UNIVERSIDADE DO MINHO

Exercício 1

Mestrado Integrado em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio

(2º Semestre / 2018-2019)

A81047 Catarina Machado

A86264 Gonçalo Faria

A82339 João Vilaça

Braga,

Março de 2019

Resumo

O presente relatório é constituído por uma breve descrição e demonstração do trabalho realizado no âmbito do Exercício 1 da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio.

Assim sendo, ao longo deste documento será apresentada toda a base do conhecimento e predicados que a constituem e, em seguida, os invariantes estruturais e referenciais que permitem garantir uma correta evolução da base de conhecimento, com adição de novo conhecimento, que também será aqui descrito.

Nas secções seguintes serão ainda analisados os tópicos de remoção de conhecimento, identificações de predicados através de alguns critérios e por fim, dentro dos tópicos obrigatórios, o cálculo de alguns custos.

Finalmente, serão apresentadas todas as funcionalidades extras que implementamos neste exercício.

Índice

Resumo ..... *ii*

Índice ..... *iii*

Índice de Figuras ..... *iv*

1. Introdução *7*

2. Preliminares *8*

3. Descrição do Trabalho e Análise de Resultados *9*

3.1. Base de Conhecimento **9**

3.2. Invariantes **12**

3.2.1. Invariantes de utente 12
3.2.2. Invariantes de serviço 13
3.2.3. Invariantes de consulta 14
3.2.4. Invariantes de médico 14
3.2.5. Invariantes de enfermeiro 15
3.2.6. Invariantes de médico e enfermeiro da família 16
3.2.7. Invariantes de exame 16

3.3. Adicionar conhecimento **17**

3.3.1. Registar utentes, serviços, consultas, médicos, enfermeiros, médicos e enfermeiros da família e exames 17

3.4. Remover conhecimento **20**

3.4.1. Remover utentes, serviços, consultas, médicos, enfermeiros, médicos e enfermeiros da família e exames 20

3.5. Identificações **24**

3.5.1. Identificar as instituições prestadoras de serviços 24
3.5.2. Identificar utentes/serviços/consultas/médicos/enfermeiros/médico e enfermeiro da família/exames por critérios de seleção 25
3.5.3. Identificar serviços prestados por instituição/cidade/datas/custo 28
3.5.4. Identificar os utentes de um serviço/instituição 31
3.5.5. Identificar serviços realizados por utente/instituição/cidade 33

3.6. Custos **34**

3.6.1. Calcular o custo total dos cuidados de saúde por utente/serviço/instituição/data 34

3.7. Mais extras **37**

3.7.1. Modularidade de Código 37
3.7.2. Guardar Estado 38
3.7.3. Ordenar por quem gastou mais dinheiro 41

4. Conclusões e Sugestões *42*

5. Referências *43*

5.1. Referências Bibliográficas **43**

5.2. Referências Eletrónicas **44**

6. Anexos *45*

6.1. Funções Auxiliares **45**

Índice de Figuras

Figura 1 - Base de conhecimento de utente 10
Figura 2 - Base de conhecimento de serviço 10
Figura 3 - Base de conhecimento de consulta 11
Figura 4 - Base de conhecimento de médico 11
Figura 5 - Base de conhecimento de enfermeiro 11
Figura 6 - Base de conhecimento de médico e enfermeiro da família 11
Figura 7 - Base de conhecimento de exame 11
Figura 8 - Invariante de utente 12
Figura 9 - Invariante de utente 12
Figura 10 - Invariante de utente 12
Figura 11 - Invariante de utente 12
Figura 12 - Invariante de utente 13
Figura 13 - Invariante de utente 13
Figura 14 - Invariante de serviço 13
Figura 15 - Invariante de serviço 13
Figura 16 - Invariante de serviço 13
Figura 17 - Invariante de serviço 13
Figura 18 - Invariante de serviço 13
Figura 19 - Invariante de consulta 14
Figura 20 - Invariante de consulta 14
Figura 21 - Invariante de consulta 14
Figura 22 - Invariante de consulta 14
Figura 23 - Invariante de consulta 14
Figura 24 - Invariante de médico 14
Figura 25 - Invariante de médico 15
Figura 26 - Invariante de médico 15
Figura 27 - Invariante de médico 15
Figura 28 - Invariante de médico 15
Figura 29 - Invariante de enfermeiro 15
Figura 30 - Invariante de enfermeiro 15
Figura 31 - Invariante de enfermeiro 15
Figura 32 - Invariante de enfermeiro 15
Figura 33 - Invariante de enfermeiro 16
Figura 34 - Invariante de médico e enfermeiro da família 16
Figura 35 - Invariante de médico e enfermeiro da família 16
Figura 36 - Invariante de médico e enfermeiro da família 16
Figura 37 - Invariante de médico e enfermeiro da família 16
Figura 38 - Invariante de médico e enfermeiro da família 16
Figura 39 - Invariante de exame 16
Figura 40 - Predicado 'novoUtente' 17
Figura 41 - Exemplificações de +invariantes de utente 17
Figura 42 - Exemplificação de +invariantes de utente 18

Figura 43 - Predicado 'novoServico' 18
Figura 44 - Exemplificações de +invariantes de serviço 18
Figura 45 - Predicado 'novaConsulta' 19
Figura 46 - Exemplificações de +invariantes de consulta 19
Figura 47 - Predicado 'novoMedico' 19
Figura 48 - Predicado 'novoEnfermeiro' 19
Figura 49 - Predicado 'novoMedEnfFamilia' 20
Figura 50 - Predicado 'novoExame' 20
Figura 51 - Predicado 'removeUtente' 20
Figura 52 - Exemplificações de -invariantes de utente 21
Figura 53 - Exemplificação de -invariantes de utente 21
Figura 54 - Predicado 'removeServico' 22
Figura 55 - Exemplificação de -invariantes de serviço 22
Figura 56 - Predicado 'removeConsulta' 22
Figura 57 - Exemplificação de -invariantes de consulta 23
Figura 58 - Predicado 'removeMedico' 23
Figura 59 - Exemplificação de -invariantes de medico 23
Figura 60 - Predicado 'removeEnfermeiro' 24
Figura 61 - Predicado 'removeMedEnfFamilia' 24
Figura 62 - Predicado 'removeExame' 24
Figura 63 - Predicado 'identificaInstituicoes' 24
Figura 64 - Exemplo 'identificaInstituicoes' 25
Figura 65 - Predicados utente por critérios de seleção 25
Figura 66 - Exemplo 'utenteCidade' 26
Figura 67 - Predicados serviço por critérios de seleção 26
Figura 68 - Predicados consulta por critérios de seleção 26
Figura 69 - Predicados médico por critérios de seleção 27
Figura 70 - Predicados enfermeiro por critérios de seleção 27
Figura 71 - Exemplo 'medicoGenero' 27
Figura 72 - Predicados médico e enfermeiro da família por critérios de seleção 28
Figura 73 - Predicados exame por critérios de seleção 28
Figura 74 - Predicado 'consultasPorInstituicao' 28
Figura 75 - Predicado 'consultasPorCidade' 29
Figura 76 – Predicado 'servicosParaConsultasServicos' 29
Figura 77 - Exemplo 'consultasPorCidade' 29
Figura 78 - Predicado 'servicosPorData' 30
Figura 79 - Predicado 'servicosPorCusto' 30
Figura 80 - Exemplo 'servicosPorCusto' 30
Figura 81 - Predicado 'utentesServico' 31
Figura 82 - Exemplo 'utentesServico' 31
Figura 83 - Predicado 'utentesInstituicao' 32
Figura 84 - Exemplo 'utentesInstituicao' 32
Figura 85 - Predicado 'servicosPorUtente' 33
Figura 86 - Exemplo 'servicosPorUtente' 33

