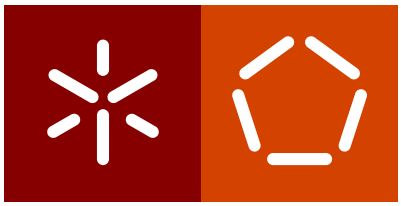


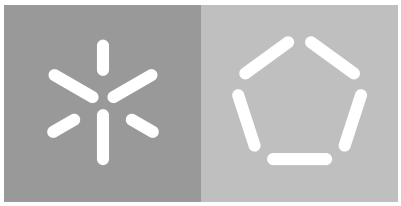
Universidade do Minho
Escola de Engenharia
Departamento de Informática

Filipe C. Oliveira, Filipe S. Marques, Luís F. Mendes

Sistemas de Representação de Conhecimento e Raciocínio

Exercício 1
Grupo 1

Março 2016



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Filipe C. Oliveira, Filipe S. Marques, Luís F. Mendes

Sistemas de Representação de Conhecimento e Raciocínio

Exercício 1
Grupo 1

Março 2016

CONTENTS

1

INTRODUÇÃO

A componente prática da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio, inserida no 3º ano do plano curricular do Mestrado Integrado em Engenharia Informática, pressupõem a realização de trabalhos de grupo na forma de um conjunto de exercícios. Este será portanto o primeiro dos exercícios práticos, tendo por principal tema a programação em lógica em PROLOG e o conceito de invariantes, no âmbito da representação de conhecimento e construção de mecanismos de raciocínio, por forma a caracterizar um universo de discurso com o qual se pretende abordar a temática do registo de eventos numa instituição de saúde.

Para o efeito, foi desenvolvido um exemplo prático elucidativo deste panorama, incluindo a identificação simplista das instituições, serviços, profissionais, utentes, e registos de eventos médicos. Foram ainda apresentadas um conjunto de funcionalidades mínimas, as quais foram devidamente cumpridas, ressalvando-se ainda a adição de funcionalidades extra que o grupo julgou serem relevantes para exemplo prático.

2

CARACTERIZAÇÃO DO PROBLEMA

A elaboração do caso prático anteriormente descrito tinha como funcionalidades obrigatórias:

- 1) Identificar os serviços existentes numa instituição;
- 2) Identificar os utentes de uma instituição;
- 3) Identificar os utentes de um determinado serviço;
- 4) Identificar os utentes de um determinado serviço numa instituição;
- 5) Identificar as instituições onde seja prestado um dado serviço ou conjunto de serviços;
- 6) Identificar os serviços que não se podem encontrar numa instituição;
- 7) Determinar as instituições onde um profissional presta serviço;
- 8) Determinar todas as instituições (ou serviços, ou profissionais) a que um utente já recorreu;
- 9) Registar utentes, profissionais, serviços ou instituições;
- 10) Remover utentes (ou profissionais, ou serviços, ou instituições) dos registos.

Ora, associando às funcionalidades descritas o tipo de conhecimento envolvido para a sua resolução obtemos:

- 1) Identificar os serviços existentes numa instituição: {**Serviço, Instituição**};
- 2) Identificar os utentes de uma instituição: {**Utente, Instituição**};
- 3) Identificar os utentes de um determinado serviço: {**Utente, Serviço**};
- 4) Identificar os utentes de um determinado serviço numa instituição: {**Utente, Serviço, Instituição**};
- 5) Identificar as instituições onde seja prestado um dado serviço ou conjunto de serviços: {**Serviço, Instituição**};
- 6) Identificar os serviços que não se podem encontrar numa instituição: {**Utente, Serviço, Instituição**};
- 7) Determinar as instituições onde um profissional presta serviço: {**Profissional, Serviço, Instituição**};

- 8) Determinar todas as instituições (ou serviços, ou profissionais) a que um utente já recorreu: {**Utente, Serviço, Instituição, Profissional, Registo Evento Médico**};
- 9) Registar utentes, profissionais, serviços ou instituições: {**Utente, Serviço, Instituição, Profissional**};
- 10) Remover utentes (ou profissionais, ou serviços, ou instituições) dos registos: {**Utente, Serviço, Instituição, Profissional**};

3

SOLUÇÃO PROPOSTA

Sabemos agora quais as entidades que teremos que relacionar para a resolução de cada funcionalidade. Resta-nos portanto indicar qual o conhecimento necessário a cada entidade (**Utente, Serviço, Instituição, Profissional, Registo Evento Médico**).

3.1 ENTIDADES

3.1.1 *Utente*

A única informação lógica relevante relativa ao utente, para o nosso exemplo prático, é o nome do mesmo, sendo portanto considerado que cada utente tem um nome único, não sendo permitidos dois utentes com nomes iguais, como formularemos logicamente nas seções 3.2.10 e 3.2.11. Assim, os enunciados:

- i) Utente Filipe Oliveira.
- ii) Utente Filipe Marques.
- iii) Utente Luís Mendes.

seriam traduzidos logicamente no nosso exemplo prático em:

```
utente(filipe_oliveira).
utente(filipe_marques).
utente(luis_mendes).
```

3.1.2 *Instituição*

A única informação lógica relevante relativa à instituição, para o nosso exemplo prático, é o nome da mesma, sendo portanto considerado que cada instituição tem um nome único, não sendo permitidos duas instituições com nomes iguais, como formularemos logicamente nas seções 3.2.10 e 3.2.11. Assim, os enunciados:

- i) Instituição Hospital da Luz.
- ii) Instituição Hospital da Boavista.

- iii) Instituição Hospital de São Marcos.

seriam traduzidos logicamente no nosso exemplo prático em:

```
instituicao(hospital_luz).
instituicao(hospital_boavista).
instituicao(hospital_sao_marcos).
```

3.1.3 *Serviço*

Relativamente aos serviços disponibilizados por uma instituição a informação lógica relevante, para o nosso exemplo prático, é o nome dos mesmos e a instituição à qual o serviço está associado, sendo portanto considerado que cada serviço distinto numa instituição tem um nome único, não sendo permitidos dois serviços com nomes iguais na mesma instituição, como formularemos logicamente nas seções 3.2.10 e 3.2.11. Assim, os enunciados:

- i) Serviço de Ortopedia na Instituição Hospital da Luz.
- ii) Serviço de Geriatria na Instituição Hospital da Boavista.
- iii) Serviço de Cardiologia na Instituição Hospital de São Marcos.

seriam traduzidos logicamente no nosso exemplo prático em:

```
servico(ortopedia,hospital_luz).
servico(geriatria,hospital_boavista).
servico(cardiologia,hospital_sao_marcos).
```

3.1.4 *Profissional*

Relativamente aos profissionais de saúde a informação lógica relevante, para o nosso exemplo prático, é o nome dos mesmos, assim como o serviço e instituição no qual executam os respectivos trabalhos, sendo portanto considerado que cada profissional distinto num serviço médico de uma instituição tem um nome único, não sendo permitidos dois profissionais com nomes iguais no mesmo serviço na mesma instituição, como formularemos logicamente nas seções 3.2.10 e 3.2.11. Assim, os enunciados:

- i) Profissional de Saúde Pedro Sanchez, no Serviço de Ortopedia na Instituição Hospital da Luz.
- ii) Profissional de Saúde Maria Carvalhais, no Serviço de Geriatria na Instituição Hospital da Boavista.
- iii) Profissional de Saúde Rui Perez, no Serviço de Cardiologia na Instituição Hospital de São Marcos.

seriam traduzidos logicamente no nosso exemplo prático em:

```
profissional(pedro_sanchez,ortopedia,hospital_luz).
profissional(maria_carvalhais,geriatria,hospital_boavista).
profissional(rui_perez,cardiologia,hospital_sao_marcos).
```

3.1.5 *Registo Evento Médico*

Relativamente aos registos de eventos médicos a informação lógica relevante, para o nosso exemplo prático, é o nome do utente, o nome do profissional de saúde envolvido no evento médico, assim como o serviço e instituição no qual executam os respectivos serviços médicos, sendo considerado que pode existir mais do que uma ocorrência médica que relacione da mesma forma os 4 intervenientes (utente,profissional,serviço,instituição), considerando-se que cada ocorrência representa um evento médico distinto, p.e., o utente Carlos poderá necessitar de mais do que uma consulta médica de nutrição com o mesmo profissional de saúde, no mesmo serviço e instituição, considerando-se cada consulta um evento médico independente, existindo portanto para cada uma um registo de evento médico na base de conhecimento. Assim, os enunciados:

- i) Evento Médico com o Utente António Sousa, Profissional de Saúde Pedro Sanchez, no Serviço de Ortopedia na Instituição Hospital da Luz.
- ii) Evento Médico com a Utente Maria Meireles, Profissional de Saúde Maria Carvalhais, no Serviço de Geriatria na Instituição Hospital da Boavista.
- iii) Evento Médico com o Utente Diamantino Marques, Profissional de Saúde Rui Perez, no Serviço de Cardiologia na Instituição Hospital de São Marcos.

seriam traduzidos logicamente no nosso exemplo prático em:

```
registo(antonio_sousa,hospital_luz,ortopedia,pedro_sanchez).
registo(maria_meireles,hospital_boavista,geriatria,maria_carvalhais).
registo(diamantino_marques,hospital_sao_marcos,cardiologia,rui_perez).
```

3.2 FUNCIONALIDADES OBRIGATÓRIAS

Caracterizado logicamente o universo de discurso, resta-nos elaborar os predicados capazes de responder às funcionalidades obrigatórias propostas. As seções seguintes descrevem a solução encontrada para cada uma das funcionalidades.

Em algumas das extensões de predicados irá encontrar o recurso a predicados como **solucoes(...),removerduplicados(...),...**. Esses mesmos predicados foram desenvolvidos no decorrer das aulas práticas da unidade curricular sendo os mesmos descritos na secção: A.2.

Denote que por forma a ilustrar correctamente todas as funcionalidades do sistema foi desenvolvida um exemplo de base de conhecimento que poderá ser consultado na seção A.1. Todos os testes são portanto o resultado de cada extensão de predicado a essa mesma base de conhecimento.

3.2.1 *Identificar os serviços existentes numa instituição*

Foi construída a extensão do predicado **servicosInstituicao(Instituicao,Servicos)**, que dada uma instituição calcula os serviços existentes nela, sendo os mesmos demonstrados em **Serviços**.

```
% 1) Extensao do predicado Identificar os servicos existentes numa instituicao
% servicosInstituicao(Instituicao,Servicos) -> {V,F}
servicosInstituicao(Instituicao,Servicos) :- solucoes(X, servico(X, Instituicao),
    Servicos).
```

*Teste do predicado **servicosInstituicao(Instituicao,Servicos)**:*

```
| ?- servicosInstituicao(hospital_sao_marcos,ListaServicos).
ListaServicos = [cardiologia,nutricionismo] ?
yes
```

```
| ?- servicosInstituicao(hospital_porto,ListaServicos).
ListaServicos = [cardiologia,nutricionismo,geriatria,neurologia,oncologia,
    clinica_geral] ?
yes
```

3.2.2 *Identificar os utentes de uma instituição*

Foi construída a extensão do predicado **utentesInstituicao(Instituicao,ListaUtentes)**, que dada uma instituição calcula os utentes da mesma, sendo os mesmos demonstrados em **ListaUtentes**.

```
% 2) Extensao do predicado Identificar os utentes de uma instituicao
% utentesInstituicao(Instituicao,ListaUtentes) -> {V,F}
utentesInstituicao(Instituicao,ListaUtentes) :-
  solucoes( (Utente), ( recorreuInstituicao(Utente, Instituicao) ), ListaUtentesRep
      ),
  removerduplicados(ListaUtentesRep, ListaUtentes).
```

Denote que a extensão do predicado **utentesServico** recorre ao predicado **recorreuInstituicao(Utente,Instituicao)** que será de seguida especificado:

```
% Extensao do predicado recorreuInstituicao que determina as instituicoes aos quais
     o utente recorreu
% recorreuInstituicao(Utente,Instituicao) -> {V,F}
recorreuInstituicao(Utente, Instituicao) :- registo(Utente,Instituicao,_,_).
```

*Teste do predicado **utentesInstituicao(Instituicao,ListaUtentes)**:*

```
| ?- utentesInstituicao(hospital_sao_marcos,ListaUtentes).
ListaUtentes = [antonio_sousa] ?
yes
```

```
| ?- utentesInstituicao(hospital_porto,ListaUtentes).
ListaUtentes = [antonio_marques,maria_meireles,diamantino_marques,rosa_sousa,
    jorge_marques] ?
yes
```

```
| ?- utentesInstituicao(hospital_lisboa,ListaUtentes).
ListaUtentes = [] ?
yes
```

3.2.3 *Identificar os utentes de um determinado serviço*

Foi construída a extensão do predicado **utentesServico(Servico,ListaUtentes)**, que dado um serviço calcula os utentes do mesmo, sendo os mesmos demonstrados em **ListaUtentes**.

```
% 3) Extensao do predicado Identificar os utentes de um determinado servico
% utentesServico(Servico,ListaUtentes) -> {V,F}
utentesServico(Servico,ListaUtentes) :-
  solucoes(Utente, recorreuServico(Utente, Servico), ListaUtentesRep),
  removerduplicados(ListaUtentesRep,ListaUtentes).
```

Denote que a extensão do predicado **utentesServico** recorre ao predicado **recorreuServico(Utente, Servico)** que será de seguida especificado:

```
% Extensao do predicado recorreuServico que determina os servicos aos quais o
    utente recorreu
```

```
% recorreuServico(Utente,Servico) -> {V,F}
recorreuServico(Utente, Servico) :- registo(Utente,_,Servico,_).
```

