------------------------------------------
% Extensao do predicado utente: IdUt,Nome,Idade,Morada -> {V,F}

utente(1,carlos,21,famalicao).
utente(2,diana,20,trofa).
utente(3,marcos,20,anais).
utente(4,vitor,20,guimaraes).
utente(5,antonio,6,povoadevarzim).
utente(6,rita,26,pontedelima).
utente(7,marta,66,guimaraes).
utente(8,paulo,16,barcelos).
utente(9,maria,43,barcelos).
utente(10,luis,51,vizela).
```

Figura 1: Predicado utente

3.1.2 Prestador

Um prestador é caracterizado pelo seu ***IdPrest*** (que é diferente para cada prestador, identificando-o), ***Nome***, ***Especialidade*** e ***Instituicao***.

```
%-------------------------------------------------------------------------------
% Extensao do predicado prestador: IdPrest,Nome,Especialidade,Instituicao -> {V,F}

prestador(1, pres1, pediatria, csjoane).
prestador(2, pres2, cardiologia, hospitalbraga).
prestador(3, pres3, cirurgia, hospitalbraga).
prestador(4, pres4, ginecologia, hospitalbraga).
prestador(5, pres5, neurologia, hsmm).
prestador(6, pres6, psiquiatria, hsog).
prestador(7, pres7, oftamologia, htrofa).
prestador(8, pres8, reumatologia, hospitalbraga).
prestador(9, pres9, psiquiatria, hospitalbraga).
```

Figura 2: Predicado prestador

3.1.3 Cuidado

Um cuidado é caracterizado por ***Data***, ***IdUt***, ***IdPrest***, ***Descricao*** e ***Custo***.

```
%-----------------------------------------------------------------------------
% Extensao do predicado cuidado: Data,IdUt,IdPrest,Descricao,Custo -> {V,F}

cuidado(01-02-2017, 1, 6, hipnose, 15).
cuidado(13-02-2017, 3, 4, papanico, 30).
cuidado(13-02-2017, 2, 5, cerebrotroca, 30).
cuidado(14-02-2017, 2, 7, olhoterapia, 7).
cuidado(20-03-2017, 4, 2, pacemaker, 20).
cuidado(02-04-2017, 7, 4, ovariologia, 5).
cuidado(03-04-2017, 3, 5, neuroterapia, 25).
cuidado(20-04-2017, 6, 7, retinoterapia, 35).
cuidado(22-05-2017, 6, 2, cardioterapia, 55).
cuidado(04-06-2017, 2, 2, cardiograma, 99).
cuidado(15-06-2017, 1, 1, terapiafala, 5).
cuidado(18-06-2017, 2, 8, reumatomagrafia, 350).
cuidado(18-06-2017, 2, 9, cbt, 10).
```

Figura 3: Predicado cuidado

3.1.4 Recibo

O recibo é informação adicional em relação à pedida no exercício. Este guarda ***IdRecibo, IdUt, NomeUt, Morada, Especialidade, Instituicao, Data, Descricao, Custo*** em relação a determinado cuidado.

```
%-----------------------------------------------------------------------------------------------
% Extensao do predicado recibo: IdRecibo, IdUt, NomeUt, Morada, Especialidade, Instituicao, Data, Descricao, Custo -> {V,F}

recibo(1, 3, marcos, anais, ginecologia, hospitalbraga, 13-02-2017, papanico, 30).
recibo(2, 4, vitor, guimaraes, cardiologia, hospitalbraga, 20-03-2017, pacemaker, 20).
```

Figura 4: Predicado recibo

3.2 Registar Utentes, Prestadores, Cuidados de Saúde e Recibo

Pedida a implementação de um predicado que permitisse a inserção de utentes, prestadores e cuidados, o grupo decidiu criar uma predicado ***inserir*** que funcionaria para qualquer um destes termos. A inserção seria verificada através dos invariantes definidos, que se apresentam de seguida. Foi criado um predicado ***insercao*** que trata de fazer o ***assert*** e o ***retract***, quando necessários. Por fim, o predicado ***inserir*** invoca ***teste*** que testa a integridade da informação a adicionar.

```
% -----------------------------------------------------
% Extensão do predicado inserir: Termo -> {V,F}
% mesmo que EVOLUCAO

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

Figura 5: Predicado inserir

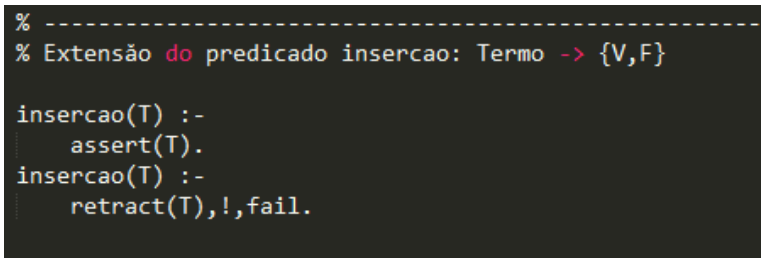

Figura 6: Predicado insercao

3.2.1 Invariantes

Para que a inserção seja íntegra e não repetida, é necessário estabelecer invariantes para utente, prestador, cuidado e recibo.

No caso do utente, não se deve permitir a inserção de conhecimento de um utente com um id já existente.

