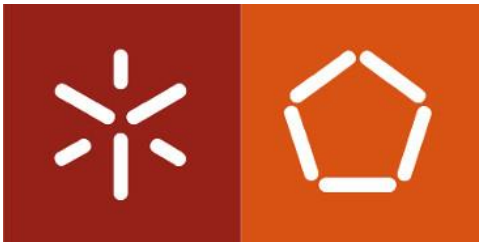
UNIVERSIDADE DO MINHO



Programação em Lógica

Mestrado Integrado em Engenharia Biomédica

Inteligência Artificial em Engenharia Biomédica

(1º Semestre/ Ano Letivo 2021/2022)

Grupo 10

Lara Alexandra Pereira Novo Martins Vaz (A88362)

Mariana Lindo Carvalho (A88360)

Tiago Miguel Parente Novais (A88397)

Braga

13 de novembro de 2021

Resumo

Este trabalho tem como principal objetivo o estudo da linguagem de programação em lógica (PROLOG) e a sua aplicação na representação de conhecimento, neste caso para caraterizar um universo de discurso na área de prestação de cuidados de saúde.

Com esta finalidade foi implementada uma base de conhecimento onde se encontram disponíveis informações relativas aos utentes, os seus prestadores e os atos realizados. Para a realização da listagem e pesquisas consideradas pertinentes para este trabalho foram desenvolvidos vários predicados. De forma a manipular de forma segura esta mesma base de conhecimento foram desenvolvidos invariantes.

Para a confirmação dos resultados pretendidos recorreu-se ao interpretador de PROLOG SWI, confirmando que todos os predicados e invariantes funcionavam conforme esperado.

Índice

1. Introdução

O presente trabalho tem como principal objetivo a utilização da linguagem de programação em lógica PROLOG, no âmbito da representação de conhecimento e da construção de mecanismos de raciocínio para a resolução de problemas. Como ferramenta de interpretação da linguagem PROLOG foi utilizado o interpretador SWI.

Neste trabalho foi desenvolvido um sistema de representação de conhecimento e raciocínio com capacidade para caraterizar um universo de discurso na área de prestação de cuidados de saúde. Esta caracterização foi feita com base nos utentes, nos prestadores de cuidados e nos atos realizados. Para o efeito, são requeridas algumas funcionalidades:

- Registar utentes, prestadores e atos médicos;
- Remover utentes, prestadores e atos médicos;
- Identificar utentes/prestadores/atos por critérios de seleção;
- Identificar atos prestados por prestador/cidade/datas/custo;
- Identificar os utentes de um prestador/instituição;
- Identificar atos realizados por utente/prestador/instituição;
- Calcular o custo total dos cuidados de saúde por utente/prestador/ato/data.

Com o intuito de garantir a consistência da base de conhecimento e a inexistência de informação repetida, foram criadas invariantes, que vão apresentar restrições à manipulação da base de conhecimento.

2. Preliminares

Este trabalho é suportado pela linguagem de programação lógica PROLOG, que é uma linguagem de programação do tipo declarativo para computação simbólica e não numérica. Os seus predicados correspondem a declarações da informação dos objetos em estudo, pelo que se conclui que um predicado é um conjunto de instâncias acerca de uma mesma relação que relaciona um conjunto de objetos. Os elementos básicos de PROLOG são herdados da lógica de predicados e são factos, regras e perguntas. Todos estes elementos terminam com um ponto final [1].

O PROLOG é, de facto, uma linguagem muito adequada para extrair informações de uma base de dados. A grande vantagem do PROLOG é a possibilidade de se referir a um objeto sem realmente especificar todos os seus componentes [1]. Neste sentido, um programa em PROLOG é uma base de dados em que a especificação das relações é parcialmente explícita (factos) e parcialmente implícita (regras) [2]. Então, é possível fazer referência a um objeto apenas indicando a sua estrutura de interesse e deixar de lado os componentes particulares em estruturas não especificadas [1].

As secções seguintes têm como objetivo abordar, de forma geral, os conceitos em cima referidos.

2.1. Factos

Os factos servem para estabelecer um relacionamento existente entre objetos de um determinado contexto de problema [1]. Por exemplo,

```
filho(joao,jose).
```

é um facto que estabelece que João é filho de José, ou seja, que a relação filho existe entre os objetos denominados João e José. Em PROLOG, os identificadores de relacionamento são denominados predicados e os termos consistem em variáveis e constantes. Estes últimos distinguem-se pela utilização de letra maiúscula para as variáveis e minúscula para as constantes [1]. Quando este programa for comunicado ao sistema PROLOG, será possível fazer algumas perguntas sobre a relação entre o João e o José.

2.2. Regras

O conceito regras permite definir novas relações em termos de outras relações já existentes. São constituídas por uma parte conclusiva, escrita à esquerda do símbolo ":-", e por uma parte condicional, escrita à direita do ":-" , com as condições separadas por vírgulas. Este exemplo,

```
pai ( X,Y ) :- filho( Y,X ).
```

define a regra pai, que estabelece que X é pai de Y se Y for filho de X. Uma diferença importante entre as regras e os factos é que um facto é sempre verdadeiro, enquanto as regras especificam algo que pode ser verdadeiro, mas apenas se determinadas condições forem satisfeitas. É importante referir que factos e regras são tipos de cláusulas e um conjunto de cláusulas constitui um programa lógico [2].

2.3. Questões

A utilização de perguntas em PROLOG permite recuperar informações de um programa lógico, verificando se um determinado relacionamento existe ou não existe entre objetos. Para obter uma resposta, o PROLOG tenta encontrar uma prova para demonstrar que o que está a ser perguntado é verdadeiro [1]. A pergunta:

```
?- pai( jose, joao ).
```

questiona se o José é pai do João, ou seja, interroga se a relação pai é válida para os objetos José e João. Tendo em conta o exemplo utilizado, a resposta seria '*yes*'. Desta forma, responder a uma pergunta com relação a um determinado programa, corresponde a determinar se a pergunta é consequência lógica do programa, ou seja, se essa questão pode ser deduzida dos factos expressos no programa. No caso de não ser possível encontrar nenhum facto que comprove a pergunta, o sistema retornaria '*no*'. Uma outra forma de realizar perguntas inclui o uso de variáveis, por exemplo:

```
?- filho( X, jose ).
```

Neste caso está a ser perguntado quem é filho do José, ou seja, se existe uma prova X que torna a pergunta verdadeira. Com esta questão, obtém-se a reposta: X = joao. Se existirem mais provas para a questão, isto é, se na base de conhecimento existirem mais factos que a satisfaçam, então é possível obter esses valores sequencialmente à medida que o utilizador valida o resultado [2]. Note-se que estas perguntas são feitas no interpretador de PROLOG SWI.

3. Descrição do Trabalho e Análise de Resultados

3.1. Declarações Iniciais

De forma a permitir ao utilizador um maior controlo da execução dos programas foram implementadas na base de conhecimento declarações iniciais. Estas declarações iniciais, denominadas *flags*, são definidas por um determinado valor, que pode ser alterado consoante as necessidades do utilizador.

