Programação em Lógica e Invariantes

SRCR 2º SEMESTRE 2016/2017

Mestrado integrado em Engenharia Informática

Universidade do Minho, Campus de Gualtar

Bruno Miguel Salgado Machado	74941
Carlos Manuel Magalhães da Silva	75107
Gonçalo Leal da Mota Meireles Moreira	73591
João Rui de Sousa Miguel	74237
Paulo Jorge Machado Guedes	74411

Resumo

Foi-nos solicitada a criação de um sistema de representação de conhecimento e raciocínio com capacidade para caracterizar um universo de discurso na área da prestação de cuidados de saúde. Este sistema deveria ser compatível com a realização de serviços de atos médicos, cuidados prestados, utentes, entre outros.

Inicialmente representou-se a base de conhecimento, criando dados suficientes para depois testar a parte de computação do sistema. Foram definidos um número razoável de utentes, cuidados prestados (cuiprest) e atos médicos (amed). Decidimos também guardar as informações importantes dos dados anteriormente especificados, como as datas, instituições, cidades, nomes, moradas, descrições dos serviços médicos, de maneira a serem usados em outras funções.

Após criados dados suficientes, prosseguiu-se com o desenvolvimento das funções de computação do sistema. Começamos por criar funções para registar novos dados, especificando quais se pretendiam adicionar, juntamente com todas as informações relevantes para o registo.

De seguida, efetuou-se a criação de um número de funcionalidades mais complexas, como identificar os utentes por critérios de seleção (idade, nome e morada). Após criar inúmeras funções, utilizando os dados criados inicialmente, prosseguimos á evolução da base de conhecimento, criando um modo de adicionar dados á base de conhecimento, apoiado nalguns invariantes. Após implementada a evolução, a adição de dados à base de conhecimento, desenvolvemos funções para remover dados da base de conhecimentos, como remover utentes, cuidados e atos médicos.

Índice

Índice

Índice de Imagens

Índice

Introdução

Neste projeto, foi nos solicitado que elaborássemos um sistema de representação de conhecimento e raciocínio com capacidade para caracterizar um universo de discurso na área da prestação de cuidados de saúde pela realização de serviços de atos médicos. Para tal, utilizou-se a linguagem de programação *Prolog*, conjuntamente com a ferramenta *SICSTUS*.

Inicialmente representamos o conhecimento, depois, elaboramos um conjunto de casos práticos como registar utentes, cuidados prestados e atos médicos. Prosseguimos á implementação de casos mais complexos para caracterizar todo o universo de discurso na área da prestação de cuidados de saúde.

Para o sistema de representação de conhecimento baseamo-nos no Pressuposto do Mundo Fechado, Nomes Únicos e de Domínio Fechado, recorrendo assim á implementação do esquema de raciocínio relacional, descrevendo a informação em termos de valores atómicos e relações entre conjuntos de valores e, recorremos também á implementação de representação de informação não monótona.

Descrição do Trabalho

Meta-Predicado nao

Este meta-predicado serve apenas para negar o valor de uma questão, sendo que quando a questão falha, este retorna o valor verdadeiro, e quando a questão não falha, este retorna o valor falso. (Figura 1)

```
% Extensao do meta-predicado nao: Questao {V,F}
nao(Q) :- Q, !, fail.
nao(Q).
```

Figura 1 - Implementação meta-predicado "nao".

Meta-Predicado solucoes

Utiliza-se este predicado quando se pretende obter a listagem de todas as soluções possíveis S, para uma dada questão Q. Cujo o formato da lista é especificado por F.

Faz-se a utilização do predicado disponibilizado pelo *Prolog*, findall, visto que este não falha na eventualidade de não existir resposta a esta questão, ao contrário do que aconteceria com o bagoff. (Figura 2)

```
% Extensao do meta-predicado solucoes: F, Q, S -> {V,F}
solucoes(F, Q, S) :- findall(F, Q, S).
```

Figura 2 - Implementação meta-predicado "solucoes".

Predicado comprimento

O predicado comprimento é utilizado sempre que se pretende obter informação relativamente ao tamanho de uma dada lista. Pode por isto utilizar-se também para verificar este mesmo tamanho. (Figura 3)

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

Figura 3 - Implementação predicado "comprimento".

Meta-Predicado testar

Utiliza-se este predicado quando se pretende verificar se todas as questões existentes numa lista são verdadeiras, isto é conseguido através da implementação de recorrência. (Figura 4)

```
% Extensao do meta-predicado testar: Li -> {V,F}
testar([]).
testar([H | T]) :- H,
                   testar(T).
```

Figura 4 - Implementação meta-predicado "testar".

Meta-Predicado inserir

Sempre que se pretende inserir um novo conhecimento no nosso sistema de representação, deverá ser utilizado este predicado, para evitar inconsistências na base de conhecimento. (Figura 5)

```
% Extensao do meta-predicado inserir: F -> {V,F}
inserir(F) :- assert(F).
inserir(F) :- retract(F), !, fail.
```

Figura 5 - Implementação meta-predicado "inserir".

Meta-Predicado evolucao

Sempre que seja necessário introduzir novo conhecimento, deverá ser utilizada a função evolução, esta assegura-se que o conhecimento só é inserido se estiver em cumprimento dos invariantes específicos a cada modelo de dados.