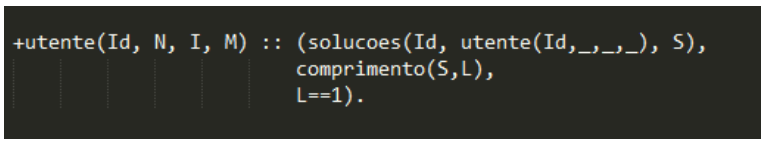

Figura 7: Invariante adicionar utente

Para o prestador, não se deve permitir a inserção de conhecimento de um prestador com um id já existente.

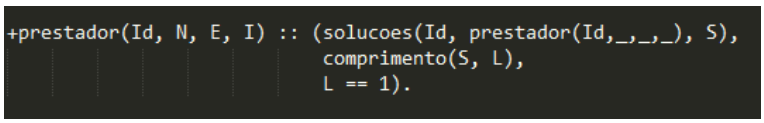

Figura 8: Invariante adicionar prestador

No caso do cuidado, não se deve permitir a inserção de conhecimento de um cuidado quando o id do utente e prestador não existem na base de conhecimento.

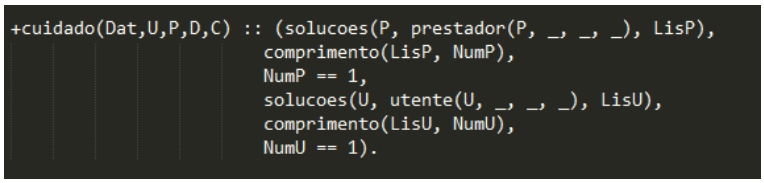

Figura 9: Invariante adicionar cuidado

Por fim, para o recibo, não se deve permitir a inserção de conhecimento de um recibo quando o id do mesmo já está presente. Deve ser verificada a existência do utente, de um prestador e do cuidado respetivo, face à base de conhecimento.

```
%-------------------------------------------------------------------------------------
% Invariante que não permite a inserção de conhecimento de um recibo com um id já existente
% uso _ quando o dado não é importante e não quero saber dele

+recibo(Id, U, N, M, E, I, Dat, D, C) ::
                        (solucoes(Id, recibo(Id,_,_,_,_,_,_,_,_), R),
                        comprimento(R,Lr),
                        Lr==1,
                        solucoes((U,N,M), utente(U,N,_,M), Uts),
                        comprimento(Uts,Lu),
                        Lu==1,
                        solucoes((P,E,I), prestador(P,_,E,I), Pres),
                        comprimento(Pres, Lp),
                        Lp>=1,
                        solucoes((Dat,U,P,D,C), cuidado(Dat,U,P,D,C), Cuids),
                        comprimento(Cuids, Lc),
                        Lc==1).
```

Figura 10: Invariante adicionar recibo

3.2.2 Definições iniciais

Naturalmente, para que os invariantes funcionem e seja de facto alterada a base de conhecimento, é necessário fazer as definições iniciais.

```
%-----------------------------------------
% SICStus PROLOG: definicoes iniciais

:- op(900,xfy,'::').

% Para os invariantes:

:- dynamic utente/4.
:- dynamic prestador/4.
:- dynamic cuidado/5.
:- dynamic recibo/9.
```

Figura 11: Definições iniciais

3.3 Remover Utentes, Prestadores, Cuidados de Saúde e Recibo

Do mesmo modo que a inserção, a remoção necessitou dos respetivos predicados ***remover*** e ***remocao***. O predicado ***remover*** verifica a existência da lista de termos a remover, testando-a e invocando o predicado ***remover***. O teste deve ser feito primeiro que a remoção pois, caso não fosse, iria ser feito um teste sobre um predicado que entretanto havia sido removido. Caso isto acontecesse, o predicado ***remover*** iria fazer o ***assert*** da informação que tinha retirado, erradamente.

```
% -----------------------------------------------------------
% Extensão do predicado remover: Termo -> {V,F}

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

Figura 12: Predicado remover

```
% ------------------------------------------------
% Extensão do predicado remocao: Termo -> {V,F}

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

Figura 13: Predicado remocao

3.3.1 Invariantes

Para que a remoção seja íntegra e correta, é necessário estabelecer invariantes para utente, prestador e cuidado. Para os recibos, deve ser garantido que nunca é possível retirar um, tal como explicitado no invariante.

No caso do utente, não se deve permitir a remoção de conhecimento de um utente não presente na base de conhecimento ou com id associado a cuidado.

```
-utente(Id, N, I, M) :: (solucoes(Id, utente(Id,N,I,M), Uts),
                         comprimento(Uts, Lu),
                         Lu==1,
                         solucoes(Id, cuidado(_,Id,_,_,_), Cuids),
                         comprimento(Cuids, Lc),
                         Lc == 0).
```

Figura 14: Invariante retirar utente

Já no prestador, não se deve permitir a remoção de conhecimento de um prestador não presente na base de conhecimento ou com id associado a cuidado.

```
-prestador(Id, N, E, I) :: (solucoes(Id, prestador(Id,_,_,_), Prests),
                            comprimento(Prests, Lp),
                            Lp == 1,
                            solucoes(Id, cuidado(_,_,Id,_,_), Cuids),
                            comprimento(Cuids, Lc),
                            Lc == 0).
```

Figura 15: Invariante retirar prestador

Para o cuidado, não deve ser permitida a remoção de conhecimento de um cuidado não presente na base de conhecimento.

```
-cuidado(Dat, U, P, D, C) :: (solucoes((Dat,U,P,D,C), cuidado(Dat,U,P,D,C), Cuids),
                              comprimento(Cuids, L),
                              L == 1).
```

Figura 16: Invariante retirar cuidado

No caso do recibo, não deve ser permitida a remoção de conhecimento. De facto numa situação real nunca se poderia anular um recibo, sob o risco de cometer uma ilegalidade.