Para o caso proposto, implementaram-se as seguintes *flags*:

```
% SWI PROLOG: Declarações iniciais
:- set_prolog_flag( discontiguous_warnings,off ).
:- set_prolog_flag( single_var_warnings,off ).
:- set_prolog_flag( unknown,fail ).
```

Às duas primeiras declarações foi-lhes atribuído o valor *off*, sendo que na primeira inibe-se o aparecimento de avisos quando cláusulas de um determinado predicado não estão juntas, e na segunda foram inibidos os avisos relativos a variáveis únicas. À última declaração foi atribuído o valor *fail*, que apresenta falha quando se chamam predicados que não estão definidos.

3.2. Definições Iniciais

Por defeito, o PROLOG define os predicados como estáticos. Para ser possível introduzir ou remover conhecimento na base, é necessário converter os predicados para

dinâmicos. Para isso, utilizaram-se definições iniciais, capazes de realizar esta conversão. Para o trabalho proposto, implementaram-se as seguintes definições iniciais:

```
% SWI PROLOG: Definições iniciais
:- op( 900,xfy,'::' ).
:- dynamic(utente/5).
:- dynamic(prestador/7).
:- dynamic(ato/6).
```

Para o operador de invariantes, foi definido como dinâmico o predicado *op*. Para os predicados que vão representar o conhecimento (*utente, prestador e ato*), estes foram definidos como dinâmicos. Além disso, foi também indicado o número de argumentos para cada predicado, sendo 5 para o *utente*, 7 para o *prestador* e 6 para o predicado *ato*.

3.3. Factos de Base

Inicialmente, no âmbito do trabalho proposto, começou-se por construir a base de conhecimento, sendo que esta é resultado da declaração de utentes, prestadores e atos. Efetivamente, a inserção dos factos e consequente construção da base de conhecimento foi possível devido à implementação de três predicados, designadamente: *utente*, *prestador* e *ato*.

3.3.1. Predicado *utente*

O predicado *utente* tem como atributos o ID do utente, o nome, a idade, a cidade e o sexo. De seguida, serão apresentados três exemplos do predicado *utente*. É de realçar que os restantes factos se encontram no Anexo A.

```
utente(1001, laravaz, 22, braga, feminino).
utente(1002, marianalindo, 20, vianadocastelo, feminino).
utente(1003, tiagonovais, 21, fafe, masculino).
```

3.3.2. Predicado *prestador*

O predicado *prestador* tem como atributos o ID do prestador, o tipo de prestador, a especialidade, o nome, a instituição, a cidade e o sexo. De seguida, serão apresentados

três exemplos do predicado *prestador*. É de realçar que os restantes factos se encontram no Anexo A.

```
prestador(2001,medico,cardiologia,ruialves,cuf,braga,masculino).
prestador(2002,enfermeiro,pediatria,anasilva,trofasaude,porto,fe
minino).
prestador(2003,medico,medicinainterna,teresaalves,publico,vianad
ocastelo,feminino).
```

3.3.3. Predicado *ato*

O predicado *ato* tem como atributos o ID do ato, a data do mesmo (no formato ddmmaa), o ID do utente, o ID do prestador, a descrição do ato e o custo, em euros. De seguida, será apresentado um exemplo do predicado *ato*. É de realçar que os restantes factos se encontram no Anexo A.

```
ato(3001, 051121, 1001, 2001, consulta,20).
ato(3002, 080921, 1008, 2002, curativo,5).
ato(3003, 080921, 1006, 2008, consulta,15).
```

3.4. Predicados Auxiliares

Os predicados desta secção são denominados de Predicados Auxiliares, uma vez que a sua função é ser a base de outros predicados necessários ao trabalho proposto. Estes vão ajudar as extensões dos predicados por critérios de seleção e os invariantes, que serão apresentados mais à frente, a atingirem o seu objetivo final.

3.4.1. Predicado *solucoes*

O predicado *solucoes* é utilizado quando se pretende listar todas as soluções como resposta a uma dada pergunta. Este predicado contém três argumentos: o X corresponde ao elemento que se pretende obter, o Y é o teorema que se pretende provar quando é feita uma dada pergunta e o Z é a lista que armazena o conjunto de soluções. Note-se que o *findall* é um predicado já implementado no interpretador SWI.

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

3.4.2. Predicado *comprimento*

Este predicado retorna o número de elementos de uma determinada lista. Para isso, recebe como argumentos uma lista L e o número de elementos dessa lista como resultado R. O critério de paragem deste predicado retorna comprimento 0, caso a lista seja vazia. Se o comprimento da lista L for N, sabe-se que o comprimento de uma lista que tem como cabeça um elemento X e como cauda L é N+1. Assim, este predicado utiliza a recursividade para somar o número de elementos de uma lista só termina quando atinge o critério de paragem.

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

3.4.3. Predicado *eliminarelemento*

O predicado *eliminarelemento* elimina todos os registos de um elemento X numa lista. Para isto recorre-se a três argumentos, o elemento X a eliminar, a lista L onde se encontra o mesmo e o resultado da operação, ou seja, a lista sem o elemento X. Esta lista final é representada por NL.

Para que este predicado seja executado, foram considerados três casos possíveis. Primeiramente, é apresentado o critério de paragem, em que se tenta eliminar um elemento X de uma lista vazia e obviamente é retornada uma lista vazia. De seguida considera-se o caso do elemento X ser o primeiro elemento da lista, eliminando-o e de seguida é utilizado um processo recursivo de pesquisa de elementos iguais na restante lista, obtendo-se como resultado NL. Por último, é considerado o caso em que o elemento a eliminar não é cabeça da lista, utilizando-se novamente um processo recursivo para encontrar o elemento no resto da lista, resultando na lista NL.

```
% Extensão do predicado eliminarelemento: Elemento, Lista,
Resultado -> {V,F}
eliminarelemento( X,[],[] ).
eliminarelemento( X,[X|L],NL ) :-
      eliminarelemento(X,L,NL).
```

```
eliminarelemento( X,[Y|L],[Y|NL] ) :-
      X \== Y,
      eliminarelemento( X,L,NL ).
```

3.4.4. Predicado *eliminarrepetidos*

O presente predicado tem a função de eliminar elementos repetidos de uma lista. Para isto, tem como argumentos uma lista de entrada e a lista resultante do processo.

De forma a este ser executado são consideradas duas possibilidades. Em primeiro lugar, é representado o caso de paragem, onde a lista introduzida é vazia, retornando assim uma lista vazia também. De seguida, é apresentado o caso contrário onde através de um processo recursivo são eliminados os elementos repetidos da lista introduzida resultando numa outra lista com elementos únicos.

```
% Extensão do predicado eliminarrepetidos: Lista, Resultado ->
{V,F}
eliminarrepetidos( [],[] ).
eliminarrepetidos( [A|B],[A|L] ) :-
      eliminarelemento( A,B,NL ),
      eliminarrepetidos( NL,L ).
```

3.4.5. Predicado *somal*

O predicado *somal* soma todos os elementos de uma lista introduzida. Para isso, este recorre apenas a dois argumentos, a lista de entrada e o resultado da soma dos elementos da mesma.