Para assegurarmos que a evolução é feita corretamente, recorremos aos predicados, solucoes para obter a listagem de todos os invariantes para um predicado, inserir para adicionar este predicado a nossa base de conhecimento e testar para verificar se todas as condições permanecem verdadeiras. (Figura 6)

```
% Extensao do meta-predicado evolucao: F -> {V,F}
evolucao(F) :- solucoes(I, +F::I, Li),
               inserir(F),
               testar(Li).
```

Figura 6 - Implementação meta-predicado "evolucao".

Predicados morada e nome

O predicado “morada” suportará todo o conhecimento relativo às moradas de todos os nossos utentes. Enquanto o predicado nome, diz respeito ao conhecimento relativo ao nomo de todos os pacientes. Começamos por definir alguns casos exemplo, para que seja possível testar posteriormente.

Foram ainda definidos dois invariantes para cada um destes predicados, um para inserção e outro para remoção. Definimos que, apenas queremos uma instância de morada/nome igual, sendo que se duas pessoas habitarem a mesma casa ou tenham o mesmo nome, este conhecimento não apareça duplicado. Definimos ainda que nunca será possível remover moradas/nomes, para não correr o risco de mais tarde ser necessário voltar a introduzir este conhecimento. (Figuras 7 e 8)

```
% Invariante Estrutural:  nao permitir a insercao de conhecimento
%                         repetido
+morada(M) :: (
                solucoes( M, morada(M), S),
                comprimento(S, N),
                N =< 1
              ).
% Invariante Estrutural:  nao permitir a remocao de conhecimento
-morada(M) :: fail.
```

Figura 7 - Implementação dos invariantes de "morada".

```
% Invariante Estrutural:  nao permitir a insercao de conhecimento
%                         repetido
+nome(N) :: (
              solucoes( N, nome(N), S),
              comprimento(S,T),
              T =< 1
            ).
% Invariante Estrutural:  nao permitir a remocao de conhecimento
-nome(N) :: fail.
```

Figura 8 - Implementação dos invariantes de "nome".

Predicado utente

Começamos também por definir alguns casos de teste. Posteriormente definimos os invariantes que nos asseguram que não existem números de identificação (ID's) iguais aquando a inserção na nossa base de conhecimento. Aquando remoção de um utente, é verificada a existência de atos médicos aos quais este esteja associado, sendo que, caso seja encontrado algum ato então o utente não poderá ser removido. (Figura 9)

```
% Invariante Referencial: nao admitir mais do que 1 utente
%                         com o mesmo id
+utente(U,N,I,M) :: (
                      solucoes( U , utente( U , _ , _ , _ ) , S ),
                      comprimento( S , T ),
                      T =< 1
                    ).
% Invariante Referencial: nao admitir remocao de utentes que
%                         estejam ligados a um ato medico
-utente(U,N,I,M) :: (
                      solucoes( U, amed( D, U, S, P), S),
                      comprimento(S,T),
                      T==0
                    ).
```

Figura 9 - Implementação dos invariantes para o predicado "utente".

Predicado registarUtente

Este predicado deverá ser utilizado sempre que se pretenda registar um utente novo, não sendo necessário ao utilizador a introdução direta na função evolução.

Ao registar um novo utente serão também inseridos os conhecimentos relativos ao seu nome e à sua morada, na eventualidade de estes ainda não estarem presentes na nossa base de representação.

Se o id de utente for único então este predicado assegurar-se-á que um novo utente foi registado. (Figura10)

```
% Extensão do predicado registarUtente que permite a evolucao
% do conhecimento de utentes
registarUtente(U,N,I,M):- evolucao(utente(U,N,I,M)),
                          N,
                          M.

                          ...
```

Figura 10 - Implementação de parte do predicado "registarUtente".

Predicado removerUtente

Ao contrário do predicado anterior, este serve para remover conhecimento relativo aos utentes.

Para que tal seja possível, recorremos a um predicado que falaremos posteriormente, o involução, e que é utilizado sempre que se pretende remover conhecimento. Vamos, portanto, testar o invariante de remoção, e caso não existam atos médicos relativos a este utente, podemos considera-lo como removido. (Figura 11)

```
% Extensão do predicado removerUtente que permite a regressao
% do conhecimento de utentes
removerUtente(U,N,I,M):- involucao(utente(U,N,I,M)).
```

Figura 11 - Implementação do predicado "removerUtente".

Predicados descricao, instituicao, cidade e data

Tal como o que acontecia nos predicados nome e morada, os predicados descricao, instituicao, cidade e data servem para que consigamos armazenar todas as informações relativas a estes assuntos.

Definimos, também, casos testes para cada um destes, e definimos ainda dois invariantes, um para inserção e outro para remoção que garantem tanto a inexistência de valores duplicados como a eliminação de conhecimento da nossa base de informação. (Figuras 12, 13, 14 e 15)

```
% Invariante Estrutural:  nao permitir a insercao de conhecimento
%                         repetido
+descricao(D) :: (
                    solucoes( D , descricao(D) , S ),
                    comprimento( S , N ),
                    N =< 1
                  ).
% Invariante Estrutural:  nao permitir a remocao de conhecimento
-descricao(D) :: fail.
```

Figura 12 - Implementação dos invariantes do predicado "descricao".

```
% Invariante Estrutural:  nao permitir a insercao de conhecimento
%                         repetido
+instituicao(I) :: (
                    solucoes( I, instituicao(I), S),
                    comprimento(S, N),
                    N =< 1
                   ).
% Invariante Estrutural:  nao permitir a remocao de conhecimento
-instituicao(I) :: fail.
```

