Universidade do Minho
Departamento de Informática
Mestrado Integrado em Engenharia Informática

Relatório do Exercício 1

Programação em Lógica e Invariantes

Sistemas de Representação de Conhecimento e Raciocínio

Grupo de trabalho 20
Ana Fernandes, A74321
Diogo Machado, A75399
Miguel Miranda, A74726
Rui Leite, A75551

Braga, 19 de Março de 2017

Conteúdo

Resumo

Este relatório tem como objetivo explicar o desenvolvimento das tarefas descritas no enunciado do primeiro exercício prático da Unidade Curricular de Sistemas de Representação de Conhecimento e Raciocínio.

O exercício proposto baseia-se na implementação de um sistema de representação de conhecimento e raciocínio capaz de caracterizar eventos e factos relacionados com utentes, cuidados de saúde e atos médicos.

Pretende-se explicitar todos os passos para a realização das funcionalidades implementadas, fazendo uma apresentação das escolhas feitas para representar a informação pretendida e da Base de Conhecimento, explicando o desenvolvimento dos predicados e analisando os resultados obtidos.

Introdução

O principal objetivo da programação em lógica prende-se com a contextualização e estruturação de um programa, cujo conteúdo se baseia, essencialmente, num conjunto de axiomas ou factos (algo que se sabe que é verdade), de regras de inferência e de predicados. A este conjunto chama-se, normalmente, Base de Conhecimento [2].

Este programa é construído com o propósito de serem feitas interrogações sobre o seu conteúdo e obter respostas. O "motor de inferência" pesquisa a Base de Conhecimento à procura de axiomas e regras que permitam, por um algoritmo de dedução ("resolução de 1.ª ordem"), validar uma resposta.

A linguagem de programação utilizada é o Prolog: uma linguagem declarativa que usa fragmentos da lógica proposicional de 1.ª ordem (*Cláusulas de Horn*) para representar o conhecimento sobre um dado problema e recorre a processos de inferência lógica para concluir o valor de verdade duma proposição em função da sua "relação" com os factos [2].

O *Prolog* recorre a um conjunto de mecanismos, como o *pattern matching*, *tree-based data structuring* e *automatic backtracking*, para dar respostas a problemas complexos associados a objetos e à relação entre eles, fazendo desta linguagem uma das mais utilizadas na área da Inteligência Artificial e *Non-Numerical Programming* [1].

Neste trabalho, pretende-se que seja 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 pela realização de atos médicos. As entidades que serão objeto de estudo são: **Instituições de Saúde**, **Utentes**, **Cuidados prestados** e **Atos médicos**.

Motivações e Objetivos

A realização deste projeto tem como objetivo particular a familiarização com o conceito de programação em lógica, colocando em prática os conhecimentos lecionados na UC.

Será também possível explorar e adquirir novos conhecimentos sobre a utilização especifica da linguagem Prolog e das vantagens que, no futuro, este paradigma de programação virá a ter.

Apresentamos como fator de motivação o facto de ser possível consolidar conceitos e noções relacionados com outras áreas, como a Lógica Proposicional e a utilização de Cláusulas de Horn.

Pretendemos que o sistema de representação de conhecimento e raciocínio tenha uma perspetiva simplificada da realidade, mas no qual possamos qualificar e caracterizar uma Base de Conhecimento consistente e capaz de responder a problemas relacionados com a área da prestação de cuidados de saúde.

É imprescindível que, da nossa parte, se realize de forma adequada a representação do conhecimento e, para além disto, que a construção dos predicados para a resolução dos problemas possam dar resposta a todos as funcionalidades obrigatórias e a outras que consideramos valorativas.

1. Base de Conhecimento

Neste capítulo são apresentadas todas as etapas de resolução das tarefas propostas. Numa primeira fase é explicitada a análise conceptual do problema e as respetivas interações e a definição da Base de Conhecimento.

1.1 Análise conceptual do problema

Depois de feita uma análise ao contexto do problema, foi elaborado um esquema conceptual representativo do mesmo.

A partir do enunciado do exercício prático foram identificadas três entidades, nomeadamente, *Utente*, *Cuidado prestado* e *Ato médico*. Por forma a obter uma melhor consistência da Base de Conhecimento, foi também considerada a entidade *Instituição* de saúde.

Deste modo, é possível definir um esquema representativo do problema:

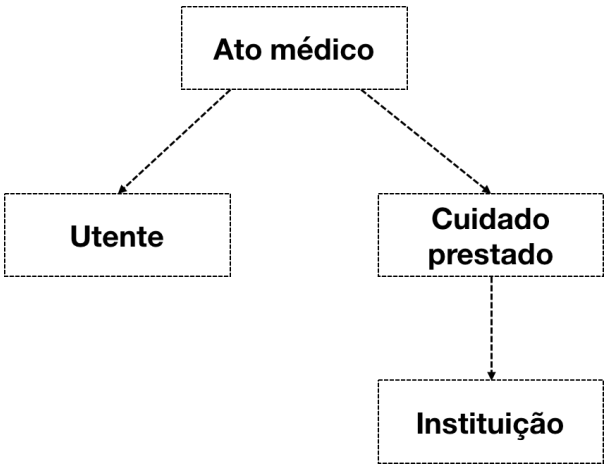

Figura 1.1: Mapa conceptual representativo do problema

1.2 Entidades

Nesta secção são apresentadas e caracterizadas as entidades identificadas anteriormente.

1.2.1 Utente

Em contextos reais, o utente de um sistema de saúde é caracterizado por fatores como o nome, a idade, o sexo, a morada, o número de utente, o número de contribuinte, o número do cartão de cidadão, o seu estado civil ou até o seu tipo de sangue. Contudo, para simplificar e a dar resposta aos requisitos base e complementares, apenas são requeridos o identificador de utente (*#IdUt*), o nome, a idade e a morada. Qualquer parâmetro extra poderia ser considerado para, num futuro, serem utilizados na produção de novos predicados.

utente(#IdUt,Nome,Idade,Morada)

A representação destas informações que definem um utente, é realizada através da declaração de factos. Cada utente é identificado por um identificador único, por forma a evitar inconformidade e ambiguidade de informação.

Assim sendo, todos os utentes existentes na nossa Base de Conhecimento são:

```
utente( 1,diogo,21,braga ).
utente( 2,rui,20,braga ).
utente( 3,esm,21,prado ).
utente( 4,miguel,22,viana ).
utente( 5,joao,26,guimaraes ).
utente( 6,lisandra,25, fafe ).
utente( 7,paulo,24,braganca ).
```

1.2.2 Instituição

Uma instituição de saúde pode ser caracterizada por várias informações além da sua designação, no entanto, para o desenvolvimento deste projeto, a designação mostrou-se mais do que suficiente.

instituicao(Designação)

A representação da informação relativa às instituições está mais uma vez feita através de factos, tais como:

```
instituicao( hpbraga ).
instituicao( hsjoao ).
instituicao( hviana ).
instituicao( hporto ).
instituicao( hfaro ).
```

1.2.3 Cuidado Prestado

A identificação de um cuidado é feita através de um identificador de serviço (*#IdServ*), de uma descrição, da instituição prestadora e da cidade.

Para garantir a consistência desta Base de Conhecimento, mais uma vez, o *#IdServ* terá de ser único.

cuidado(#IdServ,Descrição,Instituição,Cidade)

Tal como as anteriores entidades, esta vai ser também representado por factos, sendo que aqui se estabelece uma relação com a instituição descrita em cima.

Um cuidado, como referido anteriormente, é caracterizado pela sua descrição (um nome), que deverá ser elucidativa para uma fácil compreensão, como podemos conferir nos exemplos abaixo.

Os cuidados inseridos na Base de Conhecimento são:

```
cuidado( 1,analises,hpbraga,braga ).
cuidado( 2,tac,hsjoao,porto ).
cuidado( 3,nascimento,hpbraga,braga ).
cuidado( 4,febre,hviana,hviana ).
cuidado( 5,dar-sangue,hpbraga,braga ).
cuidado( 6,raioX,hporto,porto ).
cuidado( 7,consulta,hporto,porto ).
cuidado( 8,nascimento,hfaro,faro ).
cuidado( 9,ecografia,hfaro,faro ).
cuidado( 10,quimioterapia,hsjoao,porto ).
```

1.2.4 Ato médico

Um ato médico é caracterizado por uma data, o identificador do utente (*#IdUt*) a quem se refere, o identificador do cuidado realizado (*#IdServ*) e o custo associado.