*Teste do predicado **utentesServico(Servico,ListaUtentes)**:*

```
| ?- utentesServico(geriatria,ListaUtentes).
ListaUtentes = [maria_meireles,diamantino_marques,rosa_sousa,jorge_marques] ?
yes
```

```
| ?- utentesServico(oncologia,ListaUtentes).
ListaUtentes = [] ?
yes
```

3.2.4 *Identificar os utentes de um determinado serviço numa instituição*

Foi construída a extensão do predicado **utentesServicoInstituicao(Servico,Instituicao,ListaUtentes)** , que dado um serviço e uma instituição calcula os utentes dos mesmos, sendo estes demonstrados em **ListaUtentes**.

```
% 4) Extensao do predicado Identificar os utentes de um determinado servico numa
    instituicao
% utentesServicoInstituicao(Servico,Instituicao,ListaUtentes) -> {V,F}
utentesServicoInstituicao(Servico,Instituicao,ListaUtentes) :-
  solucoes( (Utente) , (registo(Utente, Instituicao, Servico,_)), ListaUtentesRep),
  removerduplicados(ListaUtentesRep,ListaUtentes).
```

*Teste do predicado **utentesServicoInstituicao(Servico,Instituicao,ListaUtentes)** :*

```
| ?- utentesServicoInstituicao(geriatria,hospital_porto,ListaUtentes).
ListaUtentes = [maria_meireles,diamantino_marques,rosa_sousa,jorge_marques] ?
yes
```

```
| ?- utentesServicoInstituicao(clinica_geral,hospital_porto,ListaUtentes).
ListaUtentes = [] ?
yes
```

3.2.5 *Identificar as instituições onde seja prestado um dado serviço ou conjunto de serviços*

Foi construída a extensão do predicado **instituicoesComServicos**, que dado um serviço ou uma lista de serviços determina a lista de instituições onde este seja prestado, sendo estas demonstradas em **ListaInstituicoes**.

```
% 5) Extensao do predicado Identificar as instituicoes onde seja prestado um
    servico ou um conjunto de servicos
% instituicoesComServicos(Servico,Instituicao) -> {V,F}
% instituicoesComServicos([Servicos],Instituicao) -> {V,F}
instituicoesComServicos([ ],_).
instituicoesComServicos(Servico,ListaInstituicoes) :- solucoes((Instituicao), (
    servico(Servico,Instituicao)), ListaInstituicoes).
instituicoesComServicos([Servico | TailServicos], ListaInstituicoes) :-
  solucoes( (Instituicao),(servico(Servico,Instituicao)), ListaInst1),
  instituicoesComServicos(TailServicos, ListaInst2),
  concatenar(ListaInst1, ListaInst2, ListaInstituicoes).
```

*Teste do predicado **utentesServicoInstituicao(Servico,Instituicao,ListaUtentes)** :*

```
| ?- instituicoesComServicos(clinica_geral,ListaInstituicoes).
ListaInstituicoes = [hospital_braga,hospital_porto] ?
yes
```

```
| ?- instituicoesComServicos([cirurgia,nutricionismo],ListaInstituicoes).
ListaInstituicoes = [hospital_braga,hospital_lisboa] ?
yes
```

3.2.6 *Identificar os serviços que não se podem encontrar numa instituição*

Foi construída a extensão do predicado **servicosNaoEncontrados(Instituicao, Servicos)**, que dada uma instituição determina a lista dos serviços médicos que não se podem encontrar nesta, sendo os serviços médicos demonstrados em **ListaServicosNaoEncontrados**.

```
% 6) Extensao do predicado Identificar os servicos que nao se podem encontrar numa
    instituicao
% servicosNaoEncontrados(Instituicao,[Servicos]) -> {V,F}
servicosNaoEncontrados(Instituicao, ListaServicosNaoEncontrados) :-
  solucoes(Servico, servico(Servico, _), L1),
  removerduplicados(L1, ListaServicosTotal),
  solucoes(Servico, servico(Servico, Instituicao), ListaServicosInst),
  intercepcao(ListaServicosInst, ListaServicosTotal, R2),
  removerduplicados(R2, ListaServicosNaoEncontrados).
```

*Teste do predicado **servicosNaoEncontrados(Instituicao, Servicos)**:*

```
| ?- servicosNaoEncontrados(hospital_porto,ListaServicosNao).
ListaServicosNao = [cirurgia,psiquiatria] ?
yes
```

```
| ?- servicosNaoEncontrados(hospital_sao_marcos,ListaServicosNao).
ListaServicosNao = [geriatria,neurologia,oncologia,cirurgia,clinica_geral,
    psiquiatria] ?
yes
```

3.2.7 *Determinar as instituições onde um profissional presta serviço*

Foi construída a extensão do predicado **instituicoesProfissionalPrestaServico(Profissional,ListaInstituicoes)**, que dada um profissional de saúde determina a lista de instituições onde este presta serviço, sendo as instituições demonstradas em **ListaInstituicoes**.

```
% 7) Extensao do predicado Determinar as instituicoes onde um profissional presta
    servico
% instituicoesProfissionalPrestaServico(Professional, [Instituicoes] ) -> {V,F}
instituicoesProfissionalPrestaServico(Profissional,ListaInstituicoes) :-
  solucoes((Instituicao),profissional(Profissional,_,Instituicao),
      ListaInstituicoesDupl),
  removerduplicados(ListaInstituicoesDupl,ListaInstituicoes).
```

*Teste do predicado **servicosNaoEncontrados(Instituicao, Servicos)**:*

```
| ?- instituicoesProfissionalPrestaServico(filipe_oliveira,ListaInst).
ListaInst = [hospital_porto,hospital_braga] ?
yes
```

```
| ?- instituicoesProfissionalPrestaServico(tiago_sousa,ListaInst).
ListaInst = [hospital_lisboa] ?
yes
```

3.2.8 *Determinar todas as instituições (ou serviços, ou profissionais) a que um utente já recorreu*