Figura 13 - Implementação dos invariantes do predicado "instituicao".

```
% Invariante Estrutural:  nao permitir a insercao de conhecimento
%                         repetido
+cidade(D) :: (
                solucoes( D, cidade(D), S),
                comprimento(S, N),
                N =< 1
              ).
% Invariante Estrutural:  nao permitir a remocao de conhecimento
-cidade(D) :: fail.
```

Figura 14 - Implementação dos invariantes do predicado "cidade".

```
% Invariante Estrutural:  nao permitir a insercao de conhecimento
%                         repetido
+data(D, M, A) :: (
                    solucoes( (D, M, A) , (data(D, M, A)) , S ),
                    comprimento(S, N),
                    N =< 1
                   ).
% Invariante Estrutural:  nao permitir a remocao de conhecimento
-data(D, M, A) :: fail.
```

Figura 15 - Implementação dos invariantes do predicado "data".

Predicado cuiprest

Este predicado é utilizado para representar o conhecimento relativo a todos os cuidados prestados aos utentes em cada instituição e cidade.

Contém dois invariantes, um que verifica se o id de inserção do cuidado prestado é repetido (caso verdadeiro não se insere conhecimento ambíguo), e outro que verifica se é possível remover dados de um cuidado prestado, verificando se existem ocorrências deste na listagem de todos os atos médicos. (Figura 16)

```
% Invariante Referencial: nao admitir mais do que 1 cuidado prestado
%                         com o mesmo id
+cuiprest(U,D,I,C) :: (
                      solucoes( U , cuiprest( U , _ , _ , _ ) , S ),
                      comprimento( S , T ),
                      T =< 1
                     ).
% Invariante Referencial: nao admitir a remocao de um cuidado prestado
%                         que esteja ligado a um ato medico
-cuiprest(U,D,I,C) :: (
                      solucoes( S , amed( _ , _ , S , _ ) , R ),
                      comprimento( R , T ),
                      T == 0
                     ).
```

Figura 16 - Implementação dos invariantes para o predicado "cuiprest".

Predicado registarCuiPrest

Para que nos seja possível registar um novo cuidado prestado, deveremos recorrer a este predicado. Este assegura-se que todos os requisitos para que o registo seja efetuado estão cumpridos, sendo que caso seja necessário adicionar algum conhecimento extra, isto será feito sem que o utilizador se aperceba. (Figura 17)

```
% Extensão do predicado registarCuiPrest que permite a evolucao
% do conhecimento de cuidados prestados
registarCuiPrest(U,D,I,C):- evolucao(cuiprest(U,D,I,C)),
                            evolucao(D),
                            evolucao(I),
                            evolucao(C).

                            ...
```

Figura 17 - Implementação de parte do predicado "registarCuidPrest".

Predicado removerCuiPrest

Tal como é possível inserir novos cuidados prestados, também é possível retira-los.

Para isto deverá ser utilizado este predicado, que com recurso ao predicado involução, assegura-se que todos os invariantes são cumpridos sem afetar a consistência da nossa base de conhecimento (neste caso, assegura-se que não existam atos médicos com este cuidado prestado). (Figura 18)

```
% Extensão do predicado removerCuiPrest que permite a regressao
% do conhecimento de cuidados prestados
removerCuiPrest(U,D,I,C):- involucao(cuiprest(U,D,I,C)).
```

Figura 18 - Implementação do predicado "removerCuiPrest".

Predicado amed

Este predicado serve para representar o conhecimento que diz respeito à listagem de atos médicos realizados. Ao contrário de outros predicados que já falamos (data, nome, cidade, …) este é descrito por apenas um invariante, o de inserção, não existe limitações a esta base de conhecimento.

O invariante de inserção assegura-se que um utente pode receber vários cuidados médicos num determinado dia, desde que estes cuidados sejam diferentes. (Figura 19)

```
% Invariante Referencial: nao admitir mais do que 1 ato medico
%                         com a mesma data e com o mesmo id de utente
%                         e cuidado prestado
+amed(D,I,I2,C) :: (
                      solucoes( (D, I, I2) , amed( D , I , I2 , _ ) , S ),
                      comprimento( S , T ),
                      T =< 1
                    ).
```

Figura 19 - Implementação do invariante para o predicado "amed".

Predicado registaAMed

Este predicado deverá ser utilizado sempre que se pretenda introduzir novos atos médicos, sendo que assegura que isto apenas ocorre quando todos os dados a inserir estão em conformidade com os dados já presentes. Para tal recorremos ao predicado evolução que falamos anteriormente. (Figura 20)

```
% Extensão do predicado registarAmed que permite a evolucao
% do conhecimento de atos medicos
registaAMed(U,D,I,C) :- evolucao(amed(U,D,I,C)).
```

Figura 20 - Implementação do predicado "registaAmed".

Predicado removerAMed

Tal como a remoção de cuidados prestados, existe este predicado para se assegurar da remoção de dados relativos aos atos médicos. Este predicado utiliza também o predicado involução, que se assegura que não existem inconsistências no nosso sistema de conhecimento, após remoção de informação. (Figura 21)

```
% Extensão do predicado registarAmed que permite a regressao
% do conhecimento de atos medicos
removerAMed(U,D,I,C) :- involucao(amed(U,D,I,C)).
```

Figura 21 - Implementação do predicado "removerAMed".