Para a implementação das datas dos atos médicos, por uma questão de conseguir satisfazer os requisitos do enunciado, foi considerado um predicado auxiliar `data` que segue a seguinte nomenclatura:

data(dia,mes,ano)

e o predicado `ato` vem, então, como:

ato(Data,#IdUt,#IdServ,Custo)

Os factos que representam um ato médico foram escritos da seguinte forma:

```
ato( data( 1,2,1996 ),3,3,10 ).
ato( data( 15,3,2017 ),1,2,15 ).
ato( data( 17,4,1997 ),4,4,5 ).
ato( data( 15,3,2007 ),1,5,0 ).
ato( data( 15,3,2007 ),2,5,0 ).
ato( data( 15,3,2007 ),3,2,0 ).
ato( data( 16,3,2017 ),5,6,12 ).
ato( data( 16,3,2007 ),6,9,20 ).
ato( data( 16,3,2007 ),3,1,40 ).
```

1.3 Integridade da Base de Conhecimento

Para que a Base de Conhecimento não seja ambígua e esteja, de certa forma, de acordo com a realidade que pretendemos representar, é necessário implementar algum tipo de mecanismo que nos garanta a sua integridade.

Assim sendo, ao longo do desenvolvimento deste trabalho fomos dando uso ao conceito de invariante. Estes permitem-nos controlar em especifico a inserção e a

remoção de informações na nossa Base de Conhecimento. Desta forma conseguimos garantir que esta é consistente em termos lógicos.

Os invariantes, em Prolog, são representados da seguinte forma:

- $+$Termo :: Premissas.

  ou

- $-$Termo :: Premissas.

O primeiro é utilizado quando queremos inserir alguma informação. O segundo é utilizado quando precisamos de fazer a remoção de alguma informação da Base de Conhecimento.

Existem já definidos, em Prolog, predicados que nos permitem adicionar e remover factos da Base de Conhecimento. Estes são o `assert` e o `retract`, respetivamente. No entanto, a utilização destes predicados, exclusivamente, não garante consistência, uma vez que os invariantes definidos não são testados por si só e, assim sendo, é importante a criação de predicados auxiliares que nos tragam garantias que a inserção e a remoção de conhecimento não alterem a consistência da Base de Conhecimento. Estes predicados auxiliares serão abordados em secções posteriores deste relatório.

2. Implementação de predicados

Neste secção do relatório são descritos e exemplificados todos os predicados que foram definidos para manter a consistência e fiabilidade da Base de Conhecimento.

2.1 Modificações do conhecimento

O predicado `solucoes` trata-se simplesmente de uma implementação que tem a mesma funcionalidade que o predicado `findall`, já existente no Prolog.

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

Este é usado para obter todos os resultados que satisfazem uma dada condição. Em `Z` está a lista de resultados de `X` que satisfazem as condições em `Y`.

Tem-se também o predicado `testa` que recebe como argumento uma lista de invariantes e percorre toda essa lista testando cada um dos seus elementos, conjugando-os. Se algum falhar, o predicado `testa` falha no seu todo.

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

Os predicados associados aos processos de *evolução* e *involução* da Base de Conhecimento foram implementados tendo em conta a seguinte metodologia:

1. Recolher todos os invariantes associados ao termo pretendido;
2. Tomar a ação pretendida (inserir ou remover conhecimento);
3. Verificar se a inserção/remoção do conhecimento provocou inconsistência na Base de Conhecimento.

Para a inserção de conhecimento, deve ser utilizado o predicado `evolução`. Este predicado determina se a inserção provoca quebra dos invariantes associados ao termo em questão e, consequentemente, a desagregação da consistência da Base de Conhecimento. Se tal não acontecer, i.e., se o termo adicionado não quebrar os invariantes, é mantido na Base de Conhecimento. Caso contrário, o processo é revertido e o termo removido, mantendo o estado inicial das informações do sistema.

2.1.1 Inserção de Conhecimento

O predicado `evolução` utiliza, então, os predicados `soluções`, `inserção` e `testa`.

```
evolucao( Termo ) :-
    solucoes( INV,+Termo::INV,LINV ),
    insercao( Termo ),
    testa( LINV ).
```

Este realiza a inserção, por via do `assert`, do termo que se pretende adicionar. No caso ocorrer falha do teste dos invariantes no predicado `evolucao`, o mecanismo de *backtracking* do Prolog, levará a que o predicado `insercao` se desenrole pela sua segunda cláusula, que remove o termo adicionado através do `retract`. Assim se consegue manter a consistência da Base de Conhecimento.

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

2.2 Remoção de Conhecimento

Para a remoção de conhecimento, deve ser utilizado o predicado designado de `involução`. Este predicado tem em conta todos os invariantes associados ao termo que se pretende remover, determinando, posteriormente, se a sua remoção provoca a quebra desses invariantes. Além disso, e antes da "tentativa" de remoção, é verificado se de facto o termo a remover existe ou não na Base de Conhecimento. Se o termo não se verificar, o processo quebra sem sequer haver o teste dos invariantes. Caso contrário, o processo segue a metodologia já descrita.

Para a correta implementação deste predicado evolução, são novamente utilizados os predicados `solucoes`, `insercao` e `testa`.

```
involucao( Termo ) :-
    solucoes( INV,-Termo::INV,LINV ),
    Termo,
    remocao( Termo ),
    testa( LINV ).
```

Tal como no predicado relativo à inserção, também o predicado da remoção tem que voltar a inserir o termo caso os invariantes sejam quebrados. Vem então a seguinte definição:

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

2.3 Invariantes

Para garantir a veracidade e consistência das informações contidas na Base de Conhecimento desenvolvida, foi necessária a implementação de alguns invariantes associados aos predicados `utente`, `cuidado` e `ato`. Para uma melhor organização, estes invariantes foram divididos em dois tipos: invariantes estruturais e invariantes referenciais.

2.3.1 Invariantes Estruturais

Os invariantes estruturais são, tal como o nome indica, responsáveis por manter a estrutura do conhecimento existente, proibindo a inserção de conhecimento duplicado.

Utente

Não é possível existir na Base de Conhecimento mais do que um utente com o mesmo *#IdUt*. O identificador de um utente é associado inequivocamente a apenas um utente.

```
+utente( ID,N,I,M ) :: ( solucoes( ID, utente( ID,_,_,_ ),S ),
                         comprimento( S,L ),
                         L == 1 ).
```

Cuidado prestado

Não é permitida a inserção de diferentes cuidados associados a um mesmo *#Id-Serv*. Este identificador é único para cada tipo de cuidado prestado. Além disso,os cuidados inseridos estão necessariamente associados a uma instituição já registada no sistema.

```
+cuidado( ID,D,I,C ) :: ( solucoes( ID,cuidado( ID,_,_,_ ),S),
                          comprimento( S,L ),
                          instituicao( I ),
                          L == 1 ).
```

Instituição

Não é possível adicionar uma instituição que já esteja representada na Base de Conhecimento.

```
+instituicao( I ) :: ( solucoes( I,instituicao( I ),S ),
                       comprimento( S,L ),
                       L == 1 ).
```

Ato

Para ser possível adicionar um ato à Base de Conhecimento, é necessário garantir que o utente associado ao ato esteja registado no sistema.

```
+ato( D,IDU,IDS,C ) :: ( solucoes(IDU,utente(IDU,_,_,_ ),S ),
                         comprimento( S,L ),
                         L == 1 ).
```

De forma semelhante, é necessário garantir que o cuidado prestado seja também valido, i.e., que exista na Base de Conhecimento.

```
+ato( D,IDU,IDS,C ) :: ( solucoes(IDS,cuidado(IDS,_,_,_),S ),
                         comprimento( S,L ),
                         L == 1 ).
```

2.3.2 Invariantes Referenciais

Os invariantes referenciais evitam que as regras lógicas associadas ao domínio de conhecimento sejam quebradas, permitindo manter a consistência dos dados.

Utente

Para ser possível remover um utente do sistema, é preciso garantir que os seus dados não estão associados a nenhum ato médico. Se estiver associado a um ou mais atos médicos, a sua remoção é proibida, pois leva à inconsistência dos dados.

```
-utente( ID,N,I,M ) :: ( solucoes( ID,ato( _,ID,_,_ ),S ),
                         comprimento( S,L ),
                         L == 0 ).
```

Cuidado prestado

Pelo mesmo motivo que deve ser controlada a remoção de utentes, também só será possível remover um cuidado se este não estiver associado a nenhum ato médico. De outro modo, a informação relativa aos atos médicos ficaria inconsistente.

```
-cuidado( ID,D,I,C ) :: ( solucoes( ID,ato(_,_,ID,_),S ),
                          comprimento( S,L ),
                          L == 0 ).
```

Instituição

Não é possível a remoção de uma instituição do sistema se ela estiver associada a algum cuidado prestado. Os cuidados prestado devem estar sempre associados à instituição que os desempenha, tendo esta que existir no sistema.

```
-instituicao( Nome ) :: ( solucoes( I,cuidado(_,_,I,_),S ),
                          comprimento( S,L ),
                          L == 0 ).
```

2.4 Predicados auxiliares

Nesta secção são apresentados alguns predicados que foram úteis para a implementação das funcionalidades do sistema. A definição explícita deles pode ser encontrada no anexo A deste relatório.