Figura 87 - Predicado 'servicoPorInstituicao' 34
Figura 88 - Predicado 'servicoPorCidade' 34
Figura 89 - Predicado 'totalUtente' 34
Figura 90 - Exemplo 'totalUtente' 35
Figura 91 - Predicado 'totalServico' 35
Figura 92 - Exemplo 'totalServico' 35
Figura 93 - Predicado 'totalInstituicao' 36
Figura 94 - Exemplo 'totaInstituicao' 36
Figura 95 - Predicado 'totalData' 36
Figura 96 - 'Includes' presentes no ficheiro exercicio1.pl 38
Figura 97 - Predicado 'saveState' 38
Figura 98 - Predicados auxiliares de 'saveState' 39
Figura 99 - Exemplo 'saveState' 40
Figura 100 - Predicado 'utentesComMaisDinheiroGasto' 41
Figura 101 - Exemplo ''utentesComMaisDinheiroGasto'' 41
Figura 102 – Meta-predicado 'solucoes' 45
Figura 103 - Predicado 'comprimento' 45
Figura 104 – Meta-predicado 'insercao' 45
Figura 105 – Meta-predicado 'remocao' 46
Figura 106 – Meta-predicado 'teste' 46
Figura 107 – Meta-predicado 'evolucao' 46
Figura 108 – Meta-predicado 'involucao' 46
Figura 109 - Predicado 'pertence' 47
Figura 110 - Predicado 'eliminaRepetidos' 47
Figura 111 - Predicado 'soma' 47
Figura 112 - Predicado 'concat' 47
Figura 113 - Predicado 'mergeSortKeyValuePairASC' 48
Figura 114 - Predicado 'take' 48
Figura 115 - Predicado 'idadeValida', 'custoValido' e 'generoValido' 48

1. Introdução

Este exercício teve como principal objetivo motivar-nos para a utilização da linguagem de programação em lógica PROLOG, no âmbito da representação de conhecimento e construção de mecanismos de raciocínio para a resolução de problemas.

Durante o exercício foi desenvolvido um sistema para caracterizar um universo de discurso na área da prestação de cuidados de saúde que incluía, por exemplo, conhecimento de utentes, consultas, exames, entre outros.

Este sistema tinha como requisitos a implementação de diversas funcionalidades, apresentadas ao longo deste relatório, que permitem uma correta manipulação da base de conhecimento (quer adição quer remoção de conhecimento) e a aplicação de uma série de *queries* sobre ela.

2. Preliminares

Antes de iniciar o desenvolvimento deste exercício a equipa teve de estudar duas grandes matérias que permitiram que a implementação deste sistema de representação de conhecimento e raciocínio decorresse sem problemas nem sobressaltos.

Por um lado, numa vertente mais técnica, a equipa estudou em detalhe o motor de inferência PROLOG, não só com a presença nas aulas teóricas e práticas da unidade curricular em que este exercício se insere. Foi também essencial a análise da documentação do SICStus Prolog, bem como a leitura de artigos encontrados na Internet e em livros.

Por outro lado, foi feita uma análise geral de como funciona o Serviço Nacional de Saúde português para que a equipa pudesse fazer paralelismos entre o sistema que estava a desenvolver e a realidade para que as funcionalidades que foram implementadas fizessem sentido num contexto real e não fossem apenas uma série de implementações sem nenhuma utilidade prática.

3. Descrição do Trabalho e Análise de Resultados

Para uma melhor exposição do trabalho realizado, dividimos o mesmo em oito diferentes partes. A primeira parte é a **Base de Conhecimento**, onde descrevemos os predicados que constituem a base de conhecimento do nosso sistema de representação de conhecimento e raciocínio.

Em seguida, referimos os **Invariantes** estruturais e referenciais que controlam a inserção e remoção de conhecimento, a **Adição de Conhecimento** e a **Remoção de Conhecimento**, onde apresentamos os procedimentos construídos para o efeito, devidamente ilustrados com simples exemplos práticos de aplicação.

Posteriormente, abordamos os predicados relativos às várias funcionalidades de **Identificações** e **Custos** requeridos, bem como vários exemplos práticos dos mesmos e, por fim, os vários **Extras** que foram incluídos no nosso sistema, quer ao nível das capacidades de representação de conhecimento, quer ao nível das faculdades de raciocínio.

Em anexo apresentamos as **Funções Auxiliares**, onde se encontram as funções que nos auxiliaram na construção dos predicados pedidos.

Ao longo da presente secção, fomos descrevendo o trabalho e simultaneamente analisando os resultados obtidos.

3.1. Base de Conhecimento

De modo a que o nosso sistema de representação de conhecimento relativo à área da prestação de cuidados de saúde seja melhor caracterizado, decidimos estender o conhecimento proposto adicionando novos predicados.

Assim, para além do **utente**, do **serviço** e da **consulta**, temos também o **médico**, o **enfermeiro**, o médico e enfermeiro da família **(medEnfFamilia)** e o **exame**.

É de salientar também que os predicados **utente** e **consulta** sofreram pequenas alterações comparativamente ao exemplo do enunciado. O **utente** passou a conter o género e o id da família em que se insere, e a **consulta** passou a ser identificada por um id.

Cada um dos predicados mencionados foi definido da seguinte forma:

---

utente: #IdUtente, Nome, Idade, Género, #IdFamília, Cidade -> {V,F}

O utente é identificado por um id único e intransmissível, o seu nome, idade, género ('M' ou 'F'), o id da família a que pertence e a cidade onde vive.