Predicado concatenar

Este predicado serve para que consigamos concatenar duas listas, obtendo uma terceira que será o resultado da agregação das duas primeiras. Poderá ser utilizada diretamente pelo utilizador, no entanto o nosso intuito foi auxiliar na implementação de outros predicados.

Para a junção de listas foi necessário a criação do predicado concatenar de modo a ser utilizado como um predicado auxiliar. Este permite a junção de duas listas numa só e faz isto adicionando à segunda lista os elementos da primeira lista, recursivamente. (Figura 22)

```
% Extensao do predico para concatenar duas listas,
% concatenar: Lista1, Lista2, e Listaf -> {V,F}
concatenar([], L2, L2).
concatenar([X|L1], L2, [X|L]) :- concatenar(L1, L2, L).
```

Figura 22 - Implementação do predicado "concatenar".

Predicado filtraIdade

Este predicado deverá ser utilizado sempre que se pretende obter informação relativa à nossa listagem de utentes. Recebe três parâmetros, um que simboliza a nossa condição de procura, outro relativo à idade e um terceiro que serve como lista de retorno.

Para se conseguir obter os nomes de utentes que têm uma idade maior ou menor, decidiu-se ir obtendo, recursivamente todos os utentes com uma certa idade, calculando, a cada passagem, o nome dos utentes com essa mesma idade, através da coincidência de padrões na base de conhecimento.

Para serem calculados todos os utentes, à idade requerida era retirado ou acrescentado, conforme o caso maior ou menor, uma unidade. Esta recursividade parava, no caso do menor com o valor 0 e no caso do maior com o valor 110 (máxima idade considerada).

Se fosse requerido somente utentes com uma determinada idade, era utilizado um predicado que calculava todos os utentes com aquela idade e apresentava todos os nomes dos utentes correspondentes. (Figura 23)

```
% Extensao do predicado de filtrar por idade,
% filtraIdade Condicao,Idade,Lista -> {V,F}
filtraIdade( <, 0, [] ).
filtraIdade( <,I,N2 ) :- R is I-1,
                         filtraIdade( <, R, N ),
                         solucoes( No, utente( _, No, R, _ ), L1 ),
                         concatenar( L1, N, N2 ).
filtraIdade( >, 110, [] ).
filtraIdade( >, I, N2 ) :- R is I+1,
                         filtraIdade( >, R, N ),
                         solucoes( No, utente( _, No, R, _ ), L1 ),
                         concatenar( L1, N, N2 ).
filtraIdade( =, I, L ) :- solucoes( No, utente( _, No, I, _ ), L ).
```

Figura 23 - Implementação do predicado "filtraIdade".

Predicados filtraMorada e filtraId

Estes predicados têm como base o predicado anterior, em que se apresenta os nomes dos utentes numa lista. A diferença é que, neste caso, a lista esta filtrada por uma determinada morada ou por um determinado ID. Estes três predicados funcionam de maneira muito semelhante, testam a base de conhecimento e quando há coincidência com a morada ou ID pretendido, esse utente, mais precisamente o seu nome, é colocado numa lista.

No fim da execução do predicado, esta lista terá todos os nomes de utentes, que tem uma morada ou ID correspondente pretendido. (Figuras 24 e 25)

```
% Extensao do predicado para filtrar utentes por morada,
% filtraMorada: Morada, Lista -> {V,F}
filtraMorada(R,L) :- solucoes(No, utente(_, No, _, R), L).
```

Figura 24 - Implementação do predicado "filtraMorada".

```
% Extensao do predicado para filtrar utentes por id,
% filtraId: ID, Lista -> {V,F}
filtraId(I,L) :- solucoes(No, utente(I, No , _ , _), L).
```

Figura 25 - Implementação do predicado "filtraId".

Predicados filtraCuiPrest e utentesByID

Finalmente identificou-se os utentes que estão ligados a um determinado serviço.

Para a elaboração deste predicado, usa-se a concordância de padrões, mais uma vez, na base de conhecimento dos cuidados prestados. Parte-se da descrição de cuidado prestado, retira-se o ID que corresponde a este cuidado, através deste ID é verificada a existência de atos médicos com este cuidado prestado. No caso em que tal se verifique, é extraído o ID do utente associado, e, posteriormente, coloca-se numa lista o nome associado a este ID de utente. (Figura 26)

Para obter a listagem de nomes de utente a partir do seu ID, utiliza-se o predicado utentesByID, que dado uma lista de id's de utentes, calcula a lista de nomes respetivos. (Figura 27)

```
% Extensao do predicado para filtrar utentes por cuidado prestado,
% filtrarCuiPrest: Descrição, Lista -> {V,F}
filtrarCuiPrest( D, L ) :-
                cuiprest(X,D,_,_),
                solucoes(N,amed(_,N,X,_),L1),
                utentesByID(L1,L).
```

Figura 26 - Implementação do predicado "filtraCuidPrest".

```
% Extensao do predicado para mostrar os utentes a partir de uma
% lista com ids de utentes
% utentesByID: Lista1, Lista -> {V,F}
utentesByID([],[]).
utentesByID([H|T],L) :-
          utente(H,N,_,_),
          utentesByID(T,R),
          concatenar([N],R,L).
```

Figura 27 - Implementação do predicado "utentesByID".

Predicados mostraInstituicoes, cuiPrestInst e cuiPrestCidade

Estes predicados servem para que o utilizador consiga obter a listagem de todas as instituições presentes na nossa base de conhecimento, de todos os cuidados prestados numa instituição e de todos os cuidados prestados numa cidade, respetivamente.