2.4.1 Concatenação de duas listas

O predicado `concat` tem como objetivo a concatenação de duas listas numa lista resultado, conforme o exemplo.

Lista 1, Lista 2, Lista resultado $\rightarrow \{\mathbb{V}, \mathbb{F}\}$

```
concat( [a,b,c],[d,c],[a,b,c,d,c] ).
```

2.4.2 Remoção de elementos repetidos numa lista

O predicado `tiraRepetidos` tem como objetivo a remoção de elementos repetidos numa lista, conforme o exemplo.

Lista, Lista resultado $\rightarrow \{\mathbb{V}, \mathbb{F}\}$

```
tiraRepetidos( [1,2,3,2,1],[1,2,3] ).
```

2.4.3 Remoção de um elemento de uma lista

O predicado `eliminaElementos` tem como objetivo a remoção de todas as ocorrências de um elemento numa lista, conforme o exemplo.

Elemento, Lista, Lista resultado $\rightarrow \{\mathbb{V}, \mathbb{F}\}$

```
eliminaElementos( 2,[1,3,2,5,6,2],[1,3,5,6] ).
```

2.4.4 Determinar a existência de um elemento numa lista

O predicado `pertence` tem como objetivo determinar a existência de um dado elemento numa lista, conformoe os exemplos.

Elemento, Lista $\rightarrow \{\mathbb{V}, \mathbb{F}\}$

```
?- pertence( 2,[1,2,3] ).
yes

?- pertence( 4,[1,2,3] ).
no
```

2.4.5 Determinar o somatório de uma lista de números

O predicado `somatorio` tem como objetivo determinar a soma de todos os elementos de uma lista de números.

Lista, Resultado $\rightarrow \{\mathbb{V}, \mathbb{F}\}$

```
somatorio( [1,2,15,-2],16 ).
```

2.4.6 Cálculo do último elementos de uma lista

O predicado `ultimo` tem como objetivo determinar qual é o último elemento de uma lista recebida como parâmetro.

Lista, Resultado $\rightarrow \{\mathbb{V}, \mathbb{F}\}$

```
ultimo( [a,b,c,d,1,4,f],f ).
```

2.4.7 Cálculo do máximo de uma lista de tuplos

O predicado `tuploMaximo` tem como objetivo determinar qual é o tuplo máximo de uma lista. É considerado o segundo elemento de cada tuplo, conforme o exemplo.

Lista, Resultado $\rightarrow \{\mathbb{V}, \mathbb{F}\}$

```
tuploMaximo( [(a,2),(c,45),(1,7),(4,46)],(4,46) ).
```

3. Funcionalidades obrigatórias

Neste trabalho foi elaborado um conjunto de funcionalidades requeridas no enunciado do exercício prático. Nesta secção são apresentados os predicados implementados.

3.1 Identificar os utentes por critérios de seleção

Pretende-se que seja possível dar um critério de pesquisa e encontrar todos os utentes que o satisfaçam. Decidiu-se que o critério de pesquisa pode estar relacionado com o:

- ***#IdUt***: um identificador de um utente a procurar;
- **Nome**: um nome de um utente a procurar;
- **Idade**: uma idade a procurar;
- **Morada**: uma morada a procurar.

Para tal foi criado o predicado `pesquisaUtentes`:

Parâmetro a procurar, Critério de pesquisa, Resultado $\rightarrow \{\mathbb{V}, \mathbb{F}\}$

```
pesquisaUtentes( id,P,L ) :-
    findall( utente( P,X,Y,Z ),utente( P,X,Y,Z ),L ).
pesquisaUtentes( nome,P,L ) :-
    findall( utente(X,P,Y,Z ),utente( X,P,Y,Z ),L ).
pesquisaUtentes( idade,P,L ) :-
    findall( utente(X,Y,P,Z ),utente( X,Y,P,Z ),L ).
pesquisaUtentes( morada,P,L ) :-
    findall( utente(X,Y,Z,P ),utente( X,Y,Z,P ),L ).
```

3.2 Identificar as instituições prestadoras de cuidados de saúde

Pretende-se obter uma lista com todas as instituições de saúde registadas e com cuidados médicos associados. Para tal criou-se o predicado `listaInst`:

Lista Resultado $\rightarrow \{\mathbb{V}, \mathbb{F}\}$

```
listaInst( S ) :-
    findall( N,cuidado( _,_,N,_ ),L ),
    tiraRepetidos( L, S ).
```

Este predicado faz uso de `tiraRepetidos`, apresentado na secção 2.4.2 deste relatório.

3.3 Identificar os cuidados prestados por instituição ou cidade

Pretende-se obter uma lista com todos os cuidados registados numa dada cidade ou numa dada instituição. O predicado criado para o efeito, `listaCui`, tem como parâmetros a opção de pesquisa ("cidade" ou 'inst") e o que se pretende procurar:

Opção, Palavra a pesquisar, Resultado $\rightarrow \{\mathbb{V}, \mathbb{F}\}$

```
listaCui( cidade, C, L) :-
    findall( (Y,Z),cuidado( X,Y,Z,C ),L).
listaCui( inst, I, L) :-
    findall( (Y,Z),cuidado( X,Y,I,Z ),L).
```

3.4 Identificar os utentes de uma instituição ou serviço

Pretende-se obter uma lista com todos os utentes com atos médicos registados numa dada instituição ou num dado serviço. O predicado criado para o efeito, `listaUtentes`, tem como parâmetros a opção de pesquisa ("cuid" ou 'inst") e o que se pretende procurar, que deve ser a descrição do cuidado prestado ou da instituição:

Opção, Palavra a pesquisar, Resultado $\rightarrow \{\mathbb{V}, \mathbb{F}\}$

```
listaUtentes( inst,I,L ) :-
    procuraCui( inst,I,Temp ),
    procuraAtos( Temp,Temp2 ),
    procuraUtentes( Temp2,L ).
listaUtentes( cuid,C,L ) :-
    procuraCui( cuid,C,Temp ),
    procuraAtos( Temp,Temp2 ),
    procuraUtentes( Temp2,L ).
```

Este predicado faz uso de outros três: `procuraCui`, `procuraAtos` e `procuraUtentes`. A especificação destes predicados pode ser consultada no anexo A.

O primeiro tem uma lista resultante com os identificadores dos cuidados prestados (*#IdServ*) da instituição especificada ou da descrição de cuidado médico especificado:

Opção, Palavra a pesquisar, Lista resultado $\rightarrow \{\mathbb{V}, \mathbb{F}\}$

O segundo tem uma lista resultante com todos os identificadores de utente (*#IdUt*) com atos registados para cada um dos serviços na lista de *#IdServ* que tem como parâmetro:

Lista de #IdServ, Lista resultado $\rightarrow \{\mathbb{V}, \mathbb{F}\}$

O terceiro predicado tem uma lista resultante com tuplos de (*#IdUt*, Nome) de cada um dos utentes na lista de *#IdUt* que tem como parâmetro:

Lista de #IdUt, Lista resultado $\rightarrow \{\mathbb{V}, \mathbb{F}\}$

3.5 Identificar os atos médicos realizados por utente, instituição ou serviço

Pretende-se obter uma lista com todos os atos médicos registados por utente, instituição ou serviço. O predicado criado para o efeito, `listaAtosMed`, tem como parâmetros a opção de pesquisa ("utente", "cuid" ou 'inst") e o que se pretende procurar. Caso sejam especificado um utente ou um cuidado, deve ser passado como parâmetro o *#IdUt* ou o *#IdServ*.

Opção, Palavra a pesquisar, Resultado $\rightarrow \{\mathbb{V}, \mathbb{F}\}$

```
listaAtosMed( utente,IDU,L ) :-
    findall( ato( D,IDU,IDC,C ),ato( D,IDU,IDC,C ),Temp ),
    tiraRepetidos( Temp,L ).
listaAtosMed( cuid,IDC,L ) :-
    findall( ato( D,IDU,IDC,C ),ato( D,IDU,IDC,C ),Temp ),
    tiraRepetidos( Temp,L ).
listaAtosMed( inst,I,L ) :-
    procuraCui( inst,I,Temp ),
    listarAtos( Temp,L ).
```

A lista de atos médicos por instituição usa como predicados auxiliares o predicado `procuraCui` (apresentado anteriormente) e o predicado `listarAtos` (ver no anexo A). Este último tem uma lista resultante com todos os atos médicos que estão associados a cada um dos cuidados prestados na lista de *#IdServ*.