---

serviço: #IdServiço, Descrição, Instituição, Cidade -> {V,F}

O serviço é caracterizado por um id, uma breve descrição, a instituição onde é realizado e a respetiva cidade.

consulta: #IdConsulta, Data, #IdUtente, #IdServiço, Custo -> {V,F}

A consulta possui também um id, a data em que se realizou, o id do utente que compareceu, o id do serviço a que a consulta se referiu e o custo da mesma.

medico: #IdMédico, Nome, Idade, Género, #IdServiço -> {V,F}

O médico é caracterizado por um id, nome, idade, género ('M' ou 'F') e pelo id do serviço que realiza.

enfermeiro: #IdEnfermeiro, Nome, Idade, Género, #IdServiço -> {V,F}

O enfermeiro é caracterizado por um id, nome, idade, género ('M' ou 'F') e pelo id do serviço que realiza.

medEnfFamilia: #IdFamília, #IdMédico, #IdEnfermeiro -> {V,F}

Tendo em consideração que o serviço nacional de saúde infere que cada família deverá ter um médico e um enfermeiro de família, o predicado "medEnfFamilia" é constituído pelo id da família e pelo id do médico e do enfermeiro responsável pela mesma.

exame: #IdConsulta, Descrição -> {V,F}

Uma vez que numa consulta pode ser realizado um determinado exame, o predicado exame caracterizado pelo id da consulta onde o mesmo se sucedeu e pela respetiva descrição.

Assim, tendo o panorama de conhecimento bem definido, povoamos o nosso sistema acrescentando factos relativos a todos os predicados mencionados anteriormente.

```
% utente: #IdUt,Nome,Idade,Género,#IdFamilia,Cidade -> {V,F}
utente(1,'Carlos',2,'M',1,'Porto').
utente(2,'Joao',20,'M',1,'Braga').
utente(3,'Vilaca',20,'M',2,'Braga').
utente(4,'Catarina',20,'F',3,'Braga').
utente(5,'Eva',0,'F',3,'Braga').
utente(6,'Maria Jose',56,'F',1,'Guimaraes').
utente(7,'Gustavo',28,'M',2,'Lisboa').
utente(8,'Helena',25,'F',5,'Lisboa').
utente(9,'Guilherme',58,'M',1,'Porto').
utente(10,'Ana',34,'F',6,'Guimaraes').
```

Figura 1 - Base de conhecimento de utente

```
servico(1,'Pediatria','Hospital Santa Maria','Porto').
servico(2,'Ortopedia','Hospital de Braga','Braga').
servico(3,'Oftalmologia','Hospital Santa Maria','Porto').
servico(4,'Cardiologia','Hospital de Braga','Braga').
servico(5,'Cirurgia Geral','Hospital Egas Moniz','Lisboa').
servico(6,'Estomatologia','Hospital Egas Moniz','Lisboa').
servico(7,'Genetica Medica','Hospital Pulido Valente','Lisboa').
servico(8,'Medicina Nuclear','Hospital Pulido Valente','Lisboa').
servico(9,'Curativo','Hospital Egas Moniz','Lisboa').
servico(10,'Vacina','Hospital de Braga','Braga').
servico(11,'Preparar para exame','Hospital Santa Maria','Porto').
```

Figura 2 - Base de conhecimento de serviço

```
% consulta: #IdConsulta,Data,#IdUt,#IdServ,Custo -> {V,F}
consulta(1,'01-01-2018',1,1,10).
consulta(2,'23-05-2016',1,1,14).
consulta(3,'05-02-2019',2,4,15).
consulta(4,'13-12-2018',2,1,20).
consulta(5,'09-09-2017',3,5,10).
consulta(6,'26-10-2016',1,3,15).
consulta(7,'29-02-2017',3,3,30).
consulta(8,'10-03-2016',4,6,20).
consulta(9,'02-04-2017',2,2,40).
consulta(10,'04-04-2017',5,7,10).
consulta(11,'12-12-2017',10,8,0).
```

Figura 3 - Base de conhecimento de consulta

```
% medico: #IdMedico,Nome,Idade,Género,#IdServ -> {V,F}
medico(1,'Josefa',47,'F',1).
medico(2,'Manuel',29,'M',6).
medico(3,'Gaspar',38,'M',4).
medico(4,'Catia',43,'F',8).
medico(5,'Luisa',40,'F',2).
medico(6,'Marta',38,'F',5).
```

Figura 4 - Base de conhecimento de médico

```
% enfermeiro: #IdEnfermeiro,Nome,Idade,Género,#IdServ -> {V,F}
enfermeiro(1,'Soraia',27,'F',9).
enfermeiro(2,'Jacinta',41,'F',10).
enfermeiro(3,'Luisa',50,'F',11).
enfermeiro(4,'Fabio',30,'M',9).
enfermeiro(5,'Sandra',50,'F',11).
```

Figura 5 - Base de conhecimento de enfermeiro

```
% medEnfFamilia: #IdFamilia,#IdMedico,#IdEnfermeiro -> {V,F}
medEnfFamilia(1,2,3).
medEnfFamilia(2,1,2).
medEnfFamilia(3,3,4).
medEnfFamilia(4,4,1).
medEnfFamilia(5,6,5).
```

Figura 6 - Base de conhecimento de médico e enfermeiro da família

```
% exame: #IdConsulta,Descrição -> {V,F}
exame(3,'Prova de Esforco').
exame(5,'Biopsia').
exame(8,'Endoscopia').
exame(10,'Exame de Sangue').
exame(11,'Exame a urina').
```

Figura 7 - Base de conhecimento de exame

3.2. Invariantes

Antes de construirmos os procedimentos que permitem a adição e a remoção de conhecimento na base de conhecimento do nosso sistema, construímos invariantes estruturais e referenciais para todos os predicados anteriormente mencionados.

3.2.1. Invariantes de utente

Começando pelo utente, aquando da **inserção** de novo conhecimento, é crucial garantir que o id do utente é único (Figura 8), que os utentes que têm ids diferentes não possam ter exatamente a mesma informação nos restantes campos (Figura 9), a idade do utente é válida (maior ou igual a 0) (Figura 10) e o género do utente é válido (maior ou igual a 0) (Figura 11).

```
% Garantir que o id de cada utente é único
+utente(Id,N,I,G,IdF,C) :: (solucoes(Id, utente(Id,_,_,_,_,_), R),
                            comprimento(R, 1)).
```

Figura 8 - Invariante de utente

```
% Garantir que utentes com ids diferentes têm diferente informação
+utente(Id,N,I,G,IdF,C) :: (solucoes((N,I,G,IdF,C), utente(_,N,I,G,IdF,C), R),
                            comprimento(R, 1)).
```

Figura 9 - Invariante de utente

```
% Garantir que a idade do utente é válida (>= 0)
+utente(_,_,I,_,_,_) :: idadeValida(I).
```

Figura 10 - Invariante de utente

```
% Garantir que o género do utente é 'M' ou 'F'
+utente(_,_,_,G,_,_) :: generoValido(G).
```

Figura 11 - Invariante de utente

No que diz respeito à **remoção** de conhecimento de utente, sentimos a necessidade de construir dois invariantes. O primeiro garante que não é possível remover utentes que estejam associados a consultas (Figura 12). O segundo, garante que caso só exista uma família com um único utente e que a essa família esteja associado um médico e enfermeiro de família, o utente não possa ser removido. Caso não exista médico e enfermeiro de família associado ao utente não haverá qualquer problema em removê-lo (Figura 13).

```
% Garantir que não é possível remover um utente com consultas
-utente(Id,_,_,_,_,_) :: (solucoes(Id, consulta(_,_,Id,_,_), R),
                          comprimento(R, 0)).
```