Para implementar todos estes predicados recorre-se ao predicado soluções, garantindo que procuramos pelo conhecimento requerido pelo utilizador, de maneira correta e no formato apropriado, retornando a listagem apropriada a cada um destes predicados. (Figuras 28, 29 e 30)

```
% Extensao do predicado para mostrar todas as instituicoes
% prestadores de cuidados de saude
% mostraInstituicoes: Lista -> {V,F}
mostraInstituicoes(L) :- solucoes( I ,cuiprest( _, _, instituicao(I), _) , L).
```

Figura 28 - Implementação do predicado "mostraInstituicoes".

```
% Extensao do predicado para mostrar os cuidados prestados
% numa determinada instituicao
% cuiPrestInst: Instituicao, Lista -> {V,F}
cuiPrestInst(I,L) :- solucoes( cuiprest( U, D, I, C), cuiprest( U, D, I, C), L).
```

Figura 29 - Implementação do predicado "cuiPrestInst".

```
% Extensao do predicado para mostrar os cuidados prestados
% numa determinada cidade
% cuiPrestCidade: Cidade, Lista -> {V,F}
cuiPrestCidade(C,L) :- solucoes(cuiprest(U,D,I,C),cuiprest(U,D,I,C), L).
```

Figura 30 - Implementação do predicado "cuiPrestCidade".

Predicado utent

Este predicado é utilizado como método auxiliar para o cálculo de utentes. Sempre que se pretende obter a listagem de utentes a partir dos seus ids, pode recorrer-se a este predicado. Permite consultar, recursivamente, a base de conhecimento a partir do predicado solucoes , para extrair os utentes, a partir da lista de Id's e juntando-os numa única lista para visualização a partir do predicado concatenar. (Figura 31)

```
% Extensao do predicado para mostrar os utentes a partir de uma
% lista com ids de utentes
% utent: Lista1, Lista -> {V,F}
utent([],[]).
utent([U|T],L) :- utent(T,L1),
                  solucoes(utente(U,N,I,M),utente(U,N,I,M),L2),
                  concatenar(L2,L1,L).
```

Figura 31 - Implementação do predicado "utent".

Predicado idU

Este predicado serve, também, como método auxiliar. Por si só, apenas calcula os ids dos utentes tendo como partida uma lista contendo os ids dos cuidados prestados. Este predicado percorrerá todos os atos médicos para interligar os ids dos cuidados prestados com os ids dos utentes. (Figura 32)

```
% Extensao do predicado para, a partir dos ids de cuidados prestados
% obter os ids de utentes que estavam ligados a atos medicos,
% idU: Lista1, Lista -> {V,F}
idU([],[]).
idU([Is|T],L) :-  idU(T,L1),
                  solucoes(H,amed(_,H,Is,_),L2),
                  concatenar(L2,L1,L).
```

Figura 32 - Implementação do predicado "idU".

Predicados idSr e uInst

Estes predicados servem para obter os ids de cuidados prestados, e utentes, respetivamente, relativos a uma instituição. Para que seja possível obter a lista de cuidados prestados numa instituição utiliza-se o predicado idSr que recorre ao predicado solucoes.

Para que seja possível obter a listagem de utentes de uma instituição é necessário realizar, numa fase inicial, a listagem de todos os cuidados da instituição, depois ver os utentes a quem foram prestados aqueles cuidados, e finalmente obtém-se a listagem de utentes que integram a instituição. (Figuras 33 e 34)

```
% Extensao do predicado para,a partir de uma instiuicao,
% obter os ids de cuidados prestados,
% idSr: ID, Lista -> {V,F}
idSr(I,L) :- solucoes(U,cuiprest(U,_,I,_),L).
```

Figura 33 - Implementação do predicado "idSr".

```
% Extensao do predicado para, a partir de uma instiuicao,
% obter os utentes de uma instituicao,
% uInst: Instituicao, Lista -> {V,F}
uInst(I,L) :- idSr(I,A),
              idU(A,B),
              utent(B,L).
```

Figura 34 - Implementação do predicado "uInst".

Predicado idUt

Para dar resposta ao requisito 'identificar utentes de um serviço' foi necessário a criação de um predicado idUt auxiliar, que dado um id de serviço produz uma lista com os ids dos utentes, dos atos médicos desse serviço. Este predicado foi feito com o auxilio do predicado solucoes. (Figura 35)

```
% Extensao do predicado para, a partir de um id de um serviço,
% obter os ids utentes que estao ligados ao ato medico daquele serviço,
% idUt: ID, Lista -> {V,F}
idUt(Is,L) :- solucoes(H,amed(_,H,Is,_),L).
```

Figura 35 - Implementação do predicado "idUt".

Predicado uServ

O predicado uServ dá resposta ao requisito 'identificar utentes de um serviço'. Este predicado foi concebido á custa da chamada encadeada dos predicados idUt e utent respetivamente. (Figura 36)

```
% Extensao do predicado para, a partir de um id de um serviço,
% obter os utentes que estao ligados ao ato medico daquele serviço,
% idUt: ID, Lista -> {V,F}
uServ(IdS,L) :- idUt(IdS,A),
                utent(A,L).
```

Figura 36 - Implementação do predicado "uServ".