Lista de #IdServ, Lista resultado $\rightarrow \{\mathbb{V}, \mathbb{F}\}$

3.6 Determinar todas as instituições ou serviços a que um utente já recorreu

Pretende-se obter uma lista com as instituições ou cuidados a que um utente já recorreu. O predicado criado para o efeito, `histUtente`, tem como parâmetros a opção de pesquisa ("inst" ou "cuid") e o identificador que se pretende procurar

(*#IdUt* ou *#IdServ*):

Opção, Identificador a pesquisar, Resultado $\rightarrow \{\mathbb{V}, \mathbb{F}\}$

```
histUtente( inst,ID,L ) :-
    atosCuidados( ID,Temp ),
    cuidadosInst( Temp,L ).
histUtente( cuid,ID,L) :-
    findall( IDS,ato( _,ID,IDS,_ ),Temp ),
    tiraRepetidos( Temp,L ).
```

A lista de instituições a que o utente já recorreu usa como predicados auxiliares o predicado `atosCuidados` (explicado anteriormente) e o predicado `cuidadosInst` (ver no anexo A). O primeiro predicado tem uma lista resultante com todos os identificadores de serviço (*#IdServ*) a que o utente já se sujeitou, que tem como parâmetro:

#IdUtente, Lista resultado $\rightarrow \{\mathbb{V}, \mathbb{F}\}$

O segundo predicado tem uma lista resultante com todas as instituições de uma lista de identificadores de serviços que tem como parâmetro:

Lista de #IdServ, Lista resultado $\rightarrow \{\mathbb{V}, \mathbb{F}\}$

3.7 Calcular o custo total dos atos médicos por utente, serviço, instituição ou data

O que se pretende é calcular o valor total gasto em atos médicos por um dado utente, num dado serviço, numa dada instituição ou numa determinada data.

Como valorização desta tarefa, decidimos estender a opção da procura por uma data à possibilidade de especificar um mês, um ano ou ambos. Especificando um mês será calculado o total gasto nesse mês, em todos os anos. Se for especificado um ano será calculado o total gasto em todos os meses desse ano. Por fim, se for especificado um mês e um ano será calculado o total gasto apenas no mês desse ano.

Para suportar esta funcionalidade foi criado o predicado `custo` que pode receber como opções:

- **u**: para calcular o total gasto pelo utente *#IdUt* (`IDU`);
- **c**: para calcular o valor gasto em todos os atos médicos associados ao cuidado prestado *#IdServ* (`IDC`);
- **inst**: para calcular o total gasto na instituição especificada;
- **d**: para calcular o valor gasto em todos os atos médicos registados na data especificada (`data(dia,mes,ano)`);

- **m**: para calcular o valor gasto em todos os atos médicos registados no mês especificado (de todos os anos);
- **a**: para calcular o valor gasto em todos os atos médicos registados no ano especificado;
- **ma**: para calcular o valor gasto em todos os dias do mês do ano especificados.

Opção, Palavra a pesquisar, Resultado $\rightarrow \{\mathbb{V}, \mathbb{F}\}$

```
custo(u,IDU,R) :-
    findall( C,ato( _,IDU,_,C ),L ),
    somatorio( L,R ).
custo(c,IDC,R) :-
    findall( C,ato( _,_,IDC,C ),L ),
    somatorio( L,R ).
custo( inst,I,R ) :-
    findall( X,cuidado( X,_,I,_ ),L ),
    custoInstituicao(L,R).
custo( d,Data,R ) :-
    findall( C,ato( Data,_,_,C ),L ),
    somatorio( L,R ).
custo( m,Mes,R ) :-
    findall( C,(ato( D,_,_,C ),mes( D,Mes ) ),L ),
    somatorio( L,R ).
custo( a,Ano,R ) :-
    findall( C,(ato( D,_,_,C ),ano( D,Ano ) ),L ),
    somatorio( L,R ).
custo( ma,Mes,Ano,R ) :-
    findall( C,(ato( D,_,_,C ),mes( D,Mes ),ano( D,Ano ) ),L
    ↪ ),
    somatorio( L,R ).
```

Este predicado faz uso do predicado `somatorio`, especificado em 2.4.5, e do predicado `custoInstituicao` (ver no anexo A), que calcula o valor total gasto em atos médicos de cada um dos cuidados prestados da lista de *#IdServ*, recebida como parâmetro e calculada no predicado `custo`.

4. Funcionalidades complementares

Tendo elaborado todos os predicados capazes de responder aos requisitos mínimos, foram ainda implementados alguns predicados que permitem extrair outras informações úteis da Base de Conhecimento.

4.1 Identificar as instituições prestadoras de um determinado serviço

O predicado `instituicaoServico` foi criado para que, a partir da descrição de um cuidado prestado, fosse possível determinar a lista com o nome das instituições hospitalares que desempenham esse serviço.

Descrição do cuidado, Lista resultado $\rightarrow \{\mathbb{V}, \mathbb{F}\}$

```
instituicoesServico( S,L ) :-
    findall( I,cuidado(_,S,I,_),L ).
```

O predicado tem no seu primeiro parâmetro a descrição do cuidado prestado a procurar e no segundo parâmetro a lista resultado com as instituições que o disponibilizam. O seu procedimento é feito recorrendo ao `findall`, que, para todos os predicados `cuidado` existentes na Base de Conhecimento cuja descrição unifique com a descrição passada como parâmetro, guardará o campo correspondente ao nome da instituição numa lista.

4.2 Determinar qual o serviço médico mais recorrido pelos utentes

O predicado `servicoMaisUsado` determina qual o serviço mais solicitado num determinado ano, devolvendo a descrição do serviço e o número de ocorrências registadas.

Ano, Tuplo máximo $\rightarrow \{\mathbb{V}, \mathbb{F}\}$

```
servicoMaisUsado( A,(Desc,N) ) :-
    findall( (ID,NA),(ato( D,_,ID,_ ),ano( D,A ),quantos(
    ↪  A,ID,NA ) ),L ),
```

```
    tuploMaximo( L,(IDS,N) ),
    findall( Descricao,cuidado( IDS,Descricao,_,_ ),Temp ),
    ultimo( Temp,Desc ).
```

O predicado começa por calcular uma lista de pares com (*#IdServ*,Número de ocorrências), através de um `findall`, percorrendo todos os atos médicos existentes na base de conhecimento e considerando apenas aqueles que decorreram no ano especificado no primeiro argumento. O número de ocorrências é calculado através do predicado auxiliar `quantos`.

Ano, #IdServ, Número de ocorrências resultado $\rightarrow \{\mathbb{V}, \mathbb{F}\}$

```
quantos( A,ID,NA ) :- findall( ID,(ato( D,_,ID,_ ),ano( D,A )
↪  ),L ),
                      comprimento( L,NA ).
```

Tendo a lista gerada, foi usado o predicado `tuploMaximo` para determinar qual o tuplo (*#IdServ*, Número de ocorrências) máximo. Este predicado foi apresentado na secção 2.4.7 deste relatorio.

Tendo o *#IdServ* do cuidado mais prestado no ano em questão, bastou apenas saber qual a descrição deste serviço. Este passo é realizado por ser mais fácil de interpretar a descrição de um cuidado médico do que simplesmente o seu identificador na Base de Conhecimento.

O par do segundo argumento do predicado `servicoMaisUsado` é então do tipo (Descrição de cuidado prestado, Número de ocorrências).

4.3 Lista das datas em que um utente esteve numa instituição

O predicado `datasVisitasInst` é usado para determinar a lista das datas em que um determinado utente esteve numa instituição indicada.

```
datasVisitaInst( N,I,L ) :-
    findall( D,( utente( IDU,N,_,_ ),ato( D,IDU,IDS,_
    ↪  ),cuidado( IDS,_,I,_ ) ),L ).
```

O predicado recebe como parâmetros o nome do utente, a descrição da instituição a pesquisar e a lista resultante com as datas em que o utente visitou essa instituição.

Para determinar as referidas datas, é feito um `findall`, onde, a partir do nome do utente, se determina o seu *#IdUt* no sistema de informação e, para todos os atos médicos associados a esse identificador e associados a um cuidado prestado na instituição hospitalar indicada, são guardadas as datas.

5. Exemplos práticos e Análise de Resultados

Nesta secção é feita uma breve análise aos resultados obtidos a algumas questões feitas à Base de Conhecimento. Procurou-se fazê-lo para cada uma das funcionalidades implementadas.

Para a demonstração+ é utilizada como Base de Conhecimento um pequeno conjunto de factos, que podem ser consultados no código fonte do programa (no anexo A) e também já apresentados na secção 1.2 deste relatório.