Para responder logicamente a esta funcionalidade foram construídas as extensões de predicados **utenteRecorreuInstituicao(Utente,Instituicoes)**, **utenteRecorreuServico(Utente,Servicos)**, **utenteRecorreuProfissional(Utente,Profissionais)**, **utenteRecorreu(Utente,Lista)**. Os 3 primeiros predicados dado um utente determinam respectivamente as listas de instituições, serviços e profissionais de saúde a que o utente já recorreu. O último predicado aglomera a informação dos 3 primeiros predicados numa única lista. Passamos de seguida a especificar os 4 predicados:

```
% 8.0.1) Extensao do predicado Determinar todas as instituicoes a que um utente ja
    recorreu
% utenteRecorreuInstituicao(Utente,[Instituicoes]) -> {V,F}
utenteRecorreuInstituicao(Utente,Instituicoes) :-.
  solucoes(X, recorreuInstituicao(Utente,X), InstituicoesComDupl),
  removerduplicados(InstituicoesComDupl, Instituicoes).

% 8.0.2) Extensao do predicado Determinar todos os servicos a que um utente ja
    recorreu
% utenteRecorreuServico(Utente,[Servicos]) -> {V,F}
utenteRecorreuServico(Utente,Servicos) :-.
  solucoes(X, recorreuServico(Utente,X), ServicosComDupl),
  removerduplicados(ServicosComDupl, Servicos).

% 8.0.3) Extensao do predicado Determinar todos os profissionais a que um utente ja
     recorreu
%utenteRecorreuProfissional(Utente,[Profissionais]) -> {V,F}
utenteRecorreuProfissional(Utente,Profissionais) :-.
  solucoes(X, recorreuProfissional(Utente,X), ProfissionaisComDupl),
  removerduplicados(ProfissionaisComDupl, Profissionais).

% 8) Extensao do predicado Determinar todas as instituicoes (ou servicos, ou
    profissionais) a que um utente ja recorreu
% utenteRecorreu(Utente,Lista) -> {V,F}
utenteRecorreu(Utente,Lista) :-
  utenteRecorreuInstituicao(Utente,LInst),
  utenteRecorreuServico(Utente,LServ),
  utenteRecorreuProfissional(Utente,LProf),
  concatenar(LInst, LServ, LInstServ),
  concatenar(LInstServ,LProf,Lista).
```

Teste dos predicados ***utenteRecorreuInstituicao(Utente,Instituicoes)***, ***utenteRecorreuServico(Utente,Servicos)***, ***utenteRecorreuProfissional(Utente,Profissionais)***, ***utenteRecorreu(Utente,Lista)***:

```
| ?- utenteRecorreuInstituicao(antonio_sousa,Lista).
Lista = [hospital_sao_marcos] ?
yes
```

```
| ?- utenteRecorreuServico(antonio_sousa,Lista).
Lista = [cardiologia,nutricionismo] ?
yes
```

```
| ?- utenteRecorreuProfissional(antonio_sousa,Lista).
Lista = [vanessa_goncalves,filipe_oliveira] ?
yes
```

```
| ?- utenteRecorreu(antonio_sousa,Lista).
Lista = [hospital_sao_marcos,cardiologia,nutricionismo,vanessa_goncalves,
    filipe_oliveira] ?
yes
```

3.2.9 *Alterações à base de conhecimento*

De forma a ser possível mudar a base de conhecimento, utilizando os predicados evolucao (seção 3.2.10) e remocao (seção 3.2.11), foi necessário adicionar as seguintes definições iniciais ao prolog:

```
% SICStus PROLOG: Definicoes iniciais
% permitida a evolucao sobre utentes, profissionais, servicos, instituicoes,
    registos de actividade medica

:- op(900,xfy,'::').
:- dynamic utente/1.
:- dynamic profissional/3.
:- dynamic servico/2.
:- dynamic instituicao/1.
:- dynamic registo/4.
```

Contudo, necessitamos ainda de garantir a preservação da correção da base conhecimento em todas as operações de alteração à mesma. Assim, como veremos nas seções seguintes, todas as operações de remoção e adição respeitam os respectivos invariantes estruturais e referenciais definidos.

3.2.10 *Registar utentes, profissionais, serviços ou instituições*

Invariantes para operações de adição

Tal como referido na seção 3.2.9 é necessário preservar a correção da base de conhecimento. Ora, para as operações de adição de conhecimento necessitamos de garantir:

- utente:
  - Invariante Estrutural:
    - utentes distintos não têm o mesmo nome;

  traduzido logicamente para:

```
+utente(Utente)::(
  solucoes( (Utente), ( utente(Utente) ), Lista),
  comprimento(Lista,N),
  N==1
  ).
```

- instituição:
  - Invariante Estrutural:
    - instituições distintas não têm o mesmo nome;

  traduzido logicamente para:

```
+instituicao(Instituicao)::(
  solucoes( (Instituicao), ( instituicao(Instituicao) ), Lista),
  comprimento(Lista,N),
  N==1
  ).
```

- serviço:
  - Invariante Estrutural:
    - serviços distintos da mesma instituição não têm o mesmo nome;
  - Invariante Referencial:
    - a instituição associada ao serviço existe;

  traduzido logicamente para:

```
+servico(Servico,Instituicao)::(
  solucoes( (Servico,Instituicao) , ( servico(Servico,Instituicao) ), Lista),
  comprimento(Lista,N),
  N==1,
  solucoes( (Instituicao) , ( instituicao(Instituicao) ), ListaInst),
  comprimento(ListaInst,NInst),
```

```
  NInst==1
  ).
```

- profissional:
  - Invariante Estrutural:
    - profissionais distintos do mesmo serviço e instituição não têm o mesmo nome;
  - Invariante Referencial:
    - a instituição associada ao profissional existe;
    - o serviço associado ao profissional existe na instituição;

  traduzido logicamente para:

```
+profissional(Profissional,Servico,Instituicao)::(
  solucoes( (Profissional), ( profissional(Profissional,Servico,Instituicao) )
      , Lista),
  comprimento(Lista,N),
  N==1,
  solucoes( (Instituicao) , ( instituicao(Instituicao) ), ListaInst),
  comprimento(ListaInst,NInst),
  NInst==1,
  solucoes( (Servico,Instituicao) , ( servico(Servico,Instituicao) ),
      ListaServ),
  comprimento(ListaServ,NServ),
  NServ==1
  ).
```

- registo:
  - Invariante Estrutural:
    - tem que existir pelo menos um registo com essas características depois da inserção;
  - Invariante Referencial:
    - o utente associado ao registo existe;
    - a instituição associada ao registo existe;
    - o serviço associado ao registo existe na instituição;
    - o profissional associado ao registo está associado ao serviço e instituição;

  traduzido logicamente para:

```
+registo(Utente,Instituicao,Servico,Profissional)::(
  solucoes( (Utente,Instituicao,Servico,Profissional), ( registo(Utente,
      Instituicao,Servico,Profissional) ), Lista),
  comprimento(Lista,N),
  N>=1,
  solucoes( (Utente), ( utente(Utente) ), ListaUten),
```

```
    comprimento(ListaUten,NUten),
    NUten==1,
    solucoes( (Instituicao) , ( instituicao(Instituicao) ), ListaInst),
    comprimento(ListaInst,NInst),
    NInst==1,
    solucoes( (Servico,Instituicao) , ( servico(Servico,Instituicao) ),
        ListaServ),
    comprimento(ListaServ,NServ),
    NServ==1,
    solucoes( (Profissional,Servico,Instituicao), ( profissional(Profissional,
        Servico,Instituicao) ), ListaProf),
    comprimento(ListaProf,NProf),
    NProf==1
    ).
```

Predicados para operações de adição

Definidos os invariantes para as operações de adição à base de conhecimento, resta-nos definir as extensões de predicados que permitem registar utentes, profissionais, serviços, instituições, registos de eventos médicos.