Predicado aMedUtent

Este predicado foi criado para dar resposta ao requisito 'identificar atos médicos realizados por utente'. Este predicado recebe como argumento um utente e produz uma lista como os atos médicos respetivos a esse mesmo utente. O predicado aMedUtent foi realizado á custa do predicado solucoes. (Figura 37)

```
% Extensao do predicado para, a partir de um utente, obter a listagem
% de todos os atos medicos por este realizado.
% aMedUtent: Utente, Lista -> {V,F}
aMedUtent(utente(I,_,_,_),L) :-  solucoes(amed(A,I,B,C),amed(A,I,B,C),L).
```

Figura 37 - Implementação do predicado "aMedUtent".

Predicado instServ

O predicado instServ foi criado como predicado auxiliar para dar resposta ao requisito, 'identificar atos médicos por instituição'. Este predicado usa como auxilio o predicado solucoes e recebe como argumento uma instituição e produz como resultado uma lista como os ids de serviçoes prestados pela respetiva instituição. (Figura 38)

```
% Extensao do predicado, para obter a listagem de todos os
% ids de serviço prestados numa instituicao
% instServ: instituicao, Lista -> {V,F}
instServ(I,L) :- solucoes(U,cuiprest(U,_,I,_),L).
```

Figura 38 - Implementação do predicado "instServ".

Predicado idServAmed

Este predicado foi criado como predicado auxiliar para dar resposta ao requisito 'identificar atos médicos por instituição'. Este predicado recebe como argumento uma lista de ids de serviço e produz a lista dos respetivos atos médicos. Este resultado foi obtido através da chamada recursiva do predicado solucoes e o predicado concatenar. (Figura 39)

```
% Extensao do predicado, para obter a listagem de todos os
% atos médicos correspondentes aos ids de servico presentes numa
% lista
% idServAmed: Lista (Ids servico), Lista (Atos medicos) -> {V,F}
idServAmed([],[]).
idServAmed([H|T],L):- idServAmed(T,L1),
                      solucoes(amed(U,N,H,M),amed(U,N,H,M),L2),
                      concatenar(L2,L1,L).
```

Figura 39 - Implementação do predicado "idServAmed".

Predicados amedInst e amedServ

Estes predicados foram criados para que seja possível obter a listagem de todos os atos médicos relativos a uma instituição, ou a um serviço, respetivamente. Para faze-lo recorre-se a predicados predefinidos anteriormente, e utiliza-se ainda o predicado solucoes. (Figuras 40 e 41)

```
% Extensao do predicado, para obter a listagem de todos os
% atos médicos correspondentes a uma instituicao
% amedInst: Instituicao, Lista -> {V,F}
amedInst(I,L) :- instServ(I,P),
                 idServAmed(P,L).
```

Figura 40 - Implementação do predicado "amedInst".

```
% Extensao do predicado, para obter a listagem de todos os
% atos médicos correspondentes a um servico
% amedServ: Servico, Lista -> {V,F}
amedServ(I,L) :- solucoes(amed(D,I2,I,C),amed(D,I2,I,C),L).
```

Figura 41 - Implementação do predicado "amedServ".

Predicados inSerAux e inSer

Um dos requisitos para o sistema de representação de conhecimento será determinar todas as instituições/serviços a que um utente já recorreu. Para tal, necessitamos de procurar na nossa base de conhecimento, todos os atos médicos que pertençam ao utente que procuramos, e de seguida necessitamos determinar que cuidados prestados são dados em que instituição. (Figuras 42 e 43)

```
% servicos prestados por instituicao apartir da listagem de
% ids de cuidados prestados
% inSerAux: Lista (Ids cuidprest), Lista -> {V,F}
inSerAux([],[]).
inSerAux([H|T],L):- solucoes((I,H,D),cuiprest(H,D,I,_),L1),
                    inSerAux(T,L2),
                    concatenar(L1,L2,L).
```

Figura 42 - Implementação do predicado auxiliar "inSerAux".

```
% Extensao do predicado, para obter a listagem de todos os
% servicos prestados por instituicao a um utente
% inSer: Utente, Lista -> {V,F}
inSer(U,L):-  solucoes(I,amed(_,U,I,_),L1),
              inSerAux(L1,L).
```

Figura 43 - Implementação do predicado "inSer".

Predicado custoData

Para desenvolver o sistema de representação de conhecimento na área de serviços médicos necessitamos também calcular o custo total dos atos médicos por utente / serviço / instituição / data.

Para calcular o custo total por data, procuramos todos os atos médicos que foram efetuados na respetiva data, e somou-se os custos de cada um destes. (Figura 44)

```
% Extensao do predicado, para obter o custo total de todos
% os servicos efetuados numa data
% custoData: Data, Resultado -> {V,F}
custoData(D,R):-  solucoes(C,amed(D,I2,I,C),L),
                  soma(L,R).
```

Figura 44 -Implementação do predicado "custoData".