Utilizaremos uma máquina abstrata de *Warren* (*WAM*) como ambiente de execução do programa em Prolog, recorrendo para isso ao interpretador, compilador e emulador *WAN* fornecedo pelo *SICStus Prolog* [3].

5.1 Funcionalidades Obrigatórias

I. **Objetivo:** Registar utentes, cuidados prestados e atos médicos.

   i. **Questão:** Inserir a utente *ana* com *#IdUt* 8, 24 anos e residente em Faro.
   Resposta: yes

```
?- evolucao( utente( 8,ana,24,faro ) ).
yes

?- listing( utente ).
utente(1, diogo, 21, braga).
utente(2, rui, 20, braga).
utente(3, esm, 21, prado).
utente(4, miguel, 22, viana).
utente(5, joao, 26, guimaraes).
utente(6, lisandra, 25, fafe).
utente(7, paulo, 24, braganca).
utente(8, ana, 24, faro).
```

   ii. **Questão:** Inserir o utente *paulo* com *#IdUt* 5, 30 anos e residente em Lisboa.
   Resposta: no
   Justificação: O utente já existe.

```
?- evolucao( utente( 5,paulo,30,lisboa ) ).
no

?- listing(utente).
utente(1, diogo, 21, braga).
utente(2, rui, 20, braga).
utente(3, esm, 21, prado).
```

```
utente(4, miguel, 22, viana).
utente(5, joao, 26, guimaraes).
utente(6, lisandra, 25, fafe).
utente(7, paulo, 24, braganca).
utente(8, ana, 24, faro).
yes
```

iii. **Questão:** Inserir o cuidado consulta com *#IdServ* 11, na instituição *hsjoao*, no Porto.
Resposta: yes

```
?- evolucao( cuidado( 11,consulta,hsjoao,porto ) ).
yes

?- listing( cuidado ).
cuidado(1, analises, hpbraga, braga).
cuidado(2, tac, hsjoao, porto).
cuidado(3, nascimento, hpbraga, braga).
cuidado(4, febre, hviana, hviana).
cuidado(5, dar-sangue, hpbraga, braga).
cuidado(6, raioX, hporto, porto).
cuidado(7, consulta, hporto, porto).
cuidado(8, nascimento, hfaro, faro).
cuidado(9, ecografia, hfaro, faro).
cuidado(10, quimioterapia, hsjoao, porto).
cuidado(11, consulta, hsjoao, porto).
yes
```

iv. **Questão:** Inserir o cuidado *consulta* com *#IdServ* 5, na instituição *hsjoao*, no Porto.
Resposta: no
Justificação: O *#IdServ* já existe.

```
?- evolucao( cuidado( 5,consulta,hsjoao,porto ) ).
no

?- listing( cuidado ).
cuidado(1, analises, hpbraga, braga).
cuidado(2, tac, hsjoao, porto).
cuidado(3, nascimento, hpbraga, braga).
cuidado(4, febre, hviana, hviana).
cuidado(5, dar-sangue, hpbraga, braga).
cuidado(6, raioX, hporto, porto).
cuidado(7, consulta, hporto, porto).
cuidado(8, nascimento, hfaro, faro).
cuidado(9, ecografia, hfaro, faro).
cuidado(10, quimioterapia, hsjoao, porto).
cuidado(11, consulta, hsjoao, porto).

yes
```

v. **Questão:** Inserir o ato médico ocorrido no dia *18-03-2017*, pelo utente com *#IdUt* 5, *#IdServ* 4 e um custo de 5€.
Resposta: yes

```
?- evolucao( ato( data( 18,3,2017 ),5,4,5 ) ).
yes

?- listing( ato ).
ato(data(1,2,1996), 3, 3, 10).
ato(data(15,3,2017), 1, 2, 15).
ato(data(17,4,1997), 4, 4, 5).
```

```
ato(data(15,3,2007), 1, 5, 0).
ato(data(15,3,2007), 2, 5, 0).
ato(data(15,3,2007), 3, 2, 0).
ato(data(16,3,2017), 5, 6, 12).
ato(data(16,3,2007), 6, 9, 20).
ato(data(16,3,2007), 3, 1, 40).
ato(data(18,3,2017), 5, 4, 5).

yes
```

vi. **Questão:** Inserir o ato médico ocorrido no dia *18-03-2017*, pelo utente com *#IdUt* 9, *#IdServ* 12 e um custo de 5€.
Resposta: no
Justificação: Nem o utente nem o cuidado prestado existem.

```
?- evolucao( ato( data( 18,3,2017 ),9,12,5 ) ).
no

?- listing( ato ).
ato(data(1,2,1996), 3, 3, 10).
ato(data(15,3,2017), 1, 2, 15).
ato(data(17,4,1997), 4, 4, 5).
ato(data(15,3,2007), 1, 5, 0).
ato(data(15,3,2007), 2, 5, 0).
ato(data(15,3,2007), 3, 2, 0).
ato(data(16,3,2017), 5, 6, 12).
ato(data(16,3,2007), 6, 9, 20).
ato(data(16,3,2007), 3, 1, 40).

yes
```

II. **Objetivo:** Identificar os utentes por critérios de seleção.
Questão: Quais os utentes com 21 anos de idade?
Resposta: utente(1,diogo,21,braga) e utente(3,esm,21,prado).

```
?- pesquisaUtentes( idade,21,L ).
L = [utente(1,diogo,21,braga),utente(3,esm,21,prado)] ?
yes
```

III. **Objetivo:** Identificar as instituições prestadoras de cuidados de saúde.
Questão: Quais as instituições que prestam cuidados?
Resposta: hpbraga, hsjoao, hviana, hporto e hfaro.

```
?- listaInst( S ).
S = [hpbraga,hsjoao,hviana,hporto,hfaro] ?
yes
```

IV. **Objetivo:** Identificar os cuidados prestados por instituição/cidade.
Questão: Quais os cuidados de saúde prestados na cidade de Braga?
Resposta: (analises,hpbraga),(nascimento,hpbraga),(dar-sangue,hpbraga).

```
?- listaCui( cidade,braga,L ).
L = [(analises,hpbraga),(nascimento,hpbraga),
 ↪  (dar-sangue,hpbraga)] ?
yes
```

V. **Objetivo:** Identificar os utentes de uma instituição/serviço.
Questão: Quais os utentes que frequentam o *hpbraga*?
Resposta: (1,diogo), (2,rui) e (3,esm).

```
listaUtentes( inst,hpbraga,L ).
L = [(3,esm),(1,diogo),(2,rui)] ?
yes
```

VI. **Objetivo:** Identificar os atos médicos realizados, por utente/instituição/serviço.
Questão: Quais os atos médicos realizados relativos ao serviço de análises do *hpbraga*, cujo *#IdServ* é o 5?
Resposta: ato(data(15,3,2007),1,5,0) e o ato(data(15,3,2007),2,5,0).

```
?- listaAtosMed( cuid,5,L ).
L = [ato(data(15,3,2007),1,5,0),ato(data(15,3,2007),2,5,0)]
 ↪  ?
yes
```

VII. **Objetivo:** Determinar todas as instituições/serviços a que um utente já recorreu.
Questão: Quais os cuidados prestados a que o utente (1,diogo,21,braga) já recorreu?
Resposta: cuidado(2,tac,hsjoao,porto),cuidado(5,dar-sangue,hpbraga,braga).

```
?- histUtente( cuid,1,L ).
L = [2,5] ?
yes
```

VIII. **Objetivo:** Calcular o custo total dos atos médicos por utente, serviço, instituição ou data.
Questão: Qual o custo total dos atos médicos em *16-03-2007*?
Resposta: 60.

```
?- custo(d,data(16,3,2007),R).
R = 60 ?
yes
```