Assim, são consideradas duas possibilidades. Uma delas corresponde ao critério de paragem, quando a lista é constituída por apenas um elemento, tendo como resultado o próprio elemento. A outra corresponde à possibilidade de uma lista ser constituída por mais que um elemento. Neste caso considera-se que se a soma de elementos de uma lista L é R1. Então uma lista de cauda L e cabeça N terá como soma dos seus elementos R1+N. Este predicado possibilita a soma de todos os elementos utilizando um processo recursivo.

```
% Extensão do predicado soma: Lista, Resultado --> {V,F}
somal( [X],X ).
somal( [N|L],R ):-
```

```
      somal( L,R1 ),
      R is N+R1.
```

3.4.6. Predicado *inserção*

O predicado *inserção* tem como função atualizar a base de conhecimento através da inserção de termos. De forma a remover termos inseridos que são inválidos, é utilizado o *retract* em conjugação com *!, fail*. Isto obriga que o processo falhe, impedindo que seja executada a inserção.

Os predicados *assert* e *retract* estão previamente implementados no SWI.

```
% Extensão do predicado insercao: Termo -> {V,F}
insercao( Termo ) :-
      assert( Termo ).
insercao( Termo ) :-
      retract( Termo ), !, fail.
```

3.4.7. Predicado *teste*

O predicado *teste* tem como objetivo a validação de um termo após a sua inserção, verificando que não existem contradições.

```
% Extensão do predicado teste: Lista -> {V,F}
teste( [] ).
teste( [I|L] ) :-I,teste( L ).
```

3.4.8 Predicado remocao

O presente predicado tem como finalidade a remoção de termos, recorrendo à função do PROLOG *retract*. Este predicado recebe um termo como argumento e remove da base uma ou mais cláusulas (por retrocesso) que unificam com esse termo [2].

```
% Extensão do predicado remocao: Termo -> {V,F}
remocao( Termo ) :-
        retract(Termo).
remocao( Termo ) :-
        assert(Termo),!,fail.
```

3.5. Extensão dos Predicados

Os predicados referidos nesta secção têm como funcionalidade reunir informações e identificar elementos que respondam a um certo requisito. Todos os predicados utilizam o predicado auxiliar *solucoes* de forma a reunir todos os elementos que cumprem as condições estabelecidas. Para além do predicado *solucoes*, recorreu-se também aos predicados *eliminarrepetidos* e *somal*.

3.5.1. Identificar utentes por critérios de seleção

Os predicados seguintes têm como objetivo identificar os utentes que cumprem um determinado critério de seleção. Para isso, recorreu-se ao predicado *solucoes* que pesquisa no predicado *utente* (e nos *prestador* e *ato* sempre que necessário) o requisito introduzido. É de realçar que os resultados da identificação dos utentes, obtidos no interpretador SWI, se encontram no Anexo B.

3.5.1.1. Predicado *utentes_ato*

O predicado *utentes_ato* tem como finalidade identificar todos os utentes a que foi prestado um determinado ato. Recebe como argumento o ato a procurar e retorna a lista com os IDs dos utentes a quem foi prestado esse ato. Neste caso, além de ser utilizado o predicado *solucoes*, foi também utilizado o predicado auxiliar *eliminarrepetidos*, já que o mesmo ato pode ser efetuado várias vezes para o mesmo utente.

```
% Extensão do predicado utentes_ato: Ato, Resultado -> {V,F}
utentes_ato(B,R) :-
    solucoes(A, (ato(_,_,A,_,B,_), utente(A,_,_,_,_)), Y),
    eliminarrepetidos(Y,R).
```

3.5.1.2. Predicado *utentes_tipoprestador*

Com a finalidade de identificar os diferentes utentes a que foram prestados atos por um prestador de um determinado tipo, foi desenvolvido o predicado *utentes_tipoprestador*. Para isso, recebe como argumento o tipo de prestador e retorna a lista com o ID dos utentes que cumprem a condição.

Neste caso, torna-se necessário pesquisar também no predicado *ato*, uma vez que é nesse predicado que ocorre a associação de um utente com um prestador. Para este predicado, também se recorreu ao predicado auxiliar *eliminarrepetidos*, pois é possível que a um mesmo utente sejam prestados vários atos pelo mesmo tipo de prestador.

```
% Extensão do predicado utentes_tipoprestador: Tipo de
Prestador, Resultado -> {V,F}
utentes_tipoprestador(B,R) :-
    solucoes(C, (prestador(A,B,_,_,_,_,_), ato(_,_,C,A,_,_),
    utente(C,_,_,_,_)), Y),eliminarrepetidos(Y,R).
```

3.5.1.3. Predicado *utentes_especialidade*

O predicado *utentes_especialidade* permite identificar todos os utentes a que foram prestados atos por um prestador de uma dada especialidade. Para isso, recebe como argumento a especialidade do prestador e retorna a lista com o ID dos utentes que cumprem a condição. Também se recorreu ao predicado auxiliar *eliminarrepetidos*, uma vez que é possível que a um mesmo utente sejam prestados vários atos por prestadores com a mesma especialidade.

```
% Extensão do predicado utentes_especialidade: Especialidade,
Resultado -> {V,F}
utentes_especialidade( D,R ) :-
    solucoes( A, ( utente(A,_,_,_,_),
    ato(_,_,A,C,_,_),
    prestador(C,_,D,_,_,_,_) ), Y ),
    eliminarrepetidos( Y,R ).
```

3.5.1.4. Predicado *utentes_cidade*

Com a finalidade de identificar todos os utentes de uma determinada cidade, foi criado o predicado *utentes_cidade*. Este predicado recebe como argumento a cidade e retorna a lista dos IDs dos utentes que cumprem a condição.

```
% Extensão do predicado utentes_cidade: Cidade, Resultado -> {V,F}
utentes_cidade( B,R ) :-
     solucoes( A, utente(A,_,_,B,_), R ).
```

3.5.1.5. Predicado *utentes_sexo*

Com a finalidade de identificar os diferentes utentes que existem de cada sexo, foi desenvolvido o predicado *utentes_sexo*. Este predicado recebe como argumento o sexo do utente e retorna a lista dos IDs dos utentes que são daquele género.

```
% Extensão do predicado utentes_sexo: Sexo, Resultado -> {V,F}
utentes_sexo( B,R ) :-
      solucoes( A, utente(A,_,_,_,B), R ).
```

3.5.1.6. Predicado *utentes_data*

O predicado *utentes_data* permite averiguar todos os utentes a que foram prestados atos numa determinada data. Para isso, recebe como argumentos a data, e retorna a lista com os IDs dos utentes que cumprem a condição. Também se recorreu ao predicado auxiliar *eliminarrepetidos*, pois é possível que a um mesmo utente sejam prestados vários atos na mesma data.

```
% Extensão do predicado utentes_data: Data, Resultado -> {V,F}
utentes_data( B,R ) :-
      solucoes( A, ( utente(A,_,_,_,_), ato(_,B,A,_,_,_)), Y),
      eliminarrepetidos( Y,R ).
```

3.5.1.7. Predicado *utentes_custo*

Através do predicado *utentes_custo* é possível identificar todos os utentes a que foram prestados atos com o mesmo custo. Recebe como argumento o custo e retorna a lista com os IDs dos utentes que cumprem a condição. Mais uma vez recorreu-se ao predicado auxiliar *eliminarrepetidos*, uma vez que podem existir atos prestados ao mesmo utente que apresentam o mesmo custo.

```
% Extensão do predicado utentes_custo: Custo, Resultado -> {V,F}
utentes_custo( B,R ) :-
      solucoes( A, ( utente(A,_,_,_,_), ato(_,_,A,_,_,B)), Y),
      eliminarrepetidos( Y,R ).
```