Predicados custoInstituicao e custoInstituicaoAux

Podemos definir estes predicados como semelhantes ao custo data, estes aplicam-se quando necessitamos de saber qual o custo total dos atos médicos por instituição. É necessário aceder á base de conhecimento para procurar quais atos médicos foram efetuados na respetiva instituição, e de seguida somar os custos desses atos médicos. (Figuras 45 e 46)

```
% Extensao do predicado, para obter o custo total de todos
% os servicos efetuados numa instituicao
% custoInstituicao: Instituicao, Resultado -> {V,F}
custoInstituicao(I,R):- solucoes(S,cuiprest(S,_,I,_),L),
                        custoInstituicaoAux(L,L2),
                        soma(L2,R).
```

Figura 45 - Implementação do predicado "custoInstituicao".

```
% Extensao do predicado, para obter uma lista com o custo
% de todos os servicos presentes numa outra lista de ids de cuidados
% prestados
% custoInstituicaoAux: Lista (ids cuidpres), Lista -> {V,F}
custoInstituicaoAux([],[]).
custoInstituicaoAux([H|T],L) :- solucoes(C,amed(D,I2,H,C),L2),
                                custoInstituicaoAux(T,L1),
                                concatenar(L2,L1,L).
```

Figura 46 - Implementação do predicado "custoInstituicaoAux".

Predicado custoServico

Quando se necessita de calcular o custo total de um dado serviço, acede-se novamente á base de conhecimento para extrair os custos de todos os atos médicos que sejam desse dado serviço, somando de seguida os respetivos custos. (Figura 47)

```
% Extensao do predicado, para obter o total gasto num dado servico
% custoServico: Id servico, Resultado -> {V,F}
custoServico(S,R):- solucoes(C,amed(D,I2,S,C),L2),
                    soma(L2,R).
```

Figura 47 - Implementação do predicado "custoServico".

Predicado soma

Desenvolvemos este predicado com o intuito de servir como método auxiliar para outros predicados. O seu objetivo é calcular a soma de todos os valores presentes numa lista. Isto é conseguido, pela iteração consecutiva por todos os elementos da lista. (Figura 48)

```
% Extensao do predicado, para obter a soma de todos os valores
% existentes numa lista
% soma: Lista, Resultado -> {V,F}
soma([],0).
soma([H|T],R):- soma(T,N),
                R is N+H.
```

Figura 48 - Implementação do predicado "soma".

Predicado custoUtente

Para calcular o custo de um dado utente na área dos serviços médicos, precisamos de extrair todos os atos médicos efetuados a esse utente, somando de seguida os respetivos custos, providenciando assim o custo total do dado utente. (Figura 49)

```
% Extensao do predicado, para obter o total gasto por um utente
% custoUtente: Id utente, Resultado -> {V,F}
custoUtente(U,R) :- solucoes(C,amed(_,U,_,C),L),
                    soma(L,R).
```

Figura 49 - Implementação do predicado "custoUtente".

Predicados remover e involucao

O predicado involucao, através de solucoes, obtém todos os invariantes de remoção correspondentes ao conhecimento que se quer eliminar. De seguida, é usado o remover, que retira esse conhecimento da base. São então testados os invariantes relativos ao caso em análise com o predicado testar. Caso o conhecimento a eliminar passe em todos os seus invariantes mantém-se eliminado, caso contrário há uma regressão e no predicado remover é inserido novamente o conhecimento. (Figuras 50 e 51)

```
% Extensao do meta-predicado remover: F -> {V,F}
remover(F) :- retract(F).
remover(F) :- assert(F), !, fail.
```

Figura 50 - Implementação do predicado "remover".

```
% Extensao do meta-predicado evolucao: F -> {V,F}
involucao(F) :- solucoes(I, -F::I, Li),
                remover(F),
                testar(Li).
```

Figura 51 - Implementação do predicado "involucao".

Análise de Resultados

Vamos agora dar exemplo de alguns dos predicados que testamos, utilizando a ferramenta *SICSTUS*.

Assim que iniciamos o nosso sistema, temos as informações dos dados que tínhamos preenchido previamente, para que seja possível testa-los agora. Vejamos de seguida a listagem dos elementos que compõem o que consideramos como utentes e como moradas.

```
| ?- listing(utente).
utente(u1, nome('joao miguel'), 30, morada('rua de barros')).
utente(u2, nome('bruno machado'), 20, morada('rua de santa apolonia')).
utente(u3, nome('carlos silva'), 20, morada('rua do souto')).
utente(u4, nome('paulo guedes'), 20, morada('rua penedo da forca')).
utente(u5, nome('goncalo moreira'), 20, morada('rua padre francisco babo')).
```

Figura 52 - Lista inicial de utentes.

```
| ?- listing(morada).
morada('rua de barros').
morada('rua de santa apolonia').
morada('rua do souto').
morada('rua penedo da forca').
morada('rua padre francisco babo').
```

Figura 53 - Lista inicial de moradas.

De seguida procede-se à inserção de um novo utente. Podemos verificar que a listagem de utentes que existiam no nosso sistema de representação de conhecimento, já contem este novo elemento. Verifica-se ainda que, indiretamente, a listagem de morada também foi atualizada, sem que o utilizador sem incomodar o utilizador com atualizações desnecessárias.

```
| ?- registarUtente(u6, nome('Manuel Alemão'), 40, morada('Rua das Tulipas')).
yes
| ?- listing(utente).
utente(u1, nome('joao miguel'), 30, morada('rua de barros')).
utente(u2, nome('bruno machado'), 20, morada('rua de santa apolonia')).
utente(u3, nome('carlos silva'), 20, morada('rua do souto')).
utente(u4, nome('paulo guedes'), 20, morada('rua penedo da forca')).
utente(u5, nome('goncalo moreira'), 20, morada('rua padre francisco babo')).
utente(u6, nome('Manuel Alemão'), 40, morada('Rua das Tulipas')).
```

Figura 54 - Registar novo utente.

```
| ?- listing(morada).
morada('rua de barros').
morada('rua de santa apolonia').
morada('rua do souto').
morada('rua penedo da forca').
morada('rua padre francisco babo').
morada('Rua das Tulipas').
```

Figura 55 - Morada após registo.