Figura 12 - Invariante de utente

```
% Garantir que não posso eliminar um utente se só restar ele na sua família
% e houver associado à sua família um medico e enfermeiro de familia
-utente(_,_,_,_,IdF,_) :: ((solucoes(IdF, utente(_,_,_,_,IdF,_), R),
                           comprimento(R, N), N >= 1,
                           solucoes(IdF, medEnfFamilia(IdF,_,_), S),
                           comprimento(S, 1));
                          (solucoes(IdF, utente(_,_,_,_,IdF,_), R),
                           comprimento(R, N), N >= 0,
                           solucoes(IdF, medEnfFamilia(IdF,_,_), S),
                           comprimento(S, 0))).
```

Figura 13 - Invariante de utente

3.2.2. Invariantes de serviço

No caso do serviço, no que diz respeito à **adição** de conhecimento, é essencial também garantir que o id de cada serviço é único (Figura 14) e que todos os serviços com ids diferentes não possam ter exatamente a mesma informação nos restantes campos (Figura 15).

```
% Garantir que o id de cada serviço é único
+servico(Id,D,I,C) :: (solucoes(Id, servico(Id,D,I,C), R),
                       comprimento(R, 1)).
```

Figura 14 - Invariante de serviço

```
% Garantir que serviços com ids diferentes têm diferente informação
+servico(Id,D,I,C) :: (solucoes((D,I,C), servico(_,D,I,C), R),
                       comprimento(R, 1)).
```

Figura 15 - Invariante de serviço

Para **remoção** de conhecimento de serviço, é necessário assegurar que não é possível remover um serviço associado a consultas (Figura 16), a médicos (Figura 17) e a enfermeiros (Figura 18).

```
% Garantir que não é possível remover um serviço associado a consultas
-servico(Id,_,_,_) :: (solucoes(Id, consulta(_,_,_,Id,_), R),
                       comprimento(R, 0)).
```

Figura 16 - Invariante de serviço

```
% Garantir que não é possível remover um serviço associado a um médico
-servico(Id,_,_,_) :: (solucoes(Id, medico(_,_,_,_,IdS), R),
                       comprimento(R, 0)).
```

Figura 17 - Invariante de serviço

```
% Garantir que não é possível remover um serviço associado a um enfermeiro
-servico(Id,_,_,_) :: (solucoes(Id, enfermeiro(_,_,_,_,IdS), R),
                       comprimento(R, 0)).
```

Figura 18 - Invariante de serviço

3.2.3. Invariantes de consulta

Na consulta, para a **inserção** de conhecimento recorremos a quatro invariantes. O primeiro garante que o id da consulta é único (Figura 19), o segundo garante que o id do utente associado à consulta existe na base de conhecimento (Figura 20), o terceiro garante que o id do serviço associado à consulta também existe no nosso sistema (Figura 21) e, por fim, o quarto invariante garante que o custo da consulta à válido (maior ou igual a 0) (Figura 22).

```
% Garantir que o id de cada consulta é único
+consulta(Id,D,IdU,IdS,C) :: (solucoes(Id, consulta(Id,D,IdU,IdS,C), R),
                              comprimento(R, 1)).
```

Figura 19 - Invariante de consulta

```
% Garantir que o id do utente associado à consulta existe
+consulta(_,_,IdU,_,_) :: (solucoes(IdU, utente(IdU,_,_,_,_,_), R),
                           comprimento(R, 1)).
```

Figura 20 - Invariante de consulta

```
% Garantir que o id do serviço associado à consulta existe
+consulta(_,_,_,IdS,_) :: (solucoes(IdS, servico(IdS,_,_,_), R),
                           comprimento(R, 1)).
```

Figura 21 - Invariante de consulta

```
% Garantir que o custo de cada consulta é válido (>= 0)
+consulta(_,_,_,_,C) :: custoValido(C).
```

Figura 22 - Invariante de consulta

Para a **remoção** de conhecimento relativo a consultas, é somente necessário assegurar que não se pode remover uma consulta que esteja associada a um exame (Figura 23).

```
% Garantir que não é possível remover uma consulta associado a um exame
-consulta(Id,_,_,_,_) :: (solucoes(Id, exame(Id,_), R),
                          comprimento(R, 0)).
```

Figura 23 - Invariante de consulta

3.2.4. Invariantes de médico

Quanto ao médico, no que diz respeito à **inserção** de novos predicados, foi necessário inserir vários invariantes, uma vez que é necessário garantir que o id de cada médico é único (Figura 24), que médicos com ids diferentes têm diferente informação (Figura 25), a idade é válida (Figura 26) e o género é válido (Figura 27).

```
% Garantir que o id de cada médico é único
+medico(Id,N,I,G,IdS) :: (solucoes(Id, medico(Id,_,_,_,_), R),
                          comprimento(R, 1)).
```

Figura 24 - Invariante de médico

```
% Garantir que médicos com ids diferentes têm diferente informação
+medico(Id,N,I,G,IdS) :: (solucoes((N,I,G,IdS), medico(_,N,I,G,IdS), R),
                          comprimento(R, 1)).
```

Figura 25 - Invariante de médico

```
% Garantir que a idade do médico é válida (>= 0)
+medico(_,_,I,_,_) :: idadeValida(I).
```

Figura 26 - Invariante de médico

```
% Garantir que o género do médico é 'M' ou 'F'
+medico(_,_,_,G,_) :: generoValido(G).
```

Figura 27 - Invariante de médico

Quanto à **remoção**, tem que se garantir que não existe nenhum médico associado a uma família (Figura 28).

```
% Garantir que não é possível remover um médico de uma familia
-medico(Id,_,_,_,_) :: (solucoes(Id, medEnfFamilia(_,Id,_), R),
                        comprimento(R, 0)).
```

Figura 28 - Invariante de médico

3.2.5. Invariantes de enfermeiro

O caso do enfermeiro é análogo ao caso do médico, ou seja, para **adicionar** conhecimento o id de cada enfermeiro tem que ser único (Figura 29), enfermeiros com ids diferentes têm diferente informação (Figura 30), a idade é válida (Figura 31) e o género é válido (Figura 32).

```
% Garantir que o id de cada enfermeiro é único
+enfermeiro(Id,N,I,G,IdS) :: (solucoes(Id, enfermeiro(Id,_,_,_,_), R),
                              comprimento(R, 1)).
```

Figura 29 - Invariante de enfermeiro

```
% Garantir que enfermeiro com ids diferentes têm diferente informação
+enfermeiro(Id,N,I,G,IdS) :: (solucoes((N,I,G,IdS), enfermeiro(_,N,I,G,IdS), R),
                              comprimento(R, 1)).
```