3.5.1.8. Predicado *utentes_prestador*

O predicado *utentes_prestador* permite identificar todos os utentes a que foram prestados atos por um determinado prestador. Tem como argumento o ID de um prestador e como resultado a lista dos IDs dos utentes que cumprem a condição. Neste caso, foi necessário pesquisar também no predicado *ato*, uma vez que é nesse predicado que ocorre a associação de um utente a um prestador. Também se recorreu ao predicado auxiliar *eliminarrepetidos*, pois, um mesmo prestador pode prestar vários atos a um mesmo utente.

```
% Extensão do predicado utentes_prestador: Prestador, Resultado
->{V,F}
utentes_prestador(B,R) :-
    solucoes(A, (ato(_,_,A,B,_,_), utente(A,_,_,_,_),
  prestador(B,_,_,_,_,_,_)), Y),eliminarrepetidos(Y,R).
```

3.5.1.9. Predicado *utentes_instituicao*

O predicado *utentes_instituicao* permite identificar todos os utentes de uma dada instituição. Tem como argumento o nome de uma instituição e como resultado a lista dos IDs dos utentes que cumprem a condição. Neste caso, foi necessário pesquisar também no predicado *ato*, uma vez que é nesse predicado que ocorre a associação de um utente a um prestador e consequentemente a uma instituição. Também se recorreu ao predicado auxiliar *eliminarrepetidos*, pois, pode ser prestado mais do que um ato a um determinado utente, na mesma instituição.

```
% Extensão do predicado utentes_instituicao: Instituição,
Resultado -> {V, F}
utentes_instituicao(C,R) :-
    solucoes((A,D),   (   utente(A,_,_,_,_),   ato(_,_,A,B,_,_),
prestador(B,_,_,_,C,D,_)), Y ),
    eliminarrepetidos(Y,R).
```

3.5.2. Identificar atos por critérios de seleção

Os predicados seguintes têm como objetivo identificar os atos que cumprem um determinado critério de seleção. Para isso, recorreu-se ao predicado *solucoes* que

pesquisa no predicado *ato* (e nos *utente* e *prestador* sempre que necessário) o requisito introduzido. Quando o ato cumpre esse requisito, é retornada a lista com as descrições desses atos. É de realçar que os resultados da identificação dos atos, obtidos no interpretador SWI, se encontram no Anexo B.

3.5.2.1. Predicado *atos_prestador*

O predicado *atos_prestador* tem como finalidade identificar todos os atos realizados por um prestador. Recebe como argumento o ID do prestador a procurar e retorna a lista das descrições dos atos prestados por esse prestador. Mais uma vez, recorreu-se ao predicado auxiliar *eliminarrepetidos*, uma vez que o mesmo prestador pode realizar várias vezes o mesmo ato.

```
% Extensão do predicado atos_prestador: Prestador, Resultado ->
{V,F}
atos_prestador(A,R) :-
    solucoes(B, (ato(_,_,_,A,B,_), prestador(A,_,_,_,_,_,_)), Y),
    eliminarrepetidos(Y,R).
```

3.5.2.2. Predicado *atos_tipoprestador*

Com a finalidade de identificar todos os atos realizados por tipo de prestador, foi desenvolvido o predicado *atos_tipoprestador*. Este predicado recebe como argumento o tipo de prestador e retorna a lista das descrições dos atos realizados por esse tipo de prestador. Neste caso, também se recorreu ao predicado auxiliar *eliminarrepetidos*, uma vez que um mesmo tipo de prestador pode realizar várias vezes o mesmo ato.

```
% Extensão do predicado atos_tipoprestador: Tipo de Prestador,
Resultado -> {V,F}
atos_tipoprestador(C,R) :-
    solucoes(B, (ato(_,_,_,A,B,_), prestador(A,C,_,_,_,_,_)),Y),
    eliminarrepetidos(Y,R).
```

3.5.2.3. Predicado *atos_especialidade*

O predicado *atos_especialidade* tem como finalidade identificar todos os atos prestados por especialidade do prestador. Recebe como argumento a especialidade do

prestador a procurar e retorna a lista das descrições dos atos prestados por essa especialidade. Novamente, recorreu-se ao predicado auxiliar *eliminarrepetidos*, uma vez que prestadores da mesma especialidade podem realizar várias vezes o mesmo ato.

```
% Extensão do predicado atos_especialidade: Especialidade,
Resultado -> {V,F}
atos_especialidade(C,R) :-
    solucoes(B, (ato(_,_,_,A,B,_), prestador(A,_,C,_,_,_,_)),Y),
    eliminarrepetidos(Y,R).
```

3.5.2.4. Predicado *atos_cidade*

Com a finalidade de identificar todos os atos efetuados numa determinada cidade, foi criado o predicado *atos_cidade*. Este predicado recebe como argumento uma cidade e retorna a lista das descrições dos atos realizados nessa cidade. Uma vez mais recorreu-se ao predicado auxiliar *eliminarrepetidos*, já que uma cidade pode ter diferentes instituições que prestam os mesmos atos.

```
% Extensão do predicado atos_cidade: Cidade, Resultado -> {V,F}
atos_cidade(C,R) :-
    solucoes(B, (ato(_,_,_,A,B,_), prestador(A,_,_,_,_,C,_)),Y),
    eliminarrepetidos(Y,R).
```

3.5.2.5. Predicado *atos_data*

O predicado *atos_data* permite consultar todos os atos realizados numa determinada data. Para isso, recebe como argumentos a data, e retorna a lista com as descrições dos atos que cumprem a condição. Neste caso, também se recorreu ao predicado auxiliar *eliminarrepetidos*, pois o mesmo ato pode ser realizado várias vezes na mesma data.

```
% Extensão do predicado atos_data: Data, Resultado -> {V,F}
atos_data(A,R) :-
    solucoes(B, ato(_,A,_,_,B,_),Y),
    eliminarrepetidos(Y,R).
```

3.5.2.6. Predicado *atos_custo*

Através do predicado *atos_custo* é possível identificar todos os atos prestados a um determinado custo. Recebe como argumento o custo e retorna a lista das descrições de atos que são prestados a esse custo. Para este predicado não foi necessário recorrer ao predicado auxiliar *eliminarrepetidos*, uma vez que um mesmo ato tem sempre o mesmo custo.

```
% Extensão do predicado atos_custo: Custo, Resultado -> {V,F}
atos_custo(B,R) :-
    solucoes(A, ato(_,_,_,_,A,B),Y),
    eliminarrepetidos(Y,R).
```

3.5.2.7. Predicado *atos_utente*

Este predicado tem como objetivo identificar todos os utentes sujeitos a um determinado ato. Assim, recebe como argumento o ID do utente e retorna a lista das descrições dos atos que cumprem a condição. Mais uma vez, recorreu-se ao predicado auxiliar *eliminarrepetidos*, já que o mesmo ato pode ser efetuado várias vezes para o mesmo utente.

```
% Extensão do predicado atos_utente: ID Utente, Resultado ->
{V,F}
atos_utente(A,R):-
     solucoes(B, (ato(_,_,A,_,B,_), utente(A,_,_,_,_)), Y),
     eliminarrepetidos(Y,R).
```