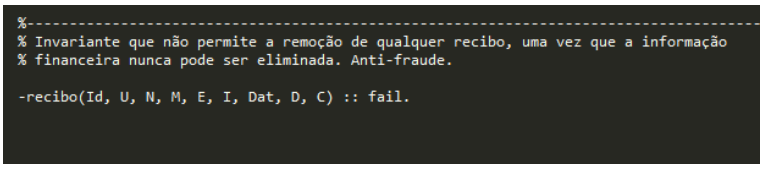

Figura 17: Invariante retirar recibo

3.4 Identificar utentes por critérios de seleção

Neste ponto é proposta a identificação de utentes por diversos critérios de seleção. No caso, especificamente, é possível identificar utentes pelo seu nome, idade e morada.

3.4.1 Utentes por Nome

Neste predicado é fornecido o Nome, sendo procuradas todas as soluções existentes, guardando-as em Lis. A lista resultante será composta pela descrição completa de todos os utentes com o nome a procurar.

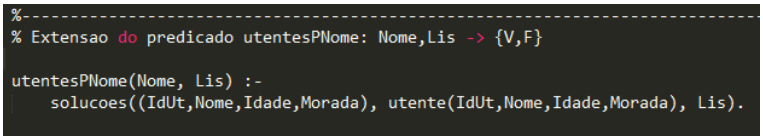

Figura 18: Predicado utentesPNome

3.4.2 Utentes por Idade

Neste predicado é fornecida a Idade, sendo procuradas todas as soluções existentes, guardando-as em Lis. A lista resultante será composta pela descrição completa de todos os utentes com a idade a procurar.

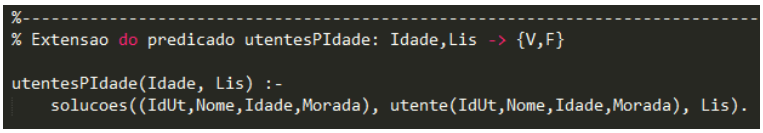

Figura 19: Predicado utentesPIdade

3.4.3 Utentes por Morada

Neste predicado é fornecida a Morada, sendo procuradas todas as soluções existentes, guardando-as em Lis. A lista resultante será composta pela descrição completa de todos os utentes com a morada a procurar.

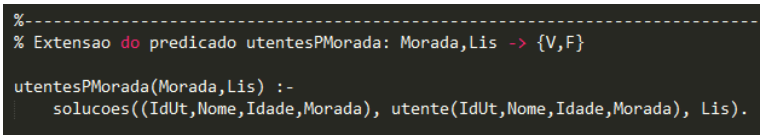

Figura 20: Predicado utentesPMorada

3.5 Instituições prestadoras de cuidados de saúde

Neste ponto é preciso identificar todas as instituições que prestam cuidados de saúde. Elas estão registadas no predicado ***prestador***. Usando o ***solucoes*** é fácil obter a lista de instituições existentes. Naturalmente, e como há repetição destas instituições, a lista obtida passará pelo filtro ***removeDup*** (especificado no fim deste documento).

```
% -------------------------------------------------------------
% Extensao do predicado instituicoes: Resultado -> {V,F}

instituicoes(Resultado) :-
    solucoes(Instituicao, prestador(_,_,_,Instituicao), Insts),
    removeDup(Insts,Resultado).
```

Figura 21: Predicado instituicoes

3.6 Cuidados prestados por instituição/cidade/datas

Neste tópico, é caracterizada a identificação de cuidados por diversos critérios, entre eles, por cuidados por instituição, cidade e por data.

3.6.1 Instituição

O predicado "cuidadosPInstituicao"fornece todos os cuidados que uma instituição presta. Assim, inserindo a instituição pretendida, este predicado começará por construir uma lista "L"com todos os ID's de prestadores, uma vez que apenas através destes se consegue chegar aos cuidados (argumento comum entre predicados "prestador"e "cuidado"). Depois de obtida esta lista, será invocado outro predicado - "cuidadosPInstituicaoAux- que irá recolher todos os cuidados fornecidos por um determinado prestador(predicado "cuidadosDePrestador"), e fará o mesmo para os restantes.

```
% Extensao do predicado cuidadosPInstituicao: Inst,Result -> {V,F}

cuidadosPInstituicao(Inst,List) :-
    solucoes(IdPrest,prestador(IdPrest,Nome,Esp,Inst),L),
    cuidadosPInstituicaoAux(L,List).

% -------------------------------------------------------------------------------------------------
% Extensao do predicado cuidadosPInstituicaoAux: L,Result -> {V,F}

cuidadosPInstituicaoAux([],[]).
cuidadosPInstituicaoAux([H|T],List) :-
    cuidadosDePrestador(H,Aux1),
    cuidadosPInstituicaoAux(T,Aux2),
    concatena(Aux1,Aux2,List).

% -------------------------------------------------------------------------------------------------
% Extensao do predicado cuidadosDePrestador: IdPrest,R -> {V,F}

cuidadosDePrestador(IdPrest,R):-
    solucoes((Data,IdUt,IdPrest,Desc,Custo),cuidado(Data,IdUt,IdPrest,Desc,Custo),R).
```

Figura 22: Predicados "cuidadosPInstituicao", "cuidadosPInstituicaoAux"e "cuidadosDePrestador"