Figura 30 - Invariante de enfermeiro

```
% Garantir que a idade do enfermeiro é válida (>= 0)
+enfermeiro(_,_,I,_,_) :: idadeValida(I).
```

Figura 31 - Invariante de enfermeiro

```
% Garantir que o género do enfermeiro é 'M' ou 'F'
+enfermeiro(_,_,_,G,_) :: generoValido(G).
```

Figura 32 - Invariante de enfermeiro

Para **remover**, tem que se garantir que não existe nenhum enfermeiro associado a uma família (Figura 33).

```
% Garantir que não é possível remover um enfermeiro de uma familia
-enfermeiro(Id,_,_,_,_) :: (solucoes(Id, medEnfFamilia(_,_,Id), R),
                            comprimento(R, 0)).
```

Figura 33 - Invariante de enfermeiro

3.2.6. Invariantes de médico e enfermeiro da família

No médico e enfermeiro da família (medEnfFamilia), construímos invariantes que garantem que a família à qual se pretende associar um médico e um enfermeiro ainda não possui nenhum médico nem enfermeiro associado (Figura 34 e Figura 35, respetivamente), garantir que existe pelo menos um utente com o id da família a inserir (Figura 36) e garantir que o id do médico e do enfermeiro que se pretendem associar também existem na base de conhecimento (Figura 37 e Figura 38, respetivamente).

```
% Garantir que a familia ainda não tem nenhum médico associado
+medEnfFamilia(IdFam,IdMed,_) ::
        (solucoes((IdFam,IdMed), medEnfFamilia(IdFam,IdMed,_), R),
         comprimento(R, 1)).
```

Figura 34 - Invariante de médico e enfermeiro da família

```
% Garantir que a familia ainda não tem nenhum enfermeiro associado
+medEnfFamilia(IdFam,_,IdEnf) ::
        (solucoes((IdFam,IdEnf), medEnfFamilia(IdFam,_,IdEnf), R),
         comprimento(R, 1)).
```

Figura 35 - Invariante de médico e enfermeiro da família

```
% Garantir que o id da familia associada existe
+medEnfFamilia(IdFam,_,_) :: (solucoes(IdFam, utente(_,_,_,_,IdFam,_), R),
                              comprimento(R, 1)).
```

Figura 36 - Invariante de médico e enfermeiro da família

```
% Garantir que o id do medico associado existe
+medEnfFamilia(_,IdMed,_) :: (solucoes(IdMed, medico(IdMed,_,_,_,_), R),
                              comprimento(R, 1)).
```

Figura 37 - Invariante de médico e enfermeiro da família

```
% Garantir que o id do enfermeiro associado existe
+medEnfFamilia(_,_,IdEnf) :: (solucoes(IdEnf, enfermeiro(IdEnf,_,_,_,_), R),
                              comprimento(R, 1)).
```

Figura 38 - Invariante de médico e enfermeiro da família

3.2.7. Invariantes de exame

Por fim, para o exame, foi necessário construir um invariante que garante que o exame se refere a uma consulta que existe na base de conhecimento (Figura 39).

```
% Garantir que o id da consulta associada existe
+exame(IdConsulta,_) :: (solucoes(IdConsulta, consulta(IdConsulta,_,_,_,_), R),
                         comprimento(R, 1)).
```

Figura 39 - Invariante de exame

3.3. Adicionar conhecimento

Para adicionar ao sistema a funcionalidade de registar novo conhecimento, foi crucial o meta-predicado *evolução* (Figura 107) e os invariantes anteriormente apresentados.

3.3.1. Registar utentes, serviços, consultas, médicos, enfermeiros, médicos e enfermeiros da família e exames

Para **registar um utente**, basta que seja invocado o predicado ***novoUtente*** com os seis parâmetros necessários.

```
novoUtente(Id,N,I,G,IdF,C) :- evolucao(utente(Id,N,I,G,IdF,C)).
```

Figura 40 - Predicado 'novoUtente'

Tal como foi mencionado na Secção 3.2.1, existem invariantes que têm que continuar válidos após a inserção de conhecimento relativo a um novo utente. Em seguida, exemplificamos dois deles, o primeiro (lado esquerdo) em que a idade é inválida e o segundo (lado direito) onde o género inserido é inválido.

```
| ?- listing.
utente(1, 'Carlos', 2, 'M', 1, 'Porto').
utente(2, 'Joao', 20, 'M', 1, 'Braga').
utente(3, 'Vilaca', 20, 'M', 2, 'Braga').
utente(4, 'Catarina', 20, 'F', 3, 'Braga').
utente(5, 'Eva', 0, 'F', 3, 'Braga').
utente(6, 'Maria Jose', 56, 'F', 1, 'Guimaraes').
utente(7, 'Gustavo', 28, 'M', 2, 'Lisboa').
utente(8, 'Helena', 25, 'F', 5, 'Lisboa').
utente(9, 'Guilherme', 58, 'M', 1, 'Porto').
utente(10, 'Ana', 34, 'F', 6, 'Guimaraes').

yes
| ?- novoUtente(11,'Joana',-1,'F',7,'Braga').
no
| ?- listing.
utente(1, 'Carlos', 2, 'M', 1, 'Porto').
utente(2, 'Joao', 20, 'M', 1, 'Braga').
utente(3, 'Vilaca', 20, 'M', 2, 'Braga').
utente(4, 'Catarina', 20, 'F', 3, 'Braga').
utente(5, 'Eva', 0, 'F', 3, 'Braga').
utente(6, 'Maria Jose', 56, 'F', 1, 'Guimaraes').
utente(7, 'Gustavo', 28, 'M', 2, 'Lisboa').
utente(8, 'Helena', 25, 'F', 5, 'Lisboa').
utente(9, 'Guilherme', 58, 'M', 1, 'Porto').
utente(10, 'Ana', 34, 'F', 6, 'Guimaraes').

yes
```

```
| ?- listing.
utente(1, 'Carlos', 2, 'M', 1, 'Porto').
utente(2, 'Joao', 20, 'M', 1, 'Braga').
utente(3, 'Vilaca', 20, 'M', 2, 'Braga').
utente(4, 'Catarina', 20, 'F', 3, 'Braga').
utente(5, 'Eva', 0, 'F', 3, 'Braga').
utente(6, 'Maria Jose', 56, 'F', 1, 'Guimaraes').
utente(7, 'Gustavo', 28, 'M', 2, 'Lisboa').
utente(8, 'Helena', 25, 'F', 5, 'Lisboa').
utente(9, 'Guilherme', 58, 'M', 1, 'Porto').
utente(10, 'Ana', 34, 'F', 6, 'Guimaraes').

yes
| ?- novoUtente(11,'Joana',35,'L',6,'Guimaraes').
no
| ?- listing.
utente(1, 'Carlos', 2, 'M', 1, 'Porto').
utente(2, 'Joao', 20, 'M', 1, 'Braga').
utente(3, 'Vilaca', 20, 'M', 2, 'Braga').
utente(4, 'Catarina', 20, 'F', 3, 'Braga').
utente(5, 'Eva', 0, 'F', 3, 'Braga').
utente(6, 'Maria Jose', 56, 'F', 1, 'Guimaraes').
utente(7, 'Gustavo', 28, 'M', 2, 'Lisboa').
utente(8, 'Helena', 25, 'F', 5, 'Lisboa').
utente(9, 'Guilherme', 58, 'M', 1, 'Porto').
utente(10, 'Ana', 34, 'F', 6, 'Guimaraes').

yes
```