De todos estes moradores estamos interessados em filtrar por idade, queremos, numa primeira fase, saber quem tem idade superior a 25 anos, e, posteriormente, quem tem uma idade inferior a 26 anos.

```
| ?- filtraIdade(>, 25, R).
R = [nome('joao miguel'),nome('Manuel Alemão')] ?
yes
```

Figura 56 - Utentes com idade superior a 25 anos.

```
| ?- filtraIdade(<, 26, R).
R = [nome('bruno machado'),nome('carlos silva'),nome('paulo guedes'),nome('gonc
alo moreira')] ?
yes
```

Figura 57 - Utilizadores com idade inferior a 26 anos.

Vejamos agora o id do utente que realizou o ato medico com o cuidado prestado de código de id = c4. Para que seja possível confirmar, vejamos ainda a listagem de todos os atos médicos presentes neste sistema.

```
| ?- idUt(c4, L).
L = [u1] ?
yes
| ?- listing(amed).
amed(data(21,2,2017), u1, c4, 100).
amed(data(21,2,2017), u1, c2, 100).
amed(data(12,1,2017), u2, c5, 10).
amed(data(30,1,2017), u3, c3, 5).
amed(data(24,2,2017), u4, c1, 20).
amed(data(24,2,2017), u3, c1, 20).
amed(data(15,3,2017), u5, c2, 45).

yes
```

Figura 58 - Verificar utentes de um cuidado médico.

Imaginemos agora que se pretende saber qual a quantidade total, gasta em todos os dados do nosso sistema para o dia de 24 fevereiro 2017. Fazemos a listagem para que seja mais fácil conferir os valores.

```
| ?- listing(amed).
amed(data(21,2,2017), u1, c4, 100).
amed(data(21,2,2017), u1, c2, 100).
amed(data(12,1,2017), u2, c5, 10).
amed(data(30,1,2017), u3, c3, 5).
amed(data(24,2,2017), u4, c1, 20).
amed(data(24,2,2017), u3, c1, 20).
amed(data(15,3,2017), u5, c2, 45).

yes
| ?- custoData( data(24,2,2017),L ).
L = 40 ?
yes
```

Figura 59 - Verificar o total gasto num determinado dia.

Como se pode verificar, o total gasto neste dia foi de 40€, confirmável pela listagem de todos os atos médicos presentes no sistema. Podemo-nos ainda, querer saber a listagem de todas as instituições que existem no nosso sistema, utilizando por isso o predicado mostraInstituicoes.

```
| ?- mostraInstituicoes(L).
L = ['sao joao','unidade local de saude','sao joao','hospital beatriz angelo','
centro hospitalar'] ?
yes
```

Figura 60 - Listagem de instituições do nosso sistema.

Finalmente consideremos que se pretende proceder à remoção de um dado ato médico, por qualquer motivo que seja. Recorremos ao predicado removerAMed. De seguida

prossegue-se com esta remoção, verificando previamente e posteriormente como garantia de que removemos a nossa informação da base de conhecimento com sucesso.

```
| ?- listing(amed).
amed(data(21,2,2017), u1, c4, 100).
amed(data(21,2,2017), u1, c2, 100).
amed(data(12,1,2017), u2, c5, 10).
amed(data(30,1,2017), u3, c3, 5).
amed(data(24,2,2017), u4, c1, 20).
amed(data(24,2,2017), u3, c1, 20).
amed(data(15,3,2017), u5, c2, 45).

yes
| ?- removerAMed(data(15,3,2017), u5, c2, 45).
yes
| ?- listing(amed).
amed(data(21,2,2017), u1, c4, 100).
amed(data(21,2,2017), u1, c2, 100).
amed(data(12,1,2017), u2, c5, 10).
amed(data(30,1,2017), u3, c3, 5).
amed(data(24,2,2017), u4, c1, 20).
amed(data(24,2,2017), u3, c1, 20).

yes
```

Figura 61 - Remoção de um ato médico.

Acabamos assim por testar todos os predicados, sendo que, dada a extensibilidade do relatório, optámos por deixar de fora aqueles que têm uma implementação igual ou semelhante à implementação daqueles que aqui estão exemplificados. Por fim, asseguramos que todos os predicados funcionam corretamente, aquando inserção, remoção e consulta.

Conclusão

No desenvolvimento do sistema de representação notámos que quando tentávamos fazer a involução e evolução da base de conhecimento, caso não existissem invariantes, a coerência do sistema era quase nula, existiam muitas informações repetidas quer relativas a nomes, datas, moradas e afins. Assim, entendeu-se que a existência de invariantes tem um papel fulcral na boa gestão de um sistema de dados.

Criou-se invariantes para apoiar a evolução e a involução. Estipulamos regras que não permitem a inserção de dados repetidos, nem remoção de dados que estejam a ser utilizados como referências. Não será possível remover cuidados prestados, ou utentes, se estes estiverem registados em atos médicos e não se poderá remover nomes, datas, instituições visto que não são dados únicos, ou seja, poderão ser usados na criação de dados futuros.

Com isto em mente, consideramos importante manter estes dados, considerando que nunca podem ser retirados. Uma morada vai sempre existir, bem como um nome, uma instituição, cidade, …