3.5.2.8. Predicado *atos_instituicao*

O predicado *atos_instituicao* tem como finalidade identificar todos os atos prestados por prestadores de uma dada instituição. Recebe como argumento a instituição a procurar e retorna a lista das descrições dos atos prestados por essa instituição. Neste caso, também se recorreu ao predicado auxiliar *eliminarrepetidos*, uma vez que um mesmo ato pode ser prestado várias vezes na mesma instituição.

```
% Extensão do predicado atos_instituicao: Instituição, Resultado
-> {V,F}
atos_instituicao(C,R) :-
```

```
    solucoes(B, (ato(_,_,_,A,B,_), prestador(A,_,_,_,C,_,_)),Y),
    eliminarrepetidos(Y,R).
```

3.5.3. Identificar prestadores por critérios de seleção

Os predicados seguintes têm como objetivo identificar os prestadores que cumprem um determinado critério de seleção. Para isso, recorreu-se ao predicado *solucoes* que pesquisa no predicado *prestador* (e nos *utente* e *ato* sempre que necessário) o requisito introduzido. Quando o prestador cumpre esse requisito, é retornada a lista com os IDs dos prestadores. É de realçar que os resultados da identificação dos prestadores, obtidos no interpretador SWI, se encontram no Anexo B.

3.5.3.1. Predicado *prestadores_atos*

O predicado *prestadores_ato* tem como finalidade identificar todos os prestadores que realizaram um determinado ato. Recebe como argumento o ato a procurar e retorna a lista com os IDs dos prestadores que realizaram esse ato. Mais uma vez recorreu-se ao predicado auxiliar *eliminarrepetidos*, já que o mesmo ato pode ser efetuado por vários prestadores.

```
% Extensão do predicado prestadores_atos: Ato, Resultado ->
{V,F}
prestadores_atos(B,R) :-
    solucoes(A, (ato(_,_,_,A,B,_), prestador(A,_,_,_,_,_,_)),Y),
    eliminarrepetidos(Y,R).
```

3.5.3.2. Predicado *prestadores_tipoprestador*

Com a finalidade de identificar os diferentes prestadores que existem para cada tipo de prestador, foi desenvolvido o predicado *prestadores_tipoprestador*. Este predicado recebe como argumento o tipo de prestador e retorna a lista dos IDs dos prestadores que são daquele tipo.

```
% Extensão do predicado prestadores_tipoprestador: Tipo de
Prestador, Resultado -> {V,F}
prestadores_tipoprestador(B,R) :-
    solucoes(A, prestador(A,B,_,_,_,_,_), R).
```

3.5.3.3. Predicado *prestadores_especialidade*

O predicado *prestadores_especialidade* tem como finalidade identificar os diferentes prestadores para cada especialidade. Recebe como argumento a especialidade do prestador a procurar e retorna a lista dos IDs dos prestadores que são daquela especialidade.

```
% Extensão do predicado prestadores_especialidade:
Especialidade, Resultado -> {V,F}
prestadores_especialidade(B,R) :-
    solucoes(A, prestador(A,_,B,_,_,_,_), R).
```

3.5.3.4. Predicado *prestadores_cidade*

Com a finalidade de identificar todos os prestadores que prestam atos numa determinada cidade, foi criado o predicado *prestadores_cidade*. Este predicado recebe como argumento a cidade e retorna a lista dos IDs dos prestadores que cumprem a condição.

```
% Extensão do predicado prestadores_cidade: Cidade, Resultado ->
{V,F}
prestadores_cidade(B,R) :-
    solucoes(A, prestador(A,_,_,_,_,B,_), R).
```

3.5.3.5. Predicado *prestadores_sexo*

Com a finalidade de identificar os diferentes prestadores que existem de cada sexo, foi desenvolvido o predicado *prestadores_sexo*. Este predicado recebe como argumento o sexo do prestador e retorna a lista dos IDs dos prestadores que são daquele género.

```
% Extensão do predicado prestadores_sexo: Sexo, Resultado ->
{V,F}
prestadores_sexo(B,R) :-
    solucoes(A, prestador(A,_,_,_,_,_,B), R).
```

3.5.3.6. Predicado *prestadores_data*

O predicado *prestadores_data* permite averiguar todos os prestadores que realizaram atos numa determinada data. Para isso, recebe como argumentos a data, e retorna a lista

com os IDs dos prestadores que cumprem a condição. Também se recorreu ao predicado auxiliar *eliminarrepetidos*, pois é possível que um mesmo prestador realize vários atos na mesma data.

```
% Extensão do predicado prestadores_data: Data, Resultado ->
{V,F}
prestadores_data(A,R) :-
    solucoes(B, (ato(_,A,_,B,_,_), prestador(B,_,_,_,_,_,_)),Y),
    eliminarrepetidos(Y,R).
```

3.5.3.7. Predicado *prestadores_custo*

Através do predicado *prestadores_custo* é possível identificar todos os prestadores que realizaram atos com o mesmo custo. Recebe como argumento o custo e retorna a lista com os IDs dos prestadores que cumprem a condição. Mais uma vez recorreu-se ao predicado auxiliar *eliminarrepetidos*, uma vez que podem existir atos prestados pelo mesmo prestador que apresentam o mesmo custo.

```
% Extensão do predicado prestadores_custo: Custo, Resultado ->
{V,F}
prestadores_custo(B,R) :-
    solucoes(A, (ato(_,_,_,A,_,B), prestador(A,_,_,_,_,_,_)),Y),
    eliminarrepetidos(Y,R).
```

3.5.3.8. Predicado *prestadores_utente*

Este predicado tem como objetivo identificar todos os utentes aos quais foram prestados atos por um determinado prestador. Assim, recebe como argumento o ID do utente e retorna a lista com os IDs dos prestadores que cumprem a condição. Para este predicado recorreu-se ao predicado auxiliar *eliminarrepetidos*, já que o mesmo prestador pode efetuar vários atos para o mesmo utente.

```
% Extensão do predicado prestadores_utente: Utente, Resultado ->
{V,F}
prestadores_utente(A,R) :-
    solucoes(B, (ato(_,_,A,B,_,_), prestador(B,_,_,_,_,_,_),
    utente(A,_,_,_,_)), Y),
```

```
    eliminarrepetidos(Y,R).
```

3.5.3.9. Predicado *prestadores_instituicao*

O predicado *prestadores_instituicao* tem como finalidade identificar todos os prestadores que realizaram atos numa dada instituição. Recebe como argumento a instituição a procurar e retorna a lista dos IDs dos prestadores que cumprem a condição.

```
% Extensão do predicado prestadores_instituicao: Instituicao, Resultado -> {V,F}
prestadores_instituicao(B,R) :-
    solucoes(A, prestador(A,_,_,_,B,_,_), R).
```

3.5.4. Calcular custo total por critérios de seleção

Os predicados seguintes têm como objetivo calcular o custo total dos atos prestados, segundo determinados critérios de seleção. Para isso, recorreu-se ao predicado *solucoes* que pesquisa nos predicados *ato, utente* e *prestador* o requisito introduzido. Quando o ato cumpre esse requisito, é retornado o custo total dos atos, através do predicado *somal*. É de realçar que os resultados do cálculo do custo total, obtidos no interpretador SWI, se encontram no Anexo B.