3.6.2 Cidade

O predicado "cuidadosPCidade"fornece todos os cuidados prestados numa cidade. Deste modo, inserindo a cidade pretendida, este predicado começará por construir uma lista "Uts"com todos os ID's de utentes, uma vez que apenas através destes se consegue chegar aos cuidados (argumento comum entre predicar "utente"e "cuidado"). Depois de obtida esta lista, será invocado outro predicado - "getCuidadosPCidadeAux- que irá recolher todos os cuidados fornecidos por uma determinada instituiçao, através da descrição. Isto é conseguido pela verificação "cuidado(_,IdUtente,_,Descricao,_)", que determina se existem cuidados para determinado utente, adicionando a descrição do mesmo à lista.

```
% Extensao do predicado cuidadosPCidade: Cidade,Result -> {V,F}

cuidadosPCidade(Cidade,List) :-
    solucoes(U, utente(U,_,_,Cidade), Uts),
    getCuidadosPCidadeAux(Uts,ListCuid),
    removeDup(ListCuid, List).

% Extensao do predicado getCuidadosPCidadeAux: L,Resultado -> {V,F}

% Recebe lista de IDs de utentes
% Obtém lista de instituições desses utentes
getCuidadosPCidadeAux([], []).
getCuidadosPCidadeAux([IdUtente | T], [Descricao | Resto]) :-
    cuidado(_, IdUtente, _, Descricao, _),
    getCuidadosPCidadeAux(T, Resto).
```

Figura 23: Predicados "cuidadosPCidade"e "getCuidadosPCidadeAux"

3.6.3 Datas

Neste predicado é fornecida a data pretendida, sendo procuradas todas as soluções existentes e guardando-as em "List". A lista resultante será composta pela descrição completa de todos os cuidados com a data a procurar.

```
% Extensao do predicado cuidadosPDatas: Data,Result -> {V,F}

cuidadosPDatas(Data,List) :-
    solucoes((Data,Ut,Prest,Desc,Cust), cuidado(Data,Ut,Prest,Desc,Cust), List).
```

Figura 24: Predicados "cuidadosPDatas"

3.7 Identificação de utentes de um prestador/especialidade/instituição

Aqui é pedido a identificação de utentes ao cargo de um dado prestador, que estejam a ser tratados segunda uma especialidade, e que foram cuidados numa instituição específica.

3.7.1 Utentes de um Prestador

Neste tópico é dado um identificador de prestador, que é usado no cruzamento com todos os prestadores com o mesmo id, através do predicado solucoes. O resultado, que é uma lista de id's de prestadores, é usada no predicado utentesDePrestadorAux. Este por sua vez, faz a interceção de todos esses id's de prestador com os cuidados que possuem um igual, neste predicado também é usado o concatena, para juntar ao resultado, a lista de id's de utentes que é gerada através do cruzamento com a cauda de id's de prestadores. De seguida, é usado o predicado removeDups, que serve para remover os id's de Utentes repetidos, quando por exemplo existem 2 cuidados iguais, em que a sua diferença é apenas a data. Por ultimo, o resultado final é gerado com a ajuda do predicado utentesPListaId, que usa uma lista de identificadores de utentes, e dá uma lista de cuidados, que tenham esses id's, o seu funcionamento é idêntico ao predicado utentesDePrestadorAux.

```
utentesDePrestador(IdPrest,Lis):-
    solucoes(IdPrest,prestador(IdPrest,Nome,Esp,Inst),Aux1),
    utentesDePrestadorAux(Aux1,Aux2),
    utentesPListaId(Aux2,Lis).

utentesDePrestadorAux([],[]).
utentesDePrestadorAux([H|T],Lis):-
    solucoes(IdUt,cuidado(_,IdUt,H,_,_),Aux1),
    utentesDePrestadorAux(T,Aux2),
    concatena(Aux1,Aux2,Aux3),
    removeDup(Aux3,Lis).

utentesPListaId([],[]).
utentesPListaId([H|T],Lis):-
    utentesPId(H,Aux1),
    utentesPListaId(T,Aux2),
    concatena(Aux1,Aux2,Lis).
```

Figura 25: Predicado utentesDePrestador

3.7.2 Utentes de uma Especialidade

O predicado utentesDeEspecialidade funciona de igual forma ao predicado utentesDePrestador, com a diferença de que é dada a especialdiade e não o identificador de um prestador, e assim o resultado vai ter todos os utentes que foram tratados pela mesma especialidade, mas por prestadores diferentes.

```
utentesDeEspecialidade(Esp,Lis):-
    solucoes(IdPrest,prestador(IdPrest,Nome,Esp,Inst),Aux1),
    utentesDePrestadorAux(Aux1,Aux2),
    utentesPListaId(Aux2,Lis).
```

Figura 26: Predicado utentesDeEspecialidade

3.7.3 Utente de uma Instituição

Para concluir esta parte do enunciado é dado um nome de uma instituição ao predicado utentesDeEspecialidade que executa de maneira idêntica ao predicado utentesDePrestador, e assim o resultado vai ter todos os utentes que foram tratados na mesma instituição, mas por prestadores diferentes.

```
utentesDeInstituicao(Inst,Lis):-
    solucoes(IdPrest,prestador(IdPrest,Nome,Esp,Inst),Aux1),
    utentesDePrestadorAux(Aux1,Aux2),
    utentesPListaId(Aux2,Lis).
```

Figura 27: Predicado utentesDeInstituicao

3.8 Cuidados de saúde realizados por utente/instituição/prestador

Neste tópico é requerido a identificação de cuidados de saúde de um dado utente, que se realizaram numa instituição específica e que foram realizados por um certo prestador.