Figura 41 - Exemplificações de +invariantes de utente

Na figura seguinte encontra-se ilustrado um exemplo prático de sucesso na inserção de um novo utente.

```
| ?- listing.
utente(1, 'Carlos', 2, 'M', 1, 'Porto').
utente(2, 'Joao', 20, 'M', 1, 'Braga').

yes
| ?- novoUtente(3, 'Joaquina',23,'F',2,'Lisboa').
yes
| ?- listing.
utente(1, 'Carlos', 2, 'M', 1, 'Porto').
utente(2, 'Joao', 20, 'M', 1, 'Braga').
utente(3, 'Joaquina', 23, 'F', 2, 'Lisboa').

yes
```

Figura 42 - Exemplificação de +invariantes de utente

No **registo de um serviço**, o predicado a utilizar é o apresentado na figura seguinte.

```
novoServico(Id,D,I,C) :- evolucao(servico(Id,D,I,C)).
```

Figura 43 - Predicado 'novoServico'

Exemplificamos também a aplicação prática dos invariantes de serviço (Secção 3.2.2), o primeiro (lado direito) garante que não existe serviços com o mesmo id, e o segundo (lado esquerdo) garante que não existe conhecimento com diferentes id mas restante informação igual.

```
| ?- listing.
servico(1, 'Pediatria', 'Hospital Santa Maria', 'Porto').
servico(2, 'Ortopedia', 'Hospital de Braga', 'Braga').
servico(3, 'Oftalmologia', 'Hospital Santa Maria', 'Porto').
servico(4, 'Cardiologia', 'Hospital de Braga', 'Braga').
servico(5, 'Cirurgia Geral', 'Hospital Egas Moniz', 'Lisboa').
servico(6, 'Estomatologia', 'Hospital Egas Moniz', 'Lisboa').
servico(7, 'Genetica Medica', 'Hospital Pulido Valente', 'Lisboa').
servico(8, 'Medicina Nuclear', 'Hospital Pulido Valente', 'Lisboa').
servico(9, 'Curativo', 'Hospital Egas Moniz', 'Lisboa').
servico(10, 'Vacina', 'Hospital de Braga', 'Braga').
servico(11, 'Preparar para exame', 'Hospital Santa Maria', 'Porto').

yes
| ?- novoServico(11, 'Fisioterapia', 'Hospital de Braga', 'Braga').
no
| ?- listing.
servico(1, 'Pediatria', 'Hospital Santa Maria', 'Porto').
servico(2, 'Ortopedia', 'Hospital de Braga', 'Braga').
servico(3, 'Oftalmologia', 'Hospital Santa Maria', 'Porto').
servico(4, 'Cardiologia', 'Hospital de Braga', 'Braga').
servico(5, 'Cirurgia Geral', 'Hospital Egas Moniz', 'Lisboa').
servico(6, 'Estomatologia', 'Hospital Egas Moniz', 'Lisboa').
servico(7, 'Genetica Medica', 'Hospital Pulido Valente', 'Lisboa').
servico(8, 'Medicina Nuclear', 'Hospital Pulido Valente', 'Lisboa').
servico(9, 'Curativo', 'Hospital Egas Moniz', 'Lisboa').
servico(10, 'Vacina', 'Hospital de Braga', 'Braga').
servico(11, 'Preparar para exame', 'Hospital Santa Maria', 'Porto').

yes
```

```
| ?- listing.
servico(1, 'Pediatria', 'Hospital Santa Maria', 'Porto').
servico(2, 'Ortopedia', 'Hospital de Braga', 'Braga').
servico(3, 'Oftalmologia', 'Hospital Santa Maria', 'Porto').
servico(4, 'Cardiologia', 'Hospital de Braga', 'Braga').
servico(5, 'Cirurgia Geral', 'Hospital Egas Moniz', 'Lisboa').
servico(6, 'Estomatologia', 'Hospital Egas Moniz', 'Lisboa').
servico(7, 'Genetica Medica', 'Hospital Pulido Valente', 'Lisboa').
servico(8, 'Medicina Nuclear', 'Hospital Pulido Valente', 'Lisboa').
servico(9, 'Curativo', 'Hospital Egas Moniz', 'Lisboa').
servico(10, 'Vacina', 'Hospital de Braga', 'Braga').
servico(11, 'Preparar para exame', 'Hospital Santa Maria', 'Porto').

yes
| ?- novoServico(12, 'Vacina', 'Hospital de Braga', 'Braga').
no
| ?- listing.
servico(1, 'Pediatria', 'Hospital Santa Maria', 'Porto').
servico(2, 'Ortopedia', 'Hospital de Braga', 'Braga').
servico(3, 'Oftalmologia', 'Hospital Santa Maria', 'Porto').
servico(4, 'Cardiologia', 'Hospital de Braga', 'Braga').
servico(5, 'Cirurgia Geral', 'Hospital Egas Moniz', 'Lisboa').
servico(6, 'Estomatologia', 'Hospital Egas Moniz', 'Lisboa').
servico(7, 'Genetica Medica', 'Hospital Pulido Valente', 'Lisboa').
servico(8, 'Medicina Nuclear', 'Hospital Pulido Valente', 'Lisboa').
servico(9, 'Curativo', 'Hospital Egas Moniz', 'Lisboa').
servico(10, 'Vacina', 'Hospital de Braga', 'Braga').
servico(11, 'Preparar para exame', 'Hospital Santa Maria', 'Porto').

yes
```

Figura 44 - Exemplificações de +invariantes de serviço

Para o **registo de uma consulta** o raciocínio foi análogo ao já mencionado.

```
novaConsulta(Id,D,IdU,IdS,C) :- evolucao(consulta(Id,D,IdU,IdS,C)).
```

Figura 45 - Predicado 'novaConsulta'