Denote que as extensões de predicados seguidamente definidas recorrem ao predicado **evolucao(Termo)**, definido durante as aulas práticas da unidade curricular e possível de ser consultado na seção A.3.

```
% 9) Extensao do predicado Registar utentes, profissionais, servicos ou
    instituicoes

% Extensao do predicado que pemite registar utentes
% registarUtente(Nome) -> {V,F}
registarUtente(Nome) :- evolucao(utente(Nome)).

% Extensao do predicado que pemite registar instituicoes
% registarInstituicao(Instituicao) -> {V,F}
registarInstituicao(Instituicao) :- evolucao(instituicao(Instituicao)).

% Extensao do predicado que pemite registar servicos numa instituicao
% registarServico(Servico,Instituicao) -> {V,F}
registarServico(Servico,Instituicao) :- evolucao(servico(Servico,Instituicao)).

% Extensao do predicado que pemite registar profissionais num determinado servico
    de numa instituicao
% registarProfissional(Profissional,Servico,Instituicao) -> {V,F}
registarProfissional(Profissional,Servico,Instituicao) :- evolucao(profissional(
    Profissional,Servico,Instituicao)).

% Extensao do predicado que pemite registar eventos medicos numa instituicao
    indicando o utente, o profissional, o servico e a instituicao
% registarEvento(Utente,Profissional,Servico,Instituicao) -> {V,F}
registarEvento(Utente,Profissional,Servico,Instituicao) :- evolucao(registo(Utente,
    Profissional,Servico,Instituicao)).
```

3.2.11 *Remover utentes (ou profissionais, ou serviços, ou instituições) dos registos*

Invariantes para operações de remoção

Tal como referido na seção 3.2.9 é necessário preservar a correção da base de conhecimento. Ora, para as operações de remoção de conhecimento necessitamos de garantir:

- utente:
  - Invariante Estrutural:
    - não pode existir o utente depois da operação de remoção;
  - Invariante Referencial:
    - utentes apenas podem ser eliminado se não existirem registos de eventos médicos a ele associado;

  traduzido logicamente para:

```
-utente(Utente)::(
  solucoes( (Utente), ( utente(Utente) ), Lista),
  comprimento(Lista,N),
  N==0,
  solucoes( (Utente,_,_,_), ( registo(Utente,_,_,_) ), ListaReg),
  comprimento(ListaReg,NReg),
  NReg==0
  ).
```

- instituição:
  - Invariante Estrutural:
    - não pode existir a instituição depois da operação de remoção;
  - Invariante Referencial:
    - instituições apenas podem ser eliminadas se não existirem registos de eventos médicos a elas associadas;
    - instituições apenas podem ser eliminadas se não existirem serviços a elas associadas;
    - instituições apenas podem ser eliminadas se não existirem profissionais de saúde a elas associadas;

  traduzido logicamente para:

```
-instituicao(Instituicao)::(
  solucoes( (Instituicao), ( instituicao(Instituicao) ), Lista),
  comprimento(Lista,N),
  N==0,
  solucoes( (_,Instituicao,_,_), ( registo(_,Instituicao,_,_) ), ListaReg),
  comprimento(ListaReg,NReg),
  NReg==0,
  solucoes( (_,Instituicao) , ( servico(_,Instituicao) ), ListaServ),
  comprimento(ListaServ,NServ),
  NServ==0,
  solucoes( (_,_,Instituicao), ( profissional(_,_,Instituicao) ), ListaProf),
  comprimento(ListaProf,NProf),
  NProf==0
  ).
```

- serviço:
  - Invariante Estrutural:
    - não pode existir o serviço depois da operação de remoção;
  - Invariante Referencial:
    - serviços apenas podem ser eliminados se não existirem registos de eventos médicos a eles associados;
    - serviços apenas podem ser eliminados se não existirem profissionais de saúde a eles associados;

  traduzido logicamente para:

```
-servico(Servico,Instituicao)::(
  solucoes( (Servico,Instituicao) , ( servico(Servico,Instituicao) ), Lista),
  comprimento(Lista,N),
  N==0,
  solucoes( (_,Instituicao,Servico,_), ( registo(_,Instituicao,Servico,_) ),
      ListaReg),
  comprimento(ListaReg,NReg),
  NReg==0,
  solucoes( (_,Servico,Instituicao), ( profissional(_,Servico,Instituicao) ),
      ListaProf),
  comprimento(ListaProf,NProf),
  NProf==0
  ).
```

- profissional:
  - Invariante Estrutural:
    - não pode existir o profissional depois da operação de remoção;

- Invariante Referencial:
  - profissionais apenas podem ser eliminados se não existirem registos de eventos médicos a eles associados;

traduzido logicamente para:

```
-profissional(Profissional,Servico,Instituicao)::(
  solucoes( (Profissional), ( profissional(Profissional,Servico,Instituicao) )
      , Lista),
  comprimento(Lista,N),
  N==0,
  solucoes( (_,Instituicao,Servico,Profissional), ( registo(_,Instituicao,
      Servico,Profissional) ), ListaReg),
  comprimento(ListaReg,NReg),
  NReg==0
  ).
```

Predicados para operações de remoção

Definidos os invariantes para as operações de remoção à base de conhecimento, resta-nos definir as extensões de predicados que permitem remover utentes, profissionais, serviços e instituições.

Denote que as extensões de predicados seguidamente definidas recorrem ao predicado **remocao(Termo)**, analogamente inverso ao predicado **evolucao(Termo)** e possível de ser consultado na seção A.4.

```
% 10) Remover utentes (ou profissionais ou servicos ou instituicoes) dos registos

% Extensao do predicado que pemite remover utentes
% removerUtente(Nome) -> {V,F}
removerUtente(Nome) :- remocao(utente(Nome)).

% Extensao do predicado que pemite remover instituicoes
% removerInstituicao(Instituicao) -> {V,F}
removerInstituicao(Instituicao) :- remocao(instituicao(Instituicao)).

% Extensao do predicado que pemite remover servicos numa instituicao
% registarServico(Servico,Instituicao) -> {V,F}
removerServico(Servico,Instituicao) :- remocao(servico(Servico,Instituicao)).

% Extensao do predicado que pemite remover profissionais num determinado servico de
     numa instituicao
% removerProfissional(Profissional,Servico,Instituicao) -> {V,F}
removerProfissional(Profissional,Servico,Instituicao) :- remocao(profissional(
    Profissional,Servico,Instituicao)).

% Extensao do predicado que pemite remover eventos medicos numa instituicao
    indicando o utente, o profissional, o servico e a instituicao
% removerEvento(Utente,Profissional,Servico,Instituicao) -> {V,F}
removerEvento(Utente,Profissional,Servico,Instituicao) :- remocao(registo(Utente,
    Profissional,Servico,Instituicao)).
```