3.8.1 Utente

Este predicado fornece a lista de cuidados que foram dados a um determinado utente, através do predicado solucoes, que faz a interceção com o Id fornecido.

```
cuidadosDeUtentes(IdUt,R):-
    solucoes((Data,IdUt,IdPrest,Desc,Custo),cuidado(Data,IdUt,IdPrest,Desc,Custo),R).
```

Figura 28: Predicado cuidadosDeUtentes

3.8.2 Instituição

Este predicado fornece a lista de cuidados que foram feitos numa dada instituição. Primeiro, é feita a seleção dos prestadores, que trabalham nessa mesma instituição, através do predicado solucoes, sendo que, depois, o predicado cuidadosDeInstituicaoAux pega na lista de identificadores de prestadores resultantes, e interceta-a com todos os cuidados que possuam esses id's, com o auxílio do predicado cuidadosDePrestador(explicado mais à frente). Por ultimo, as respostas tanto do cuidadosDePrestador como do cuidadosDeInstituicaoAux aplicado à cauda do argumento inicial, são concatenadas.

```
cuidadosDeInstituicao(Inst,Lis):-
    solucoes(IdPrest,prestador(IdPrest,Nome,Esp,Inst),Aux),
    cuidadosDeInstituicaoAux(Aux,Lis).

cuidadosDeInstituicaoAux([],[]).
cuidadosDeInstituicaoAux([H|T],Lis):-
    cuidadosDePrestador(H,Aux1),
    cuidadosDeInstituicaoAux(T,Aux2),
    concatena(Aux1,Aux2,Lis).
```

Figura 29: Predicado cuidadosDeInstituicao

3.8.3 Prestador

Este predicado fornece a lista de cuidados que foram efetuados por um prestador específico, através do predicado solucoes, que faz a interceção com o Id fornecido.

```
cuidadosDePrestador(IdPrest,R):-
    solucoes((Data,IdUt,IdPrest,Desc,Custo),cuidado(Data,IdUt,IdPrest,Desc,Custo),R).
```

Figura 30: Predicado cuidadosDePrestador

3.9 Instituições/prestadores a que um utente já recorreu

Neste ponto pretende-se obter os pares prestador/instituição a partir do identificador de um utente. Para isso, foi primeiro necessário definir um predicado ***getInstPrestByPrestadores*** que obtenha os pares prestador/instituição a partir de uma lista de prestadores:

```
% Extensao do predicado getInstPrestByPrestadores: IdPrestadores, Resultado -> {V,F}
getInstPrestByPrestadores([], []).
getInstPrestByPrestadores([IdPrestador | T], [(Instituicao, IdPrestador) | Resto]) :-
  prestador(IdPrestador, _, _, Instituicao),
  getInstPrestByPrestadores(T, Resto).
```

Figura 31: Predicado getInstPrestByPrestadores

Este predicado funciona de uma maneira simples, obtendo a instituição de cada prestador, um a um, até ter a lista completa.

De seguida, foi definido o predicado ***getInstPrestByUtente*** que, fazendo uso do código mostrado acima e do "filtro"***removeDup***, obtém uma lista de pares instituição/prestador a que um utente já recorreu, a partir do ID desse utente:

```
% Extensao do predicado getInstPrestByUtente: IdUtente, Resultado -> {V,F}
getInstPrestByUtente(IdUtente, R) :-
  solucoes(IdPrestador, cuidado(_, IdUtente, IdPrestador, _, _), S),
  removeDup(S, Resultado),
  getInstPrestByUtenteAux(Resultado, R).
```

Figura 32: Predicado getInstPrestByUtente

3.10 Cálculo do custo total dos cuidados de saúde por utente, especialidade, prestador e datas

3.10.1 Utente

Para calcular o custo total dos cuidados por utente, foi definido o predicado ***getTotalByUtente***:

```
% Extensao do predicado getTotalByUtente: IdUtente, Total -> {V,F}
getTotalByUtente(IdUtente, T) :-
  solucoes(Custo, cuidado(_, IdUtente, _, _, Custo), S),
  somatorio(S, T).
```

Figura 33: Predicado getTotalByUtente

Este funciona de maneira simples, primeiro obtendo uma lista de custos relacionados a um utente, e de seguida calculando a sua soma, fazendo recurso ao predicado ***somatorio***.

3.10.2 Especialidade

Para calcular o custo total dos cuidados por especialidade, foi definido o predicado ***getTotalByEspecialide***:

```
% Extensao do predicado getTotalByEspecialidade: Especialidade, Total -> {V,F}
getTotalByEspecialidade(Especialidade, T) :-
  solucoes(IdPrest, prestador(IdPrest, _, Especialidade, _), Prestadores),
  getTotalByPrestadores(Prestadores, T).
```

Figura 34: Predicado getTotalByEspecialidade

Este obtém a lista de prestadores associados a uma especialidade, e de seguida usa o predicado ***getTotalByPrestadores*** para obter o custo total.

O predicado ***getTotalByPrestadores*** foi definido da seguinte maneira:

```
% Extensao do predicado getTotalByPrestadores: IdPrestador[], Total -> {V,F}
getTotalByPrestadores([], 0).
getTotalByPrestadores([IdPrestador | L], T) :-
  getTotalByPrestador(IdPrestador, Custo),
  getTotalByPrestadores(L, RestoDoTotal),
  T is RestoDoTotal + Custo.
```

Figura 35: Predicado getTotalByPrestadores

Este faz uso do predicado ***getTotalByPrestador***, que também foi necessário implementar e é explicado abaixo.