3.5.4.1. Predicado *custototal_utente*

Este predicado tem como finalidade calcular o custo total dos atos prestados a um determinado utente. Recebe como argumento o ID do utente e recorrendo ao predicado auxiliar *solucoes*, cria uma lista Y que contém os custos de todos os atos associados a esse utente. Essa lista Y é então, passada como argumento ao predicado auxiliar *somal* que retorna a soma de todos esses custos.

```
% Extensão do predicado custototal_utente: Utente, Resultado -> {V,F}
custototal_utente(A,R) :-
    solucoes(B, (utente(A,_,_,_,_), ato(_,_,A,_,_,B)), Y),
    somal(Y,R).
```

3.5.4.2. Predicado *custototal_prestador*

Este predicado tem como finalidade calcular o custo total dos atos prestados por um determinado prestador. Recebe como argumento o ID do prestador e recorrendo ao predicado auxiliar *solucoes*, cria uma lista Y que contém os custos de todos os atos realizados por esse prestador. Essa lista Y é então, passada como argumento ao predicado auxiliar *somal* que retorna a soma de todos esses custos.

```
% Extensão do predicado custototal_prestador: Prestador,
Resultado -> {V,F}
custototal_prestador(A,R) :-
    solucoes(B, ( prestador(A,_,_,_,_,_,_),
    ato(_,_,_,A,_,B)),Y), somal(Y,R).
```

3.5.4.3. Predicado *custototal_tipoprestador*

Este predicado tem como finalidade calcular o custo total dos atos prestados por um determinado tipo de prestador. Recebe como argumento o tipo do prestador e recorrendo ao predicado auxiliar *solucoes*, cria uma lista Y que contém os custos de todos os atos realizados por esse tipo de prestador. Essa lista Y é então, passada como argumento ao predicado auxiliar *somal* que retorna a soma de todos esses custos.

```
% Extensão do predicado custototal_tipoprestador: Tipo de
prestador, Resultado -> {V,F}
custototal_tipoprestador(B,R) :-
    solucoes(C, ( prestador(A,B,_,_,_,_,_),
    ato(_,_,_,A,_,C)),Y), somal(Y,R).
```

3.5.4.4. Predicado *custototal_especialidade*

Este predicado tem como finalidade calcular o custo total dos atos prestados por especialidade do prestador. Recebe como argumento a especialidade e recorrendo ao predicado auxiliar *solucoes*, cria uma lista Y que contém os custos de todos os atos realizados por essa especialidade. Essa lista Y é então, passada como argumento ao predicado auxiliar *somal* que retorna a soma de todos esses custos.

```
% Extensão do predicado custototal_especialidade: Especialidade,
Resultado -> {V,F}
custototal_especialidade(B,R) :-
```

```
    solucoes(C, ( prestador(A,_,B,_,_,_,_),
    ato(_,_,_,A,_,C)),Y), somal(Y,R).
```

3.5.4.5. Predicado *custototal_instituicao*

Com este predicado é possível calcular o custo total dos atos realizados numa determinada instituição. Recebe como argumento o nome da instituição e recorrendo ao predicado auxiliar *solucoes*, cria uma lista Y que contém os custos de todos os atos realizados nessa instituição. Essa lista Y é então, passada como argumento ao predicado auxiliar *somal* que retorna a soma de todos esses custos.

```
% Extensão do predicado custototal_instituicao: Instituição,
Resultado -> {V,F}
custototal_instituicao(B,R) :-
    solucoes(C, ( prestador(A,_,_,_,B,_,_),
  ato(_,_,_,A,_,C)),Y), somal(Y,R).
```

3.5.4.6. Predicado *custototal_ato*

O predicado *custototal_ato* tem como finalidade calcular o custo total dos atos realizados. De salientar, que neste caso consideraram-se as várias descrições de atos, não fazendo distinção da instituição em que foram realizados. Assim, recebe como argumento o nome do ato e recorrendo ao predicado auxiliar *solucoes*, cria uma lista Y que contém os custos de todos os atos prestados. Essa lista Y é então, passada como argumento ao predicado auxiliar *somal* que retorna a soma de todos esses custos.

```
% Extensão do predicado custototal_ato: Ato, Resultado -> {V,F}
custototal_ato(A,R) :-
    solucoes(B, ato(_,_,_,_,A,B),Y),somal(Y,R).
```

3.5.4.7. Predicado *custototal_data*

O predicado *custototal_data* tem como finalidade calcular o custo total dos atos realizados numa determinada data. Recebe como argumento uma data e recorrendo ao predicado auxiliar *solucoes*, cria uma lista Y que contém os custos de todos os atos prestados nessa data. Essa lista Y é então, passada como argumento ao predicado auxiliar *somal* que retorna a soma de todos esses custos.

```
% Extensão do predicado custototal_data: Data, Resultado ->
{V,F}
custototal_data(A,R) :-
    solucoes(B, ato(_,A,_,_,_,B),Y), somal(Y,R).
```

3.6. Predicados Principais

A criação dos predicados *registar* e *remover* surge com a necessidade de se poder modificar a base de conhecimento desenvolvida. Com estes predicados, é possível registar e remover factos relativos aos utentes, aos prestadores e aos atos, permitindo a manipulação dinâmica da base de conhecimento. De salientar, que estes predicados dependem da conformidade com os invariantes, descritos na secção seguinte 3.7.

3.6.1. Predicado *registar*

O predicado registar possibilita a inserção de novos factos relativos a um novo utente, prestador ou ato. Este predicado recebe como argumento um Termo, que corresponde ao conhecimento que queremos inserir. Através do predicado auxiliar *solucoes* (secção 3.4.1) gera-se uma lista (Linv), que inclui todos os invariantes descritos para o termo introduzido. De seguida, é feita a inserção desse Termo através da invocação do predicado *insercao*, que recorre à funcionalidade do *assert*. Posteriormente, invoca-se o predicado *teste*, responsável por verificar se cada um dos invariantes continua válido ou não. Caso algum invariante não seja válido retrocede-se à segunda cláusula do predicado *insercao*, sendo esta responsável pela remoção do facto que se tentou inserir, através da funcionalidade do *retract*. A presença do !, fail (insucesso na prova) na segunda cláusula do predicado *insercao* obriga a que o procedimento falhe aí e impede o seu retrocesso. Desta forma, é garantida a validade da base de conhecimento consoante os invariantes que foram criados, ou seja, é feito um teste à sua consistência.

```
% Extensão do predicado que permite a inserção de conhecimento:
% Termo -> {V,F}
registar( Termo ) :-
     solucoes(I, +Termo :: I, Linv ),
     insercao( Termo ),
     teste( Linv ).
```

3.6.2. Predicado *remover*

O predicado *remover* possibilita a remoção de um determinado facto da base de conhecimento, ou seja, remover informação relativa a um determinado utente, prestador ou ato. O predicado *remover* recebe como argumento um Termo, que corresponde ao conhecimento que se pretende remover. Através do predicado auxiliar *solucoes*, gera-se uma lista (Linv), que inclui todos os invariantes descritos para o Termo introduzido. De seguida, invoca-se o predicado *teste*, responsável por verificar se cada um dos invariantes continua válido ou não. Caso o invariante seja válido, invoca-se o predicado auxiliar *remocao*, responsável pela remoção do facto pretendido, através da funcionalidade do *retract*. Ao contrário do que se sucedia anteriormente para o predicado *registar*, em que os invariantes testavam a consistência da base de conhecimento, neste caso, os invariantes fazem um "teste à permissão" para remover conhecimento. Assim, o predicado *teste* é invocado antes do predicado *remocao*.

```
% Extensão do predicado que permite a remoção de conhecimento:
Termo % -> {V,F}
remover( Termo ) :-
      solucoes( I, -Termo :: I, Linv ),
      teste( Linv ),
      remocao( Termo ).
```

3.7. Invariantes

Os invariantes podem ser vistos como testes à consistência da base de conhecimento em momentos de alteração da mesma. A manipulação da base de conhecimento foi conseguida através da utilização de invariantes. Estes garantem uma integridade estrutural e referencial da base de conhecimento, ou seja, impedem a inserção de conhecimento repetido (invariantes estruturais) e asseguram a consistência da informação existente na base de conhecimento (invariantes referenciais).