IX. **Objetivo:** Remover utentes, cuidados e atos médicos.

i. **Questão:** Remover o utente *paulo* com *#IdUt* 7, 24 anos e residente em Bragança.
Resposta: yes

```
?- involucao( utente( 7,paulo,24,braganca ) ).
yes

?- listing( utente ).
utente(1, diogo, 21, braga).
utente(2, rui, 20, braga).
utente(3, esm, 21, prado).
utente(4, miguel, 22, viana).
utente(5, joao, 26, guimaraes).
utente(6, lisandra, 25, fafe).
yes
```

ii. **Questão:**Remover o utente *diogo* com *#IdUt* 1, 21 anos e residente em Braga.
Resposta: no
Justificação: O utente tem atos médicos associados.

```
involucao( utente( 1,diogo,21,braga ) ).
no

?- listing( utente ).
utente(2, rui, 20, braga).
utente(3, esm, 21, prado).
utente(4, miguel, 22, viana).
utente(5, joao, 26, guimaraes).
utente(6, lisandra, 25, fafe).
utente(1, diogo, 21, braga).
yes
```

iii. **Questão:** Remover o cuidado *quimioterapia* com *#IdServ* 10, na instituição *hsjoao*, no Porto.
Resposta: yes

```
?- involucao( cuidado( 10,quimioterapia,hsjoao,porto )
↪  ).
yes

?- listing( cuidado ).
cuidado(1, analises, hpbraga, braga).
cuidado(2, tac, hsjoao, porto).
cuidado(3, nascimento, hpbraga, braga).
cuidado(4, febre, hviana, hviana).
cuidado(5, dar-sangue, hpbraga, braga).
cuidado(6, raioX, hporto, porto).
cuidado(7, consulta, hporto, porto).
cuidado(8, nascimento, hfaro, faro).
cuidado(9, ecografia, hfaro, faro).
yes
```

iv. **Questão:** Remover o cuidado *analises* com *#IdServ* 1, na instituição *hpbraga*, em Braga.
Resposta: no
Justificação: Existem atos médicos associados a este cuidado prestado.

```
?- involucao( cuidado( 1,analises,hpbraga,braga ) ).
no

?- listing( cuidado ).
cuidado(2, tac, hsjoao, porto).
cuidado(3, nascimento, hpbraga, braga).
cuidado(4, febre, hviana, hviana).
cuidado(5, dar-sangue, hpbraga, braga).
cuidado(6, raioX, hporto, porto).
cuidado(7, consulta, hporto, porto).
cuidado(8, nascimento, hfaro, faro).
cuidado(9, ecografia, hfaro, faro).
cuidado(1, analises, hpbraga, braga).
yes
```

v. **Questão:** Remover o ato médico ocorrido no dia *16-03-2007*, pelo utente com *#IdUt* 3, *#IdServ* 1 e custo de 40€.
Resposta: yes

```
?- involucao( ato( data( 16,3,2007 ),3,1,40 ) ).
yes

?- listing( ato ).
ato(data(1,2,1996), 3, 3, 10).
ato(data(15,3,2017), 1, 2, 15).
```

```
ato(data(17,4,1997), 4, 4, 5).
ato(data(15,3,2007), 1, 5, 0).
ato(data(15,3,2007), 2, 5, 0).
ato(data(15,3,2007), 3, 2, 0).
ato(data(16,3,2017), 5, 6, 12).
ato(data(16,3,2007), 6, 9, 20).
```

vi. **Questão:** Remover o ato médico ocorrido no dia *17-03-2007*, pelo utente com *#IdUt* 10, *#IdServ* 20 e custo de 10€.
Resposta: no
Justificação: O ato médico não existe.

```
?- involucao( ato( data( 17,3,2007 ),19,20,10 ) ).
no

| ?- listing( ato ).
ato(data(1,2,1996), 3, 3, 10).
ato(data(15,3,2017), 1, 2, 15).
ato(data(17,4,1997), 4, 4, 5).
ato(data(15,3,2007), 1, 5, 0).
ato(data(15,3,2007), 2, 5, 0).
ato(data(15,3,2007), 3, 2, 0).
ato(data(16,3,2017), 5, 6, 12).
ato(data(16,3,2007), 6, 9, 20).
yes
```

5.2 Funcionalidades Complementares

I. **Objetivo:** Identificar as instituições prestadoras de um determinado serviço.
Questão: Que instituições prestam o serviço *nascimento*?
Resposta: hpbraga e hfaro.

```
?- instituicoesServico(nascimento,L).
L = [hpbraga,hfaro] ?
yes
```

II. **Objetivo:** Determinar qual o serviço médico mais recorrido pelos utentes.
Questão: Qual o serviço médico mais recorrido no ano *2007*?
Resposta: dar-sangue.

```
?- servicoMaisUsado(2007,R).
R = (dar-sangue,2) ?
yes
```

III. **Objetivo:** Determinar as datas em que um utente esteve numa instituição
Questão: Em que datas o utente *esm* esteve no *hpbraga* ?
Resposta: 1-2-1996 e 16-3-2007.

```
?- datasVisitaInst(esm,hpbraga,L).
L = [data(1,2,1996),data(16,3,2007)] ?
yes
```

Conclusão e aspetos a melhorar

Tal como foi referido no capítulo *Motivação e Objetivos*, o principal objetivo deste trabalho recaiu sobre a utilização da linguagem de programação em lógica PROLOG. Aqui desenvolvemos a capacidade de representação de conhecimento, bem como o tratamento de problemas sobre uma base de conhecimento e a evolução da mesma.

Acreditamos termos implementado um sistema de representação de conhecimento que não só responde a todos os principais requisitos, como ainda apresenta um conjunto de funcionalidades adicionais que aproxima a nossa base de conhecimento a uma com mais semelhanças à realidade.

Devido às faculdades obtidas e desenvolvidas ao longo da realização deste trabalho prático, conseguimos gerir e organizar toda a informação e predicados relativos a estas de forma natural. Assim sendo, podemos referir que a zona do trabalho que nos causou mais dificuldade foi a forma de representação das datas, apesar de que com um pouco de estudo esta ter sido completamente superada.

Para finalizar, podemos afirmar que este projeto foi importante para sedimentar e perceber melhor alguns conceitos relacionados com a linguagem de programação em lógica. Além disto, fazendo uma análise de resultados, verificamos que o nosso sistema funciona sem qualquer tipo de problema, sendo que este produto final consegue corresponder a todos os requisitos do enunciado, assim como aos nossos requisitos pessoais enquanto grupo.

Em termos de aspetos a melhorar, denotamos que seria vantajoso elaborar um predicado `pesquisaUtente` com funcionalidades mais interessantes, como por exemplo “pesquisar todos os utentes com mais do que 20 anos”. Também o predicado `instituição` poderia ser melhorado acrescentando a cidade da instituição aos parâmetros, que está atualmente no predicado `cuidado`. Não procedemos a esta alteração por uma questão de tempo.