3.10.3 Prestador

O cálculo do custo total por prestador foi definido pelo seguinte predicado ***getTotalByPrestador***:

```
% Extensao do predicado getTotalByPrestador: IdPrestador, Total -> {V,F}
getTotalByPrestador(IdPrestador, T) :-
  solucoes(Custo, cuidado(_, _, IdPrestador, _, Custo), S),
  somatorio(S, T).
```

Figura 36: Predicado getTotalByPrestador

Este predicado, de maneira semelhante aos anteriores, obtém a lista de custos relacionados a um prestador, e de seguida calcula a sua soma com recurso ao predicado ***somatorio***.

3.10.4 Datas

Para a criação do predicado ***getTotalByDatas***, foi primeiro necessário definir o predicado ***getTotalByData***:

```
% Extensao do predicado getTotalByData: Data, Total -> {V,F}
getTotalByData(Data, T) :-
  solucoes(Custo, cuidado(Data, _, _, _, Custo), S),
  somatorio(S, T).
```

Figura 37: Predicado getTotalByData

Este predicado funciona de uma maneira simples e semelhante aos anteriores, fazendo uso de ***somatorio***.

Uma vez este definido, passamos a definir o predicado ***getTotalByDatas***:

```
% Extensao do predicado getTotalByDatas: Data[], Total -> {V,F}
getTotalByDatas([], 0).
getTotalByDatas([Data | L], T) :-
  getTotalByData(Data, Custo),
  getTotalByDatas(L, RestoDoTotal),
  T is RestoDoTotal + Custo.
```

Figura 38: Predicado getTotalByDatas

Por sua vez, este predicado calcula o custo total a partir de uma lista de datas, usando ***getTotalByData*** em cada elemento da lista e acrescentado esse valor ao resultado final.

3.11 Predicados sobre recibo

Como solicitado no enunciado deste exercício, o grupo definiu alguns predicados extraordinários, os quais são apresentados de seguida. Todos eles tiram partido do predicado ***recibo***, também extra ao enunciado.

3.11.1 Utentes que gastaram mais do que um determinado valor

Com este predicado pretende-se saber quais os utentes que, em um qualquer recibo, tenham gasto mais que determinado valor. Foram desenvolvidos os predicados ***utentesQGastaramMaisQX*** e ***utentesQGastaramMaisQXAux***.

O primeiro encontra todas as relações Utente-Custo, enviando para o segundo essas relações, que são filtradas, ficando apenas as que têm valor superior ao definido.

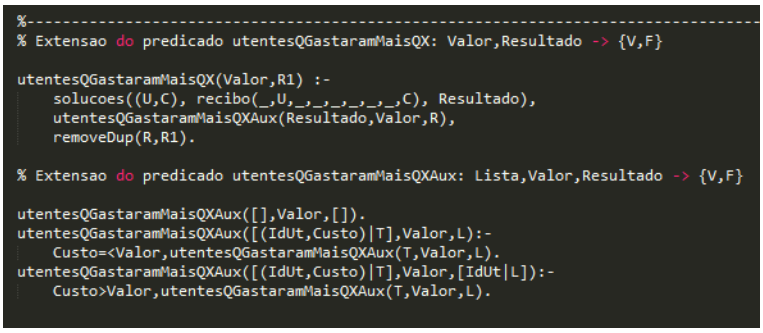

Figura 39: Predicado utentesQGastaramMaisQX

3.11.2 Utentes mais frequentes

Com este predicado pretende-se saber quais os utentes que mais registaram recibos. Foram desenvolvidos os predicados ***listarUtentesMaisFreq***, ***listarUtentesMaisFreqAux***, ***quantosTem*** e ***ordenarDecresc***.

O primeiro lista todos os utentes existentes e todos os utentes que obtiveram recibos. Esta informação é passada ao segundo predicado que, invocando o terceiro, obtém a listagem Utente-#Recibos. Feito isto, a lista é ordenada por ordem decrescente pelo quarto predicado.

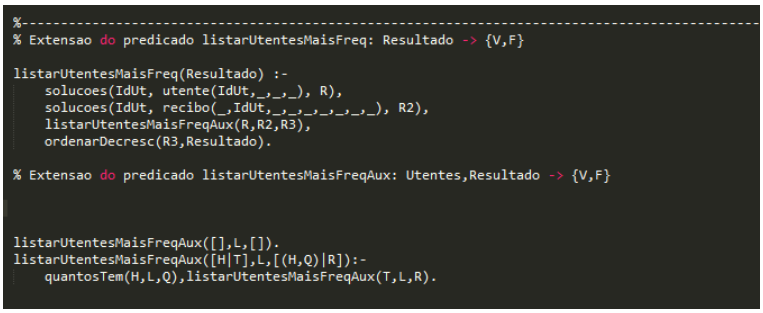

Figura 40: Predicado listarUtentesMaisFreq

3.11.3 Gasto total registado de um utente

Por fim, foi desenvolvido o predicado que obtém o gasto que um determinado utente registou, através dos recibos. Foram desenvolvidos os predicados ***gastoRegistadoPorUtente*** e ***somatorio***.

A primeira recolhe todos os custos associados a um determinado utente, passando-a à segunda, responsável por somar o conjunto recebido.