3.7.1. Invariante para registar utente

Para a inserção de factos sobre utentes foi criado um invariante. Trata-se de um invariante de inserção referencial, que não permite o registo de um utente com um ID já existente na base de conhecimento.

```
+utente( A,B,C,D,E ) :: ( solucoes( A, utente( A,_,_,_,_ ), S),
                          comprimento( S,N ),
                          N == 1).
```

Exemplo de tentativa de registo de um utente com repetição de ID:

```
| ?- registar(utente(1001, laravaz, 22, braga, feminino)). false
```

3.7.2. Invariante para remover utente

Para a remoção de utentes da base de conhecimento foi criado um invariante, que apenas permite a remoção de um dado utente se este não estiver associado a nenhum ato.

```
-utente( A,B,C,D,E ) :: ( solucoes( A, ato( _,_,A,_,_,_ ), S ),
                          comprimento( S,N ), N == 0).
```

Exemplo de tentativa de remoção de um utente com um ato associado:

```
| ?- remover(utente(1002, marianalindo, 20, vianadocastelo,
feminino)). false
```

3.7.3. Invariante para registar prestador

Para a inserção de factos sobre prestadores foi criado um invariante. Trata-se de um invariante de inserção referencial, que não permite o registo de um prestador com um ID já existente na base de conhecimento.

```
+prestador( A,B,C,D,E,F,G ) :: ( solucoes( A,
prestador(A,_,_,_,_,_,_), S ),comprimento( S,N ), N == 1).
```

Exemplo de tentativa de registo de um prestador com repetição de ID:

```
| ?-
registar(prestador(2001,medico,cardiologia,ruialves,cuf,braga,ma
sculino)).false
```

3.7.4. Invariante para remover prestador

Para a remoção de prestadores da base de conhecimento foi criado um invariante, que apenas permite a remoção de um dado prestador se este não estiver associado a nenhum ato.

```
-prestador( A,B,C,D,E,F,G ) :: ( solucoes( A, ato( _,_,_,A,_,_
), S ),comprimento( S,N ), N == 0).
```

Exemplo de tentativa de remoção de um prestador com um ato associado:

```
| ?-
remover(prestador(2001,medico,cardiologia,ruialves,cuf,braga,mas
culino)).false
```

3.7.5. Invariante para registar ato

Para a inserção de factos sobre atos foi criado um invariante. Trata-se de um invariante de inserção referencial, que não permite o registo de um ato com um ID já existente na base de conhecimento.

```
+ato( A,B,C,D,E,F) :: ( solucoes( A, ato( A,_,_,_,_,_), S ),
                        comprimento( S,N ),
                        N == 1).
```

Exemplo de tentativa de remoção de um ato com repetição de ID:

```
| ?- registar(ato(3001, 051121, 1001, 2001, consulta,20)). False
```

3.7.6. Invariante para remover ato

Para a remoção de atos da base de conhecimento foi criado um invariante, que apenas permite a remoção de um ato, se este não tiver nenhum ID, data, utente e prestador associado, uma vez que não se pode remover um ato que já ocorreu.

```
-ato( A,B,C,D,E,F) :: ( solucoes( A, ato( A,B,C,D,, ), S ),
                        comprimento( S,N ),
                        N == 0).
```

Exemplo de tentativa de remoção de um ato com ID, data, utente e prestador associado:

```
| ?- remover(ato(3001, 051121, 1001, 2001, consulta,20)). false
```

4. Conclusões e Sugestões

Com a elaboração deste trabalho foi possível aplicar, num caso prático, os conceitos aprendidos na Unidade Curricular de Inteligência Artificial em Engenharia Biomédica, sobre a linguagem PROLOG.

A criação prévia de uma base de conhecimento que incluía cláusulas dos utentes, prestadores e atos, seguida da implementação de vários predicados, que auxiliaram na exploração dessa base, identificando os factos por critério de seleção, permitiu que estes mesmos fossem testados. Foram também implementados invariantes que permitiram a manipulação de conhecimento sem contradições ou repetições, sendo estes também testadas a nível funcional.

Após a realização de todos os testes considera-se que todos os objetivos estabelecidos foram satisfeitos. Ainda assim, conclui-se que há margem para melhorias através de uma maior exploração de outros predicados.

Referências Bibliográficas

[1] BRATKO, Ivan
"Prolog Programming for Artificial Intelligence, 2nd Edition"
Addison-Wesley, 1990.

[2] LAGO, Silvio.
"Introdução à linguagem Prolog"

Anexo A. Base de Conhecimento

Esta secção apresenta os factos que constituem a base de conhecimento construída para a elaboração deste trabalho. Alguns destes factos encontram-se na secção 3.3, como forma de demonstração.

```
%Extensão do predicado utente: ID Utente, Nome, Idade, Cidade, Sexo -> {V,F}

   utente(1001, laravaz, 22, braga, feminino).
   utente(1002, marianalindo, 20, vianadocastelo, feminino).
   utente(1003, tiagonovais, 21, fafe, masculino).
   utente(1004, luisvaz, 47, braga, masculino).
   utente(1005, manuelmartins, 86, barcelos, masculino).
   utente(1006, mariasilva, 3, famalicao, feminino).
   utente(1007, luisparente, 44 , braga, masculino).
   utente(1008, teresaferreira, 5, porto, feminino).
   utente(1009, nunocosta, 29, viladoconde, masculino).
   utente(1010, catarinacameira, 15, vianadocastelo, feminino).
   utente(1011, paulolopes, 58, porto, masculino).

% Extensão do predicado prestador: Id Prestador, Tipo, Especialidade, Nome, Instituição, Cidade e Sexo -> {V,F}

   prestador(2001,medico,cardiologia,ruialves,cuf,braga,masculino).
   prestador(2002,enfermeiro,pediatria,anasilva,trofasaude,porto,feminino).
   prestador(2003,medico,medicinainterna,teresaalves,publico,vianadocastelo,feminino).
   prestador(2004,enfermeiro,oftalmologia,goretiparente,publico,vianadocastelo, feminino).
   prestador(2005,medico,pneumologia,luismartins,cuf,braga,masculino).
   prestador(2006,medico,dentaria,jorgeamorim,trofasaude,porto,masculino).
```

```
   prestador(2007,medico,dermatologia,diogopereira,trofasaude,br
aga,masculino).
   prestador(2008,medico,pediatria,rosasilva,publico,braga,
feminino).

% Extensão do predicado ato: Id Ato, Data, ID Utente, ID Prestador,
Descricao e Custo -> {V,F}

   ato(3001, 051121, 1001, 2001, consulta,20).
   ato(3002, 080921, 1008, 2002, curativo,5).
   ato(3003, 080921, 1006, 2008, consulta,15).
   ato(3004, 070520, 1003, 2003, internamento,50).
   ato(3005, 070520, 1007, 2004, exame,16).
   ato(3006, 231120, 1002, 2005, consulta,21).
   ato(3007, 010921, 1010, 2006, limpezadentaria,34).
   ato(3008, 050720, 1004, 2004, consulta,8).
   ato(3005, 210821, 1007, 2004, exame,16).
   ato(3006, 031021, 1011, 2007, consulta,20).
```