A. Código fonte

```
%---------------------------------- - - - - - - - - - -  -  -
% SIST. REPR. CONHECIMENTO E RACIOCINIO - MiEI/3

%---------------------------------- - - - - - - - - - -  -  -
% Resolução do exercício 1 (Trabalho prático)
t
%---------------------------------- - - - - - - - - - -  -  -
% SICStus PROLOG: Declaracoes iniciais

:- set_prolog_flag( discontiguous_warnings,off ).
:- set_prolog_flag( single_var_warnings,off ).
:- set_prolog_flag( unknown,fail ).

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

:- op( 900,xfy,'::' ).
:- dynamic utente/4.
:- dynamic ato/4.
:- dynamic instituicao/1.
:- dynamic cuidado/4.


%---------------------------------- - - - - - - - - - -  -  -
% Dados da base de conhecimento
utente( 1,diogo,21,braga ).
utente( 2,rui,20,braga ).
utente( 3,esm,21,prado ).
utente( 4,miguel,22,viana ).
utente( 5,joao,26,guimaraes ).
utente( 6,lisandra,25, fafe ).
utente( 7,paulo,24,braganca ).

instituicao( hpbraga ).
instituicao( hsjoao ).
instituicao( hviana ).
instituicao( hporto ).
instituicao( hfaro ).

cuidado( 1,analises,hpbraga,braga ).
cuidado( 2,tac,hsjoao,porto ).
cuidado( 3,nascimento,hpbraga,braga ).
cuidado( 4,febre,hviana,hviana ).
cuidado( 5,dar-sangue,hpbraga,braga ).
cuidado( 6,raioX,hporto,porto ).
cuidado( 7,consulta,hporto,porto ).
cuidado( 8,nascimento,hfaro,faro ).
cuidado( 9,ecografia,hfaro,faro ).
cuidado( 10,quimioterapia,hsjoao,porto ).
```

```
ato( data( 1,2,1996 ),3,3,10 ).
ato( data( 15,3,2017 ),1,2,15 ).
ato( data( 17,4,1997 ),4,4,5 ).
ato( data( 15,3,2007 ),1,5,0 ).
ato( data( 15,3,2007 ),2,5,0 ).
ato( data( 15,3,2007 ),3,2,0 ).
ato( data( 16,3,2017 ),5,6,12 ).
ato( data( 16,3,2007 ),6,9,20 ).
ato( data( 16,3,2007 ),3,1,40 ).


%-------------------------------- - - - - - - - - - - - - -
% Extensao dos predicados dia, mes e ano: Data, Dia/Mes/Ano ->
↪  {V,F}

dia( data( D,M,A ),D ).
mes( data( D,M,A ),M ).
ano( data( D,M,A ),A ).


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

% Invariante Estrutural: nao permitir a insercao de conhecimento
%                        repetido

+utente( ID,N,I,M ) :: ( solucoes( ID, utente( ID,_,_,_ ),S ),
                         comprimento( S,L ),
                         L == 1 ).


% Invariante Referencial: garantir a consistênica de conhecimento

% O Utente removido não pode ter atos associados
-utente( ID,N,I,M ) :: ( solucoes( ID,ato( _,ID,_,_ ),S ),
                         comprimento( S,L ),
                         L == 0 ).


%-------------------------------- - - - - - - - - - - - - -
% Extensao do predicado cuidado prestado:
% IdServ, Descrição, Instituição, Cidade -> {V,F}

% Invariante Estrutural: nao permitir a insercao de conhecimento
%                        repetido

+cuidado( ID,D,I,C ) :: ( solucoes( ID,cuidado( ID,_,_,_ ),S ),
                          comprimento( S,L ),
                          instituicao( I ),
                          L == 1 ).

% Invariante Referencial: garantir a consistênica de conhecimento

% O Cuidado removido não pode ter atos medicos associados
-cuidado( ID,D,I,C ) :: ( solucoes( ID,ato( _,_,ID,_ ),S ),
                          comprimento( S,L ),
                          L == 0 ).


%-------------------------------- - - - - - - - - - - - - -
% Extensao do predicado Instituição:
```

```
% Nome -> {V,F}

% Invariante Estrutural: nao permitir a insercao de conhecimento
%                        repetido

+instituicao( I ) :: ( solucoes( I,instituicao( I ),S ),
                       comprimento( S,L ),
                       L == 1 ).

% Invariante Referencial: garantir a consistênica de conhecimento

-instituicao( I ) :: ( solucoes( I,cuidado(_,_,I,_),S ),
                       comprimento( S,L ),
                       L == 0 ).


%--------------------------------- - - - - - - - - - -  -  -
% Extensao do predicado ato medico:
% Data, IdUtente, IdServico, Custo -> {V,F}

% Invariante Estrutural: nao permitir a insercao de conhecimento
%                        repetido

+ato( D,IDU,IDS,C ) :: ( solucoes( IDU,utente( IDU,_,_,_ ),S ),
                          comprimento( S,L ),
                          L == 1 ).
+ato( D,IDU,IDS,C ) :: ( solucoes( IDS,cuidado( IDS,_,_,_ ),S ),
                          comprimento( S,L ),
                          L == 1 ).

%--------------------------------- - - - - - - - - - -  -  -
% Identificar os utentes por critérios de seleção;

%--------------------------------- - - - - - - - - - -  -  -
% Extensao do predicado pesquisaUtentes:
% Opcao, Parametro, Lista -> {V,F}

pesquisaUtentes( id,P,L ) :-
    findall( utente( P,X,Y,Z ),utente( P,X,Y,Z ),L ).
pesquisaUtentes( nome,P,L ) :-
    findall( utente(X,P,Y,Z ),utente( X,P,Y,Z ),L ).
pesquisaUtentes( idade,P,L ) :-
    findall( utente(X,Y,P,Z ),utente( X,Y,P,Z ),L ).
pesquisaUtentes( morada,P,L ) :-
    findall( utente(X,Y,Z,P ),utente( X,Y,Z,P ),L ).


%--------------------------------- - - - - - - - - - -  -  -
% Identificar as instituições prestadoras de cuidados de saúde;

%--------------------------------- - - - - - - - - - -  -  -
% Extensao do predicado listaInst: Lista -> {V,F}
% Lista é a lista de todas as instituições a que estão associados
% cuidados medicos

listaInst( S ) :-
    findall( N,cuidado( _,_,N,_ ),L),
    tiraRepetidos( L, S ).


%--------------------------------- - - - - - - - - - -  -  -
% Identificar os cuidados prestados por instituição/cidade;
```

```
%-------------------------------- - - - - - - - - - -  -  -
% Extensao do predicado listaCui: Opcao, Parametro, Lista -> {V,F}
% Lista é a lista de todos os cuidados na cidade/instituição

listaCui( cidade, C, L) :-
    findall( (Y,Z),cuidado( X,Y,Z,C ),L).
listaCui( inst, I, L) :-
    findall( (Y,Z),cuidado( X,Y,I,Z ),L).


%-------------------------------- - - - - - - - - - -  -  -
% Identificar os utentes de uma instituição/servico

%-------------------------------- - - - - - - - - - -  -  -
% Extensao do predicado listaUtentes: Opcao, Parametro, Lista ->
↪  {V,F}
% Lista é a lista de todos os utentes com atos medicos na
% instituição/cuidado
% Parametro é a descrição da instituição/cuidado

listaUtentes( inst,I,L ) :-
    procuraCui( inst,I,Temp ),
    procuraAtos( Temp,Temp2 ),
    procuraUtentes( Temp2,L ).
listaUtentes( cuid,C,L ) :-
    procuraCui( cuid,C,Temp ),
    procuraAtos( Temp,Temp2 ),
    procuraUtentes( Temp2,L ).

%-------------------------------- - - - - - - - - - -  -  -
% Extensao do predicado procuraCui: Tipo, Parametro, Lista ->
↪  {V,F}
% Lista é a lista dos IdsServ da instituição/cuidado
% Parametro é a descrição da instituição/cuidado

procuraCui( inst,I,L ) :-
    findall( X,cuidado( X,_,I,_ ),Temp ),
    tiraRepetidos( Temp,L ).
procuraCui( cuid,D,L ) :-
    findall( X,cuidado( X,D,_,_ ),Temp ),
    tiraRepetidos( Temp,L ).

%-------------------------------- - - - - - - - - - -  -  -
% Extensao do predicado procuraAtos: ListaIdsServ, Lista -> {V,F}
% Lista é a lista dos IdsUt com serviços da ListaIdsServ

procuraAtos( [X],L ) :-
    findall( U,ato( _,U,X,_ ),L ).
procuraAtos( [H|T],L ) :-
    findall( U, ato( _,U,H,_ ),Temp ),
    procuraAtos( T,Temp2 ),
    concat( Temp,Temp2,Temp3 ),
    tiraRepetidos( Temp3,L ).

%-------------------------------- - - - - - - - - - -  -  -
% Extensao do predicado procuraUtentes: ListaIdsUt, Lista -> {V,F}
% Lista é a lista com (IdUt, Nome) dos utentes em ListaIdsUt

procuraUtentes( [X],L ) :-
    findall( ( X,N ),utente( X,N,_,_ ),L ).
procuraUtentes( [H|T],L ) :-
```

```
    findall( ( H,N ),utente( H,N,_,_ ),Temp ),
    procuraUtentes( T,Temp2 ),
    concat( Temp,Temp2,Temp3 ),
    tiraRepetidos( Temp3,L ).


%--------------------------------- - - - - - - - - - -  -  -
% Identificar os atos médicos realizados, por
↪  utente/instituição/serviço;

%--------------------------------- - - - - - - - - - -  -  -
% Extensao do predicado listaAtosMed: Opcao, Parametro, Lista ->
↪  {V,F}
% Lista é a lista dos atos medicos do utente/instituição/serviço

listaAtosMed( utente,IDU,L ) :-
    findall( ato( D,IDU,IDC,C ),ato( D,IDU,IDC,C ),Temp ),
    tiraRepetidos( Temp,L ).
listaAtosMed( cuid,IDC,L ) :-
    findall( ato( D,IDU,IDC,C ),ato( D,IDU,IDC,C ),Temp ),
    tiraRepetidos( Temp,L ).
listaAtosMed( inst,I,L ) :-
    procuraCui( inst,I,Temp ),
    listarAtos( Temp,L ).

%--------------------------------- - - - - - - - - - -  -  -
% Extensao do predicado listarAtos: ListaIdsServ, Lista -> {V,F}
% Lista é a lista dos atos medicos dos serviços em ListaIdsServ

listarAtos( [ID],L ) :-
    findall( ato( X,Y,ID,Z ),ato( X,Y,ID,Z ),L ).
listarAtos( [ID|T],L ) :-
    findall( ato( X,Y,ID,Z ),ato( X,Y,ID,Z ),Temp ),
    listarAtos( T,Temp2 ),
    concat( Temp,Temp2,Temp3 ),
    tiraRepetidos( Temp3,L ).


%--------------------------------- - - - - - - - - - -  -  -
% Determinar todas as instituições/serviços a que um utente já
↪  recorreu;

%--------------------------------- - - - - - - - - - -  -  -
% Extensao do predicado histUtente: Tipo, id Utente, Lista ->
↪  {V,F}
% Lista é a lista das instituições/serviços a que o utente
↪  recorreu

histUtente( inst,ID,L ) :-
    atosCuidados( ID,Temp ),
    cuidadosInst( Temp,L ).
histUtente( cuid,ID,L) :-
    findall( IDS,ato( _,ID,IDS,_ ),Temp ),
    tiraRepetidos( Temp,L ).

%--------------------------------- - - - - - - - - - -  -  -
% Extensao do predicado atosCuidados: IdUt, Lista -> {V,F}
% Lista é a lista com os IdServ a que o utente se sujeitou

atosCuidados( ID,L ) :-
    findall( IDS,ato( _,ID,IDS,_ ),Temp ),
    tiraRepetidos( Temp,L ).
```

```
%-------------------------------- - - - - - - - - - -  -  -
% Extensao do predicado cuidadosInst: ListaIdsServ, Lista -> {V,F}
% Lista é a lista das instituições dos IdsServ em ListaIdsServ

cuidadosInst( [IDC],L ) :-
    findall( I,cuidado( IDC,_,I,_ ),L ).

cuidadosInst( [IDC|T],L ) :-
    findall( I,cuidado( IDC,_,I,_ ),Temp ),
    cuidadosInst( T,Temp2 ),
    concat( Temp,Temp2,Temp3 ),
    tiraRepetidos( Temp3,L ).


%-------------------------------- - - - - - - - - - -  -  -
% Calcular o custo total dos atos medicos por
% utente/serviço/instituição/data;

%-------------------------------- - - - - - - - - - -  -  -
% Extensao do predicado custo: Tipo,Parametro,Retorno -> {V,F}
% Parametro : IDutente, IDcuidado, Instituicao, Data, Mes, Ano,
↪  Mes&Ano

custo(u,IDU,R) :-
    findall( C,ato( _,IDU,_,C ),L ),
    somatorio( L,R ).
custo(c,IDC,R) :-
    findall( C,ato( _,_,IDC,C ),L ),
    somatorio( L,R ).
custo( inst,I,R ) :-
    findall( X,cuidado( X,_,I,_ ),L ),
    custoInstituicao(L,R).
custo( d,Data,R ) :-
    findall( C,ato( Data,_,_,C ),L ),
    somatorio( L,R ).
custo( m,Mes,R ) :-
    findall( C,(ato( D,_,_,C ),mes( D,Mes ) ),L ),
    somatorio( L,R ).
custo( a,Ano,R ) :-
    findall( C,(ato( D,_,_,C ),ano( D,Ano ) ),L ),
    somatorio( L,R ).
custo( ma,Mes,Ano,R ) :-
    findall( C,(ato( D,_,_,C ),mes( D,Mes ),ano( D,Ano ) ),L ),
    somatorio( L,R ).

%-------------------------------- - - - - - - - - - -  -  -
% Extensao do predicado custoInst: ListaIdsServ, Custo -> {V,F}

custoInstituicao( [IDC],R ) :-
    findall( C,ato( _,_,IDC,C ),L ),
    somatorio( L,R ).

custoInstituicao( [IDC|T],R ) :-
    findall( C,ato( _,_,IDC,C ),L ),
    somatorio( L,Temp ),
    custoInstituicao( T,Temp2 ),
     R is Temp + Temp2.


%--------------- Funcionalidades Extra ---------------
```

```
%--------------------------------- - - - - - - - - - -  -  -
% Extensao do predicado instituicoesServico: Serviço, Lista ->
↪  {V,F}
% Lista é a lista das instituições que disponibilizam um serviço

instituicoesServico( S,L ) :-
    findall( I,cuidado(_,S,I,_),L ).


%--------------------------------- - - - - - - - - - -  -  -
% Extensao do predicado servicoMaisUsado:
% Ano, (Serviço, NOcorrencias) -> {V,F}

servicoMaisUsado( A,(Desc,N) ) :-
    findall( (ID,NA),( ato( D,_,ID,_ ),ano( D,A ),quantos( A,ID,NA
    ↪  ) ),L ),
    tuploMaximo( L,(IDS,N) ),
    findall( Descricao, cuidado( IDS,Descricao,_,_ ),Temp ),
    ultimo( Temp,Desc ).

%--------------------------------- - - - - - - - - - -  -  -
% Extensao do predicado quantos: Ano, IdServ, Numero -> {V,F}
% Número é o número de atos registado no Ano, para o serviço
↪  IdServ

quantos( A,ID,NA ) :- findall( ID, ( ato( D,_,ID,_ ),ano( D,A )
↪  ),L ),
                      comprimento( L,NA ).


%--------------------------------- - - - - - - - - - -  -  -
% Extensao do predicado datasVisitaInst: NomeUt, Inst, Lista ->
↪  {V,F}
% Lista é a lista das datas em que o utente visitou a instituição

datasVisitaInst( N,I,L ) :-
    findall( D,( utente( IDU,N,_,_ ),ato( D,IDU,IDS,_ ),cuidado(
    ↪  IDS,_,I,_ ) ),L ).



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

%--------------------------------- - - - - - - - - - -  -  -
% Extensao do predicado concat: Lista1, Lista2, R -> {V,F}

concat( [],L,L ).
concat( [X|Xs],L2,[X|L] ) :-
    concat( Xs,L2,L ).


%--------------------------------- - - - - - - - - - -  -  -
% Extensao do predicado tiraRepetidos: Lista, Resultado -> {V,F}

tiraRepetidos([],[]).
tiraRepetidos([H|Lista],Res) :-
    eliminaElementos(H,Lista,R),
    concat([H],Rn,Res),
    tiraRepetidos(R,Rn).


%--------------------------------- - - - - - - - - - -  -  -
```

```
% Extensao do predicado eliminaElementos: Elemento, Lista,
↪  Resultado -> {V,F}

eliminaElementos(X,[],[]).
eliminaElementos(X,[X|T],R) :-
    eliminaElementos(X,T,R).
eliminaElementos(X,[H|T],[H|Z]) :-
    X \== H,
    eliminaElementos(X,T,Z).


%--------------------------------- - - - - - - - - - -  -  -
% Extensao do meta-predicado nao: Questao -> {V,F}

nao( Questao ) :-
    Questao,
    !, fail.

nao( Questao ).


%--------------------------------- - - - - - - - - - -  -  -
% Extensao do predicado pertence: Elem, Lista -> {V,F}

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


%--------------------------------- - - - - - - - - - -  -  -
% Extensao do predicado solucoes

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


%--------------------------------- - - - - - - - - - -  -  -  -
↪  -  -
% Predicado «comprimento» que calcula o número de elementos
% existentes numa lista

comprimento( L,S ) :-
    length( L,S ).


%--------------------------------- - - - - - - - - - -  -  -
% Extensão do predicado que permite a evolução do conhecimento

evolucao( Termo ) :-
    solucoes( INV,+Termo::INV,LINV ),
    insercao( Termo ),
    testa( LINV ).


%--------------------------------- - - - - - - - - - -  -  -
% Extensão do predicado que permite a inserção de conhecimento

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