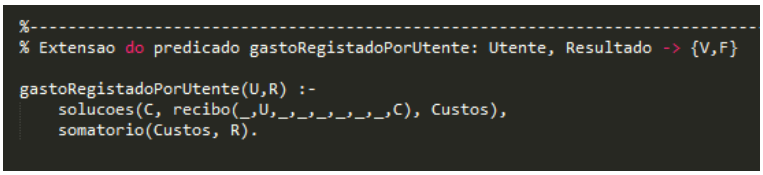

Figura 41: Predicado gastoRegistadoPorUtente

3.12 Outros predicados

Nesta secção resumem-se os predicados base que foram usados como auxílio de outros ao longo do relatório.

3.12.1 Remover duplicados de uma lista

Predicado que, ao receber uma lista, verifica a existência de elementos repetidos. Em caso afirmativo, os mesmos são removidos, e no final dará origem a uma outra lista sem elementos repetidos.

```
% Extensao do predicado removeDup: L,R -> {V,F}

removeDup([],[]).
removeDup([X|T],R):-
    pertence(X,T),
    removeDup(T,R).
removeDup([X|T],[X|R]):-
    nao(pertence(X,T)),
    removeDup(T,R).
```

Figura 42: Predicado removeDup

3.12.2 Número de vezes que um elemento aparece numa lista

Neste predicado é calculado o número de vezes que um determinado elemento aparece numa lista.

```
% Extensao do predicado quantosTem:A,Lista,Resultado  -> {V,F}

quantosTem(A,[],0).
quantosTem(A,[H|T],Resultado):-
    (A == H), quantosTem(A,T,R), Resultado is R+1.
quantosTem(A,[H|T],Resultado):-
    (A \= H), quantosTem(A,T,Resultado).
```

Figura 43: Predicado quantosTem

3.12.3 Ordenar por ordem decrescente

Predicado que, ao receber uma lista, ordena os seus elementos decrescentemente numa outra lista resultado.

```
% Extensao do predicado ordenarDecresc: L,Resultado -> {V,F}

ordenarDecresc([X],[X]).
ordenarDecresc([X|Y],T):-
    ordenarDecresc(Y,R),
    insereOrdenado(X,R,T).
```

Figura 44: Predicado ordenarDecresc

3.12.4 Soluções

"soluções"é responsável por determinar todas as possibilidades de prova de um teorema, e resulta em verdadeiro quando o predicado "findAll"também o for.

```
% -------------------------------------------------------------
% Extensão do predicado solucoes: X,Y,Z -> {V,F}

solucoes(X,Y,Z) :-
    findall(X,Y,Z).
```

Figura 45: Predicado solucoes

3.12.5 Comprimento

Predicado que, dada uma lista, determina o seu comprimento.

```
% -------------------------------------------------------------
% Extensão do predicado comprimento: Lista, Resultado -> {V,F}

comprimento(X,Z):-
    length(X,Z).
```

Figura 46: Predicado comprimento

3.12.6 Testar

Este predicado testa se uma lista é válida.

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

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

Figura 47: Predicado teste

3.12.7 Somatório

Predicado que calcula o somatório de todos os elementos de uma lista (se o conjunto for vazio, considera-se o seu somatório = 0).

```
% -------------------------------------------------------------
% Extensao do predicado somatorio: lista, resultado -> {V,F}

somatorio([], 0).
somatorio([X|Y], R) :-
    somatorio(Y,G),
    R is X+G.
```

Figura 48: Predicado somatorio

3.12.8 Pertence

"pertence"verifica se um elemento está contido numa determinada lista.

```
% Extensao do predicado pertence: X,L -> {V,F}

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

Figura 49: Predicado pertence

3.12.9 Predicado não

Este predicado calcula o valor de verdade "contrário"à resposta de uma determinada questão.

```
% Extensao do predicado nao: Q -> {V,F}

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

Figura 50: Predicado nao

3.12.10 Insere ordenado

Predicado que insere ordenadamente um dado valor numa lista (ordenada).

```
% Extensao do predicado insereOrdenado: X,L,Resultado  -> {V,F}

insereOrdenado((X1,Y1),[],[(X1,Y1)]).
insereOrdenado((X1,Y1),[(X2,Y2)|Z],[(X1,Y1)|[(X2,Y2)|Z]]):-
    Y1>Y2.
insereOrdenado((X1,Y1), [(X2,Y2)|Z], [(X2,Y2)|R2]) :-
    Y1=<Y2,
    insereOrdenado((X1,Y1),Z,R2).
```

Figura 51: Predicado insereOrdenado

3.12.11 Concatena

Predicado responsável por concatenar a lista L1 e L2.

```
% Extensao do predicado concatena(L1,L2,L3)->{V,F}

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

Figura 52: Predicado concatena

3.13 Análise de Resultados

De seguida, são apresentados alguns testes efetuados, que comprovam o funcionamento dos predicados.