3.3 FUNCIONALIDADES ADICIONAIS

Em adição às funcionalidades obrigatórias foram incluídas novas funcionalidades no sistema que permitem:

- E1) determinar o número de utentes de uma instituição;
- E2.0.1) dada uma lista de instituições determinar a lista de utilização em número de utentes para cada instituição;
- E2) determinar a lista de utilização em número de utentes de todas as instituições na base de conhecimento;
- E3.0.1) determinar o número de eventos médicos de um utente;
- E3) determinar todos os eventos médicos de um utente;
- E4.0.1) dada uma lista de utentes determinar a lista do número de eventos médicos para cada utente;
- E4) determinar a lista de número de eventos médicos por utente de todos os utentes na base de conhecimento;
- E5.0.1) determinar todos os eventos médicos de um profissional de saúde;
- E5) determinar o número de eventos médicos em que um profissional esteve envolvido;
- E6.0.1) dada uma lista de profissionais determinar a lista do número de eventos médicos em que cada um esteve envolvido;
- E6) determinar a lista de número de eventos médicos por utente de todos os profissionais na base de conhecimento;
- E7.0.1) determinar todos os eventos médicos de um serviço;
- E7) determinar o número de eventos médicos em que um serviço esteve envolvido;
- E8.0.1) dada uma lista de serviços determinar a lista do número de eventos médicos em que cada um esteve envolvido;
- E8) determinar a lista de número de eventos médicos por serviço de todos os serviços na base de conhecimento;

Dado que as restantes extensões de predicados são invocadas pelos predicados com as funcionalidades E2, E4, E6, E8, iremos dedicar especial atenção a estes, incluindo para cada um a sua extensão e respectivos testes. Os restantes predicados poderão ser consultados integralmente na seção A.5.

3.3.1 *Determinar a lista de utilização em número de utentes de todas as instituições na base de conhecimento*

```
% E2) Extensao do predicado que permite determinar a lista de utilizacao em numero
    de utentes de todas as instituicoes na base de conhecimento.
%mapaUtilizacaoInstituicoes(ListaUtilizacao) -> {V,F}.
mapaUtilizacaoInstituicoes(ListaUtilizacao) :-.
  solucoes( (Instituicao),( instituicao(Instituicao) ), ListaInstituicoes),
  listaUtilizacaoInstituicoes(ListaInstituicoes,ListaUtilizacao).
```

*Teste do predicado **mapaUtilizacaoInstituicoes(ListaUtilizacao)**:*

```
| ?- mapaUtilizacaoInstituicoes(Mapa).
Mapa = [[hospital_sao_marcos,1],[hospital_braga,0],[hospital_lisboa,0],[
    hospital_porto,5],[hospital_leiria,0]] ?
yes
```

3.3.2 *Determinar a lista de número de eventos médicos por utente de todos os utentes na base de conhecimento*

```
% E4) Extensao do predicado que permite determinar a lista de numero de eventos
    medicos por utente de todos os utentes na base de conhecimento.
%mapaEventosMedicosPorUtente(ListaEventosUtente) -> {V,F}.
mapaEventosMedicosPorUtente(ListaEventosUtente) :-.
  solucoes( (Utente),( utente(Utente) ), ListaUtentes),
  listaEventosMedicosUtente(ListaUtentes,ListaEventosUtente).
```

*Teste do predicado **mapaEventosMedicosPorUtente(ListaEventosUtente)**:*

```
| ?- mapaUtilizacaoInstituicoes(Mapa).
| ?- mapaEventosMedicosPorUtente(Mapa).
Mapa = [[antonio_sousa,3],[antonio_marques,1],[maria_meireles,2],[
    diamantino_marques,2],[delfina_araujo,0],[jorge_marques,1],[rosa_sousa,1]] ?
yes
```

3.3.3 *Determinar a lista de número de eventos médicos por utente de todos os profissionais na base de conhecimento*

```
% E6) Extensao do predicado que permite determinar a lista de numero de eventos
    medicos por utente de todos os profissionais na base de conhecimento.
%mapaEventosMedicosPorProfissional(ListaEventosProfissional) -> {V,F}.
mapaEventosMedicosPorProfissional(ListaEventosProfissional) :-.
  solucoes( (Profissional),( profissional(Profissional,_,_) ),
      ListaProfissionaisRep),
  removerduplicados(ListaProfissionaisRep, ListaProfissionais),
  listaEventosMedicosProfissional(ListaProfissionais,ListaEventosProfissional).
```

Teste do predicado ***mapaEventosMedicosPorProfissional(ListaEventosProfissional)****:*

```
| ?- mapaEventosMedicosPorProfissional(Mapa).
Mapa = [[salvador_sousa,0],[andre_santos,6],[tiago_sousa,0],[vanessa_goncalves,2],[
    marta_caetano,0],[filipe_oliveira,1],[filipe_marques,1],[luis_mendes,0],[
    luis_sousa,0],[...|...]] ?
yes
```

3.3.4 *Determinar a lista de número de eventos médicos por serviço de todos os serviços na base de conhecimento*

```
% E8) Extensao do predicado que permite determinar a lista de numero de eventos
    medicos por servico de todos os servicos na base de conhecimento.
%mapaEventosMedicosPorServico(ListaEventosServico) -> {V,F}.
mapaEventosMedicosPorServico(ListaEventosServico) :-.
  solucoes( (Servico),( servico(Servico,_) ), ListaServicosRep),
  removerduplicados(ListaServicosRep, ListaServicos),
  listaEventosMedicosServico(ListaServicos,ListaEventosServico).
```

Teste do predicado ***mapaEventosMedicosPorServico(ListaEventosServico)****:*

```
| ?- mapaEventosMedicosPorServico(Mapa).
Mapa = [[cardiologia,2],[nutricionismo,2],[geriatria,6],[neurologia,0],[oncologia
    ,0],[cirurgia,0],[clinica_geral,0],[psiquiatria,0]] ?
yes
```

4

CONCLUSÃO

O presente trabalho de grupo, em todas as suas fases teve como principal propósito a familiarização da utilização da linguagem de programação em lógica PROLOG, como forma de conhecimento e construção de mecanismos de raciocínio para a resolução de problemas. Tal objectivo foi cumprido.

Apesar do modelo apresentado possuir limitações dada a sua simplicidade, pode concluir-se que garante a correcta evolução da base de conhecimento, possibilitando um suporte base sólido para o desenvolvimento de funcionalidades adicionais. O modelo proposto pode servir como um ponto de partida para o desenvolvimento de um sistema mais completo, capaz de verdadeiramente auxiliar os gestores de unidades de cuidados de saúde profissionais, assim como na prática clínica por parte dos profissionais.