Anexo B. Perguntas

Esta secção apresenta exemplos de resultados da identificação dos utentes, atos prestadores, obtidos no interpretador SWI, segundo critérios de seleção através utilização dos predicados apresentados na secção 3.5.

B.1. Identificar utentes por critério de seleção

Utentes por cidade

```
?-
|    utentes_cidade(Braga,X).
X = [1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009|...]

?- utentes_cidade(vianadocastelo,X).
X = [1002, 1010].
```

Utentes por sexo

```
?- utentes_sexo(feminino,Y).
Y = [1001, 1002, 1006, 1008, 1010].

?- utentes_sexo(masculino,Y).
Y = [1003, 1004, 1005, 1007, 1009, 1011].
```

Utentes por data

```
?- utentes_data(080921,X).
X = [1006, 1008] ,

?- utentes_data(070520,X).
X = [1003, 1007] ,
```

Utentes por custo

```
?- utentes_custo(16,X).
X = [1007] ,

?- utentes_custo(20,X).
X = [1001, 1011] ,
```

Utentes por ato

```
?- utentes_ato(internamento,X).
X = [1003] ,

|    utentes_ato(consulta,X).
X = [1001, 1006, 1002, 1004, 1011]
```

Utentes por especialidade

```
?- utentes_especialidade(oftalmologia,X).
X = [1004, 1007] .

?- utentes_especialidade(cardiologia,X).
X = [1001]
```

Utentes por instituição

```
?- utentes_instituicao(trofasaude,X).
X = [(1008, porto), (1010, porto), (1011, braga)] .

?- utentes_instituicao(publico,X).
X = [(1003, vianadocastelo), (1004, vianadocastelo), (1006, braga), (1007, vianadocastelo)]
```

Utentes por prestador

```
?- utentes_prestador(2002,X).
X = [1008] .

?- utentes_prestador(2004,X).
X = [1007, 1004] .
```

Utentes por tipo de prestador

```
?- utentes_tipoprestador(medico,X).
X = [1001, 1003, 1002, 1010, 1011, 1006] .

?- utentes_tipoprestador(enfermeiro,X).
X = [1008, 1007, 1004] .
```

B2. Identificar atos por critério de seleção

Atos por prestador

```
?- atos_prestador(2001,X).
X = [consulta] .

?- atos_prestador(2006,X).
X = [limpezadentaria] .

?- atos_prestador(2004,X).
X = [exame, consulta]
```

Atos por tipo de prestador

```
?- atos_tipoprestador(medico,X).
X = [consulta, internamento, limpezadentaria] ,

?- atos_tipoprestador(enfermeiro,X).
X = [curativo, exame, consulta]
```

Atos por especialidade

```
?- atos_especialidade(cardiologia,X).
X = [consulta] ,

?- atos_especialidade(oftalmologia,X).
X = [exame, consulta]
```

Atos por data

```
?- atos_data(080921,X).
X = [curativo, consulta] ,

?- atos_data(231120,X).
X = [consulta]
```

Atos por custo

```
?- atos_custo(16,X).
X = [exame] ,

?- atos_custo(20,X).
X = [consulta] ,
```

Atos por utente

```
?- atos_utente(1001,X).
X = [consulta] ,

?- atos_utente(1007,X).
X = [exame]
```

Atos por instituição

```
?- atos_instituicao(trofasaude,X).
X = [curativo, limpezadentaria, consulta] ,

?- atos_instituicao(publico,X).
X = [consulta, internamento, exame]
```

B3. Identificar prestadores por critério de seleção

Prestadores por atos

```
?- prestadores_atos(consulta,X).
X = [2001, 2008, 2005, 2004, 2007] ,

?- prestadores_atos(exame,X).
X = [2004]
```

Prestadores por tipos de prestadores

```
?- prestadores_tipoprestador(medico,X).
X = [2001, 2003, 2005, 2006, 2007, 2008].

?- prestadores_tipoprestador(enfermeiro,X).
X = [2002, 2004].
```

Prestadores por especialidade

```
?- prestadores_especialidade(cardiologia,X).
X = [2001].

?- prestadores_especialidade(pediatria,X).
X = [2002, 2008].
```

Prestadores por cidade

```
?- prestadores_cidade(braga,X).
X = [2001, 2005, 2007, 2008].

?- prestadores_cidade(porto,X).
X = [2002, 2006].
```

Prestadores por sexo

```
?- prestadores_sexo(masculino,X).
X = [2001, 2005, 2006, 2007].

?- prestadores_sexo(feminino,X).
X = [2002, 2003, 2004, 2008].
```

Prestadores por data

```
?- prestadores_data(080921,X).
X = [2002, 2008] ,

?- prestadores_data(070520,X).
X = [2003, 2004] ,
```

Prestadores por custo

```
?- prestadores_custo(16,X).
X = [2004] ,

?- prestadores_custo(20,X).
X = [2001, 2007]
```

Prestadores por utente

```
?-
|    prestadores_utente(1001,X).
X = [2001] ,

?- prestadores_utente(1007,X).
X = [2004] ,
```

Prestadores por instituição

```
?- prestadores_instituicao(trofasaude,X).
X = [2002, 2006, 2007].

?- prestadores_instituicao(publico,X).
X = [2003, 2004, 2008].
```

B4. Calcular custo total por critério de seleção

Custo Total por utente

```
?- custototal_utente(1007,X).
X = 32 ,

?- custototal_utente(1001,X).
X = 20 ,
```

Custo Total por prestador

```
?- custototal_prestador(2001,X).
X = 20 ,

?- custototal_prestador(2002,X).
X = 5 ,
```

Custo Total por tipo de prestador

```
?- custototal_tipoprestador(medico,X).
X = 160 ,

?- custototal_tipoprestador(enfermeiro,X).
X = 45 ,
```

Custo Total por especialidade

```
?- custototal_especialidade(oftalmologia,X).
X = 40 .

?- custototal_especialidade(dentaria,X).
X = 34 .
```

Custo Total por instituição

```
?- custototal_instituicao(trofasaude,X).
X = 59 .

?- custototal_instituicao(cuf,X).
X = 41 .
```

Custo Total por ato

```
?- custototal_ato(curativo,X).
X = 5 .

?- custototal_ato(consulta,X).
X = 84 .
```

Custo Total por data

```
?- custototal_data(080921,X).
X = 20 .

?- custototal_data(051121,X).
X = 20 .
```