```
| ?- listing(recibo).
recibo(1, 3, marcos, anais, ginecologia, hospitalbraga, 13-2-2017, papanico, 30).
recibo(2, 4, vitor, guimaraes, cardiologia, hospitalbraga, 20-3-2017, pacemaker, 20).

yes
| ?- inserir(recibo(3, 1, carlos, famalicao, psiquiatria, hsog, 01-02-2017, hipnose, 15)).
yes
| ?- listing(recibo).
recibo(1, 3, marcos, anais, ginecologia, hospitalbraga, 13-2-2017, papanico, 30).
recibo(2, 4, vitor, guimaraes, cardiologia, hospitalbraga, 20-3-2017, pacemaker, 20).
recibo(3, 1, carlos, famalicao, psiquiatria, hsog, 1-2-2017, hipnose, 15).

yes
| ?- inserir(recibo(3, 1, carlos, famalicao, psiquiatria, hsog, 01-02-2017, hipnose, 15)).
no
| ?- inserir(recibo(2, 1, carlos, famalicao, psiquiatria, hsog, 01-02-2017, hipnose, 15)).
no
| ?- listing(recibo).
recibo(1, 3, marcos, anais, ginecologia, hospitalbraga, 13-2-2017, papanico, 30).
recibo(2, 4, vitor, guimaraes, cardiologia, hospitalbraga, 20-3-2017, pacemaker, 20).
recibo(3, 1, carlos, famalicao, psiquiatria, hsog, 1-2-2017, hipnose, 15).

yes
| ?-
```

Figura 53: Inserir recibo na base de conhecimento

```
--
| ?- listing(recibo).
recibo(1, 3, marcos, anais, ginecologia, hospitalbraga, 13-2-2017, papanico, 30).
recibo(2, 4, vitor, guimaraes, cardiologia, hospitalbraga, 20-3-2017, pacemaker, 20).
recibo(3, 1, carlos, famalicao, psiquiatria, hsog, 1-2-2017, hipnose, 15).

yes
| ?-
| ?-
| ?- gastoRegistadoPorUtente(3,R).
R = 30 ?
yes
| ?-
| ?- gastoRegistadoPorUtente(4,R).
R = 20 ?
yes
| ?-
| ?- gastoRegistadoPorUtente(23,R).
R = 0 ?
yes
| ?-
```

Figura 54: Obter gasto de determinado utente

```
| ?- utentesDePrestador(2,R).
R = [(4,vitor,20,guimaraes),(6,rita,26,pontedelima),(2,diana,20,trofa)] ?
yes
| ?- utentesDePrestador(6,R).
R = [(1,carlos,21,famalicao)] ?
yes
| ?- utentesDePrestador(9,R).
R = [(2,diana,20,trofa)] ?
yes
| ?- utentesDePrestador(111,R).
R = [] ?
yes
| ?-
```

Figura 55: Utentes cuidados pelos prestadores 2,6,9,111

```
| ?- cuidadosDeInstituicao(htrofa,R).
R = [(14-2-2017,2,7,olhoterapia,7),(20-4-2017,6,7,retinoterapia,35),(18-6-2017,2,8,reumatomagrafia,350)] ?
yes
| ?- cuidadosDeInstituicao(csjoane,R).
R = [(15-6-2017,1,1,terapiafala,5)] ?
yes
| ?- cuidadosDeInstituicao(uminho,R).
R = [] ?
yes
| ?-
```

Figura 56: Cuidados realizados no htrofa, no csjaone e na uminho

```
| ?- getInstPrestByUtente(2,R).
R = [(hsmm,5),(htrofa,7),(hospitalbraga,2),(htrofa,8),(hospitalbraga,9)] ?
yes
| ?-
```

Figura 57: Lista de (Instituição, IdPrestador) relativa ao utente com ID 2

4 Conclusões e Sugestões

A resolução deste primeiro exercício prático foi bastante importante e enriquecedora, pois permitiu aos membros do grupo perceber e interiorizar melhor os conceitos abordados nas aulas práticas da Unidade Curricular de Sistemas de Representação de Raciocínio e Conhecimento. Deste modo, foram adquiridos conhecimentos básicos acerca de programação em lógica com invariantes, consciência da sintaxe de *Prolog* e da ferramenta *Sicstus*, e entendimento para representar conhecimento e construir mecanismos de raciocínio para dar solução a problemas, que certamente serão úteis para o continuamento da UC.

No que diz respeito à representação de conhecimento, o grupo achou por bem, por forma a representar um sistema de cuidados de saúde mais preciso, adicionar o "recibo". Para além da representação de conhecimento, seria necessário definir invariantes de modo a manter a consistência e integridade do conhecimento. Assim, o grau de dificuldade desta parte foi relativamente baixo, já que o mais labiríntico seria a definição correta dos invariantes, sendo que a equipa ficou contente com os resultados obtidos.

No que toca à resolução das funcionalidades propostas pela equipa docente, juntamente com as adicionais relativas ao "recibo", estas acarretaram mais trabalho pelos elementos, uma vez que seria preciso elaborar um raciocínio que levasse à resolução correta do problema. Ainda assim, e através da correta divisão de tarefas e revisão por toda a equipa, foi conseguida uma solução simples e eficaz.

Em suma, é feita uma apreciação positiva relativamente ao trabalho realizado, visto que a implementação de todas as funcionalidades propostas foram conseguidas com sucesso. O grupo conseguiu tirar partido dos conhecimentos adquiridos neste projeto, sentido-se capaz de, num contexto futuro, aplicar os conceitos subjacentes de forma eficaz. Sugere-se apenas que, num outro contexto (como um projeto de grandes dimensões), seria benéfico que fossem implementadas um maior conjunto de funcionalidades adicionais para uma melhor gestão do sistema.

Referências

[1] [Analide, 2011] ANALIDE, César, NOVAIS, Paulo, NEVES, José,
"Sugestões para a Redação de Relatórios Técnicos",
Relatório Técnico, Departamento de Informática, Universidade do Minho, Portugal, 2011.

[2] [Bratko, 1986] BRAKTO, Ivan
"PROLOG: Programming for Artificial Intelligence",
British Library of Congress, Grã Bretanha, 2011.