A

ANEXOS

A.1 EXEMPLO PRÁTICO DE BASE DE CONHECIMENTO

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Base de Conhecimento sobre Utentes
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

utente(antonio_sousa).
utente(antonio_marques).
utente(maria_meireles).
utente(diamantino_marques).
utente(delfina_araujo).
utente(jorge_marques).
utente(rosa_sousa).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Base de Conhecimento sobre Instituicoes
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

instituicao(hospital_sao_marcos).
instituicao(hospital_braga).
instituicao(hospital_lisboa).
instituicao(hospital_porto).
instituicao(hospital_leiria).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Base de Conhecimento sobre Servicos
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

servico(cardiologia,hospital_sao_marcos).
servico(cardiologia,hospital_braga).
servico(cardiologia,hospital_leiria).
servico(cardiologia,hospital_porto).

servico(nutricionismo,hospital_sao_marcos).
servico(nutricionismo,hospital_braga).
servico(nutricionismo,hospital_leiria).
servico(nutricionismo,hospital_porto).

servico(geriatria,hospital_porto).
```

```
servico(neurologia,hospital_porto).

servico(oncologia,hospital_porto).

servico(cirurgia,hospital_braga).
servico(cirurgia,hospital_lisboa).

servico(clinica_geral,hospital_braga).
servico(clinica_geral,hospital_porto).

servico(psiquiatria,hospital_braga).

%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Base de Conhecimento sobre Profissionais
%%%%%%%%%%%%%%%%%%%%%%%%%%%

profissional(salvador_sousa, oncologia, hospital_porto).
profissional(filipe_oliveira, nutricionismo, hospital_porto).
profissional(filipe_marques, nutricionismo, hospital_porto).
profissional(filipe_oliveira, clinical_geral, hospital_porto).
profissional(filipe_marques, clinical_geral, hospital_porto).
profissional(luis_mendes, clinica_geral, hospital_porto).
profissional(andre_santos, geriatria, hospital_porto).

profissional(tiago_sousa, cirurgia, hospital_lisboa).

profissional(vanessa_goncalves, cardiologia, hospital_sao_marcos).
profissional(marta_caetano, nutricionismo, hospital_sao_marcos).

profissional(filipe_oliveira, nutricionismo, hospital_braga).
profissional(filipe_marques, nutricionismo, hospital_braga).
profissional(luis_mendes, nutricionismo, hospital_braga).
profissional(luis_mendes, clinical_geral, hospital_braga).
profissional(luis_sousa, clinica_geral, hospital_braga).
profissional(andreia_goncalves, cirurgia, hospital_braga).

%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Base de Conhecimento sobre Registo de entradas
%%%%%%%%%%%%%%%%%%%%%%%%%%%

registo(antonio_sousa, hospital_sao_marcos, cardiologia, vanessa_goncalves).
registo(antonio_sousa, hospital_sao_marcos, cardiologia, vanessa_goncalves).
registo(antonio_sousa, hospital_sao_marcos, nutricionismo, filipe_oliveira).
registo(antonio_marques, hospital_porto, nutricionismo, filipe_marques).
registo(maria_meireles, hospital_porto, geriatria, andre_santos).
registo(maria_meireles, hospital_porto, geriatria, andre_santos).
registo(diamantino_marques, hospital_porto, geriatria, andre_santos).
registo(diamantino_marques, hospital_porto, geriatria, andre_santos).
registo(rosa_sousa, hospital_porto, geriatria, andre_santos).
registo(jorge_marques, hospital_porto, geriatria, andre_santos).
```

A.2 EXTENSÃO DO PREDICADOS AUXILIARES

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

% Verifica se elemento existe dentro de uma lista de elementos
pertence(X,[X | _ ]).
pertence(X,[ _ | XS]) :- pertence(X,XS).

% Nr de elementos existentes numa lista
comprimento([],0).
comprimento([_ | L],R) :- comprimento(L,N),R is N+1.

% Apaga a primeira ocorrencia de um elemento numa lista
apagar(X, [X | XS], XS).
apagar(E, [X | XS], [X | YS]) :- apagar(E, XS, YS).

% Apaga todas as ocorrencias de um elemento numa lista
apagartudo(_, [], []).
apagartudo(X,[X | XS], YS) :- apagartudo(X,XS,YS).
apagartudo(E,[X | XS], [X | YS]) :- apagartudo(E, XS, YS).

% Insere elemento a cabeca da lista, caso ainda nao exista
adicionar(X, L, L) :- pertence(X,L).
adicionar(X, L, [X | L]).

% Concatenacao da lista L1 com lista L2
concatenar([], L2, L2).
concatenar([X | L1], L2, [X | R]) :- concatenar(L1, L2, R).

% Inverte ordem dos elementos de uma lista
inverter([X],[X]).
inverter([X | XS], L2) :- inverter(XS, YS), concatenar(YS,[X],L2).

% Verifica se S e sublista de L
sublista(S,L) :- concatenar(S,_,L).
sublista(S,L) :- concatenar(_,S,L).
sublista(S, [ _ | YS]) :-
  sublista(S, YS).

% Remove elementos duplicados de uma lista
removerduplicados([],[]).
removerduplicados([H|T],C) :- pertence(H,T), !, removerduplicados(T,C).
removerduplicados([H|T],[H|C]) :- removerduplicados(T,C).

% Subtrai elementos de L1 a L2, produzindo L3
intercepcao([], L, L).
intercepcao([H | Tail], L2, L3) :- apagar(H, L2, R), intercepcao(Tail, R, L3).
```

A.3 Extensão do predicado que permite a evolução de conhecimento

```
% Extensao do predicado que permite a evolucao do conhecimento
% disponibilizada pelo professor na aula pratica da semana5
evolucao( Termo ):-solucoes(Invariante,+Termo::Invariante,Lista),
inserir(Termo),
testar(Lista).

% predicado disponibilizado pelo professor na semana5
% inserir: T -> {V,F}
inserir(Termo):-assert(Termo).
inserir(Termo):-retract(Termo),!,fail.

% predicado disponibilizado pelo professor na semana5
% testar: Li -> {V,F}.
testar([]).
testar([I|L]):-I,testar(L).

% Extensao do predicado que permite a remocao do conhecimento
% remocao(Termo) -> {V,F}
remocao( Termo ):-solucoes(Invariante,-Termo::Invariante,Lista),
remover(Termo),
testar(Lista).

% remover: T -> {V,F}
remover(Termo):-retract(Termo).
remover(Termo):-assert(Termo),!,fail.

% predicado disponibilizado pelo professor na semana5
% solucoes X,Y,Z -> {V,F}
solucoes(X,Y,Z):-findall(X,Y,Z).
```

A.4 Extensão do predicado que permite a remoção de conhecimento

```
% Extensao do predicado que permite a remocao do conhecimento
% remocao(Termo) -> {V,F}
remocao( Termo ):-solucoes(Invariante,-Termo::Invariante,Lista),
remover(Termo),
testar(Lista).

% remover: T -> {V,F}
remover(Termo):-retract(Termo).
remover(Termo):-assert(Termo),!,fail.
```