Em seguida, ilustramos também a eficácia dos invariantes de consulta construídos (Secção 3.2.3), onde, na Figura 46, no lado esquerdo apresentamos a garantia do invariante que não permite que se insira uma consulta com um id de um utente inválido (tal como se pode ver na Figura 1, o idUtente = 12 não existe) e do lado direito a garantia de que não existirão consultas com custos inválidos.

```
| ?- listing.
consulta(1, '01-01-2018', 1, 1, 10).
consulta(2, '23-05-2016', 1, 1, 14).
consulta(3, '05-02-2019', 2, 4, 15).
consulta(4, '13-12-2018', 2, 1, 20).
consulta(5, '09-09-2017', 3, 5, 10).
consulta(6, '26-10-2016', 1, 3, 15).
consulta(7, '29-02-2017', 3, 3, 30).
consulta(8, '10-03-2016', 4, 6, 20).
consulta(9, '02-04-2017', 2, 2, 40).
consulta(10, '04-04-2017', 5, 7, 10).
consulta(11, '12-12-2017', 10, 8, 0).

yes
| ?- novaConsulta(12,'23-10-2016',12,8,0).
no
| ?- listing.
consulta(1, '01-01-2018', 1, 1, 10).
consulta(2, '23-05-2016', 1, 1, 14).
consulta(3, '05-02-2019', 2, 4, 15).
consulta(4, '13-12-2018', 2, 1, 20).
consulta(5, '09-09-2017', 3, 5, 10).
consulta(6, '26-10-2016', 1, 3, 15).
consulta(7, '29-02-2017', 3, 3, 30).
consulta(8, '10-03-2016', 4, 6, 20).
consulta(9, '02-04-2017', 2, 2, 40).
consulta(10, '04-04-2017', 5, 7, 10).
consulta(11, '12-12-2017', 10, 8, 0).

yes
```

```
| ?- listing.
consulta(1, '01-01-2018', 1, 1, 10).
consulta(2, '23-05-2016', 1, 1, 14).
consulta(3, '05-02-2019', 2, 4, 15).
consulta(4, '13-12-2018', 2, 1, 20).
consulta(5, '09-09-2017', 3, 5, 10).
consulta(6, '26-10-2016', 1, 3, 15).
consulta(7, '29-02-2017', 3, 3, 30).
consulta(8, '10-03-2016', 4, 6, 20).
consulta(9, '02-04-2017', 2, 2, 40).
consulta(10, '04-04-2017', 5, 7, 10).
consulta(11, '12-12-2017', 10, 8, 0).

yes
| ?- novaConsulta(12,'23-10-2016',1,1,-1).
no
| ?- listing.
consulta(1, '01-01-2018', 1, 1, 10).
consulta(2, '23-05-2016', 1, 1, 14).
consulta(3, '05-02-2019', 2, 4, 15).
consulta(4, '13-12-2018', 2, 1, 20).
consulta(5, '09-09-2017', 3, 5, 10).
consulta(6, '26-10-2016', 1, 3, 15).
consulta(7, '29-02-2017', 3, 3, 30).
consulta(8, '10-03-2016', 4, 6, 20).
consulta(9, '02-04-2017', 2, 2, 40).
consulta(10, '04-04-2017', 5, 7, 10).
consulta(11, '12-12-2017', 10, 8, 0).

yes
```

Figura 46 - Exemplificações de +invariantes de consulta

Para que os novos predicados inseridos também tivessem funcionalidade de inserção, construímos também predicados de registos para o **médico** (Figura 47), para o **enfermeiro** (Figura 48), para o **médico e enfermeiro da família** (Figura 49) e para o **exame** (Figura 50).

```
novoMedico(Id,N,I,G,IdS) :- evolucao(medico(Id,N,I,G,IdS)).
```

Figura 47 - Predicado 'novoMedico'

```
novoEnfermeiro(Id,N,I,G,IdS) :- evolucao(enfermeiro(Id,N,I,G,IdS)).
```

Figura 48 - Predicado 'novoEnfermeiro'

```
novoMedEnfFamilia(IdFam,IdMed,IdEnf) :- evolucao(medEnfFamilia(IdFam,IdMed,IdEnf)).
```

Figura 49 - Predicado 'novoMedEnfFamilia'

```
novoExame(IdConsulta,D) :- evolucao(exame(IdConsulta,D)).
```

Figura 50 - Predicado 'novoExame'

Também os invariantes construídos para cada um desses predicados são fundamentais para garantir a consistência da nossa base de conhecimento (Secções 3.2.4, 3.2.5, 3.2.6 e 3.2.7).

3.4. Remover conhecimento

Para remover conhecimento do sistema recorremos ao meta-predicado construído para esse efeito, *involução* (Figura 108), e aos invariantes já apresentados.

3.4.1. Remover utentes, serviços, consultas, médicos, enfermeiros, médicos e enfermeiros da família e exames

De forma a tornar os procedimentos de remoção mais práticos e tendo em consideração que o **utente**, o **serviço**, a **consulta**, o **médico** e o **enfermeiro** são predicados inequivocamente identificados por um id (dentro da sua determinada categoria), para remover conhecimento de cada um desses casos basta que seja invocada o procedimento respetivo com o id desejado. Por outro lado, no caso dos **médicos e enfermeiros da família** e do **exame** é necessário que o procedimento de remoção receba todos os parâmetros visto que esses predicados não possuem um id único, e somente conseguem ser inequivocamente identificados através de todos os seus campos.

No caso da **remoção de conhecimento de um utente**, é então necessário que o predicado ***removeUtente*** receba como parâmetro o id do utente que se pretende eliminar. O predicado *utenteID* é responsável por colocar em X o primeiro elemento da lista de utentes com o id passado como parâmetro (a lista só possui um elemento uma vez que os ids são únicos) e, em seguida, a *involução* remove o determinado utente X da base de conhecimento.

```
removeUtente(Id) :- utenteID(Id,[X|_]), involucao(X).
```

Figura 51 - Predicado 'removeUtente'

Tal como anteriormente mencionado, existem algumas premissas que têm que permanecer verdadeiras para que a remoção do conhecimento de utente efetivamente se concretize (Secção 12). Na figura seguinte (Figura 52), exemplificamos os dois invariantes de remoção construídos, onde o primeiro, à esquerda, garante que não é possível remover um utente que esteja associado a alguma consulta, e o segundo, à direita, que assegura que se o utente em questão for o único elemento de uma família com um médico e enfermeiro atribuído, o utente não poderá ser removido.

Na Figura 53 apresentamos um caso de **sucesso**, onde o utente com id igual a 6 é removido da base de conhecimento.

```
| ?- listing.
consulta(1, '01-01-2018', 1, 1, 10).
consulta(2, '23-05-2016', 1, 1, 14).

utente(1, 'Carlos', 2, 'M', 1, 'Porto').
utente(2, 'Joao', 20, 'M', 1, 'Braga').

yes
| ?- removeUtente(1).
no
| ?- listing.
consulta(1, '01-01-2018', 1, 1, 10).
consulta(2, '23-05-2016', 1, 1, 14).

utente(2, 'Joao', 20, 'M', 1, 'Braga').
utente(1, 'Carlos', 2, 'M', 1, 'Porto').

yes
```

```
| ?- listing.
medEnfFamilia(4, 4, 1).
medEnfFamilia(5, 6, 5).

utente(1, 'Carlos', 2, 'M', 1, 'Porto').
utente(8, 'Helena', 25, 'F', 5, 'Lisboa').

yes
| ?- removeUtente(8).
no
| ?- listing.
medEnfFamilia(4, 4, 1).
medEnfFamilia(5, 6, 5).

utente(1, 'Carlos', 2, 'M', 1, 'Porto').
utente(8, 'Helena', 25, 'F', 5, 'Lisboa').

yes
```