```
%--------------------------------- - - - - - - - - - - - - -
% Extensão do predicado que permite a involução do conhecimento

involucao( Termo ) :-
    solucoes( INV,-Termo::INV,LINV ),
    Termo,
    remocao( Termo ),
    testa( LINV ).


%--------------------------------- - - - - - - - - - - - - -
% Extensão do predicado que permite a remoção de conhecimento

remocao( Termo ) :-
    retract( Termo ).
remocao( Termo ) :-
    assert( Termo ),
    !, fail.


%--------------------------------- - - - - - - - - - - - - -
% Extensão do predicado que testa uma lista de termos

testa( [] ).
testa( [H|T] ) :-
    H,
    testa(T).


%--------------------------------- - - - - - - - - - - - - -
% Extensão do predicado que calcula a soma de um conjunto de
↪  valores

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


%--------------------------------- - - - - - - - - - - - - -
% Extensão do predicado que calcula o último elemento de uma lista

ultimo([X|Xs], Last) :-  lastAux(Xs, X, Last).

lastAux([], Last, Last).
lastAux([X|Xs], _, Last) :- lastAux(Xs, X, Last).


%--------------------------------- - - - - - - - - - - - - -
% Extensao do predicado maximo: Lista, Resultado -> {V,F}
% Resultado é o maximo da lista de tuplos

tuploMaximo( [(X,Y)], (X,Y) ).
tuploMaximo( [(X,Y)|T], (X,Y)) :-
    tuploMaximo(T, (XX,YY)),
    Y > YY.
tuploMaximo( [(X,Y)|T], (XX,YY)) :-
    tuploMaximo(T, (XX,YY)),
    Y =< YY.
```

Bibliografia

[1] BRATKO, I. *PROLOG Programming for Artificial Intelligence*, 2nd ed. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[2] FRADE, M. J. *Lógica Computacional*. Universidade do Minho. Departamento de Informática, Braga, Portugal, 2006.

[3] PROLOG, S. Introduction. `https://sicstus.sics.se/sicstus/docs/3.7.1/html/sicstus_1.html`.