A.5 EXTENSÃO DE PREDICADOS DE FUNCIONALIDADES ADICIONAIS

```
% E1) Extensao do predicado que permite determinar o numero de utentes de uma
    instituicao
% quantosUtentesInstituicao(Instituicao,NumeroUtentes) -> {V,F}
quantosUtentesInstituicao(Instituicao,NumeroUtentes) :-
  utentesInstituicao(Instituicao,ListaUtentes),
  comprimento(ListaUtentes,NumeroUtentes).

% E2.0.1) Extensao do predicado dada uma lista de instituicoes determina a lista de
     utilizacao em numero de utentes para cada instituicao
%listaUtilizacaoInstituicoes( [I],  [[I,N]]) -> {V,F}
listaUtilizacaoInstituicoes( [I],  [[I,N]]) :-
  quantosUtentesInstituicao(I,N).
listaUtilizacaoInstituicoes( [I|TAIL_INST],  [[I,N] | XS]) :-
  quantosUtentesInstituicao(I,N),
  listaUtilizacaoInstituicoes(TAIL_INST,XS).

% E2) Extensao do predicado que permite determinar a lista de utilizacao em numero
    de utentes de todas as instituicoes na base de conhecimento
%utilizacaoInstituicoes(ListaUtilizacao) -> {V,F}
mapaUtilizacaoInstituicoes(ListaUtilizacao) :-
  solucoes( (Instituicao),( instituicao(Instituicao) ), ListaInstituicoes),
  listaUtilizacaoInstituicoes(ListaInstituicoes,ListaUtilizacao).

% E3) Extensao do predicado que permite determinar o numero de eventos medicos de
    um utente
% quantosEventosUtente(Utente,NumeroEventos) -> {V,F}
quantosEventosMedicosUtente(Utente,NumeroEventos) :-
  eventosMedicosUtente(Utente,ListaEventosMedicos),
  comprimento(ListaEventosMedicos,NumeroEventos).

% E3.0.1) Extensao do predicado que permite identificar todos os eventos medicos de
     um utente
eventosMedicosUtente(Utente,ListaEventosMedicos) :-
  solucoes( (Utente), ( registo(Utente,_,_,_) ), ListaEventosMedicos).

% E4.0.1) Extensao do predicado dada uma lista de utentes determina a lista do
    numero de eventos medicos para cada utentes
%listaEventosMedicosUtente( [Utentes],  [[Utente,N]]) -> {V,F}
listaEventosMedicosUtente( [Utente],  [[Utente,N]]) :-
  quantosEventosMedicosUtente(Utente,N).
listaEventosMedicosUtente( [Utente|TAIL_UT],  [[Utente,N] | XS]) :-
  quantosEventosMedicosUtente(Utente,N),
  listaEventosMedicosUtente(TAIL_UT,XS).

% E4) Extensao do predicado que permite determinar a lista de numero de eventos
    medicos por utente de todos os utentes na base de conhecimento
%mapaEventosMedicosUtentes(ListaEventos) -> {V,F}
mapaEventosMedicosPorUtente(ListaEventosUtente) :-
  solucoes( (Utente),( utente(Utente) ), ListaUtentes),
  listaEventosMedicosUtente(ListaUtentes,ListaEventosUtente).
```

```
% E5) Extensao do predicado que permite determinar o numero de eventos medicos em
    que um profissional esteve envolvido
% quantosEventosMedicosProfissional(Profissional,NumeroEventos) -> {V,F}
quantosEventosMedicosProfissional(Profissional,NumeroEventos) :-
  eventosMedicosProfissional(Profissional,ListaEventosMedicos),
  comprimento(ListaEventosMedicos,NumeroEventos).

% E5.0.1) Extensao do predicado que permite identificar todos os eventos medicos de
     um profissional
eventosMedicosProfissional(Profissional,ListaEventosMedicos) :-
  solucoes( (Profissional), ( registo(_,_,_,Profissional) ), ListaEventosMedicos).

% E6.0.1) Extensao do predicado dada uma lista de profissionais determina a lista
    do numero de eventos medicos em que cada um esteve envolvido
%listaEventosMedicosProfissional( [Profissionais],  [[Profissional,N]]) -> {V,F}
listaEventosMedicosProfissional( [Profissional],  [[Profissional,N]]) :-
  quantosEventosMedicosProfissional(Profissional,N).
listaEventosMedicosProfissional( [Profissional|TAIL_PF],  [[Profissional,N] | XS])
    :-
  quantosEventosMedicosProfissional(Profissional,N),
  listaEventosMedicosProfissional(TAIL_PF,XS).

% E6) Extensao do predicado que permite determinar a lista de numero de eventos
    medicos por utente de todos os profissionais na base de conhecimento
%mapaEventosMedicosPorProfissional(ListaEventosProfissional) -> {V,F}
mapaEventosMedicosPorProfissional(ListaEventosProfissional) :-
  solucoes( (Profissional),( profissional(Profissional,_,_) ),
      ListaProfissionaisRep),
  removerduplicados(ListaProfissionaisRep, ListaProfissionais),
  listaEventosMedicosProfissional(ListaProfissionais,ListaEventosProfissional).

% E7) Extensao do predicado que permite determinar o numero de eventos medicos em
    que um servico esteve envolvido
% quantosEventosMedicosServico(Servico,NumeroEventos) -> {V,F}
quantosEventosMedicosServico(Servico,NumeroEventos) :-
  eventosMedicosServico(Servico,ListaEventosMedicos),
  comprimento(ListaEventosMedicos,NumeroEventos).

% E7.0.1) Extensao do predicado que permite identificar todos os eventos medicos de
     um servico
eventosMedicosServico(Servico,ListaEventosMedicos) :-
  solucoes( (Servico), ( registo(_,_,Servico,_) ), ListaEventosMedicos).

% E8.0.1) Extensao do predicado dada uma lista de servicos determina a lista do
    numero de eventos medicos em que cada um esteve envolvido
%listaEventosMedicosServico( [Servicos],  [[Servico,N]]) -> {V,F}
listaEventosMedicosServico( [Servico],  [[Servico,N]]) :-
  quantosEventosMedicosServico(Servico,N).
listaEventosMedicosServico( [Servico|TAIL_SERV],  [[Servico,N] | XS]) :-
  quantosEventosMedicosServico(Servico,N),
  listaEventosMedicosServico(TAIL_SERV,XS).

% E8) Extensao do predicado que permite determinar a lista de numero de eventos
    medicos por servico de todos os servicos na base de conhecimento
```

```
%mapaEventosMedicosPorServico(ListaEventosServico) -> {V,F}
mapaEventosMedicosPorServico(ListaEventosServico) :-
  solucoes( (Servico),( servico(Servico,_) ), ListaServicosRep),
  removerduplicados(ListaServicosRep, ListaServicos),
  listaEventosMedicosServico(ListaServicos,ListaEventosServico).
```