Figura 52 - Exemplificações de -invariantes de utente

```
| ?- listing.
consulta(5, '09-09-2017', 3, 5, 10).
consulta(6, '26-10-2016', 1, 3, 15).

medEnfFamilia(3, 3, 4).

utente(5, 'Eva', 0, 'F', 3, 'Braga').
utente(6, 'Maria Jose', 56, 'F', 1, 'Guimaraes').
utente(7, 'Gustavo', 28, 'M', 2, 'Lisboa').

yes
| ?- removeUtente(6).
yes
| ?- listing.
consulta(5, '09-09-2017', 3, 5, 10).
consulta(6, '26-10-2016', 1, 3, 15).

medEnfFamilia(3, 3, 4).

utente(5, 'Eva', 0, 'F', 3, 'Braga').
utente(7, 'Gustavo', 28, 'M', 2, 'Lisboa').

yes
```

Figura 53 - Exemplificação de -invariantes de utente

Na figura seguinte apresentamos o predicado responsável por **remover um serviço** da base de conhecimento.

```
removeServico(Id) :- servicoID(Id,[X|_]), involucao(X).
```

Figura 54 - Predicado 'removeServico'

Em seguida, encontra-se exemplificado um invariante do predicado serviço (Secção 3.2.2), que, neste caso, é o invariante que garante que não seja possível remover um serviço que é realizado por um médico.

```
| ?- listing.
medico(1, 'Josefa', 47, 'F', 1).
medico(2, 'Manuel', 29, 'M', 6).

servico(1, 'Pediatria', 'Hospital Santa Maria', 'Porto').
servico(2, 'Ortopedia', 'Hospital de Braga', 'Braga').

yes
| ?- removeServico(1).
no
| ?- listing.
medico(1, 'Josefa', 47, 'F', 1).
medico(2, 'Manuel', 29, 'M', 6).

servico(2, 'Ortopedia', 'Hospital de Braga', 'Braga').
servico(1, 'Pediatria', 'Hospital Santa Maria', 'Porto').

yes
```

Figura 55 - Exemplificação de -invariantes de serviço

Para além deste invariante, o serviço tem também invariantes que garantem a consistência dos serviços associados a consultas e a enfermeiros.

Para a **remoção de uma consulta** utilizamos o predicado apresentado na figura seguinte.

```
removeConsulta(Id) :- consultaID(Id,[X|_]), involucao(X).
```

Figura 56 - Predicado 'removeConsulta'

A consulta possui um invariante (Secção 3.2.3) que garante que não é possível remover uma consulta que tem um exame associado. Na figura seguinte encontra-se um exemplo prático de aplicação desse invariante.

```
| ?- listing.
consulta(1, '01-01-2018', 1, 1, 10).
consulta(2, '23-05-2016', 1, 1, 14).
consulta(3, '05-02-2019', 2, 4, 15).

exame(3, 'Prova de Esforco').
exame(5, 'Biopsia').

yes
| ?- removeConsulta(3).
no
| ?- listing.
consulta(1, '01-01-2018', 1, 1, 10).
consulta(2, '23-05-2016', 1, 1, 14).
consulta(3, '05-02-2019', 2, 4, 15).

exame(3, 'Prova de Esforco').
exame(5, 'Biopsia').

yes
```

Figura 57 - Exemplificação de -invariantes de consulta

A par da inserção, também na remoção de conhecimento construímos procedimentos para os nossos predicados extras.

O predicado que permite a **remoção de médicos** encontra-se ilustrado na Figura 58. Esta remoção possui um único invariante que garante que não se pode remover um médico que está associado a um médico e enfermeiro da família (Secção 3.2.4), tal como se pode ver no exemplo meramente ilustrativo da Figura 59.

```
removeMedico(Id) :- medicoID(Id,[X|_]), involucao(X).
```

Figura 58 - Predicado 'removeMedico'

```
| ?- listing.
medEnfFamilia(3, 3, 4).
medEnfFamilia(4, 4, 1).

medico(3, 'Gaspar', 38, 'M', 4).
medico(4, 'Catia', 43, 'F', 8).

yes
| ?- removeMedico(3).
no
| ?- listing.
medEnfFamilia(3, 3, 4).
medEnfFamilia(4, 4, 1).

medico(4, 'Catia', 43, 'F', 8).
medico(3, 'Gaspar', 38, 'M', 4).

yes
```

Figura 59 - Exemplificação de -invariantes de medico

Para o **enfermeiro**, com o predicado de remoção apresentado na Figura 60, o invariante de remoção é semelhante ao do médico, ou seja, garante que não se pode remover um enfermeiro que está associado a um médico e enfermeiro da família (Secção 3.2.5).

```
removeEnfermeiro(Id) :- enfermeiroID(Id,[X|_]), involucao(X).
```

Figura 60 - Predicado 'removeEnfermeiro'

O **médico e enfermeiro da família** e o **exame** não possuem invariantes estruturais nem referenciais de remoção, logo, haverá sempre sucesso quando se pretende remover conhecimento destas duas extensões de predicados.

```
removeMedEnfFamilia(IdFam,IdMed,IdEnf) :- involucao(medEnfFamilia(IdFam,IdMed,IdEnf)).
```

Figura 61 - Predicado 'removeMedEnfFamilia'

```
removeExame(IdConsulta,D) :- involucao(exame(IdConsulta,D)).
```

Figura 62 - Predicado 'removeExame'

3.5. Identificações

Foram construídos vários procedimentos que visam pesquisar dados no nosso sistema de representação de conhecimento e raciocínio, de modo a tornar o nosso sistema muito mais operacional.

3.5.1. Identificar as instituições prestadoras de serviços

Para identificarmos quais são as instituições que prestam serviços temos que pesquisar no predicado **serviço** as instituições associadas a cada um deles. Através do meta-predicado *soluções* (Figura 102) colocamos numa lista temporária (Tmp) todas as instituições onde se realizam serviços. No entanto, uma vez que a mesma instituição pode estar associada a vários serviços essa lista pode conter repetidos. Assim, através do procedimento *eliminaRepetidos* (Figura 110), é colocado em R o conjunto de instituições, sem repetições.

Na figura seguinte encontra-se o procedimento descrito.

```
identificaInstituicoes(R) :- solucoes(I, servico(_,_,I,_), Tmp),
                             eliminaRepetidos(Tmp, R).
```

Figura 63 - Predicado 'identificaInstituicoes'

Testando o predicado ***identificaInstituicoes***, obtemos o resultado pretendido, tal como se pode ver no exemplo da figura seguinte.

```
| ?- listing.
servico(1, 'Pediatria', 'Hospital Santa Maria', 'Porto').
servico(2, 'Ortopedia', 'Hospital de Braga', 'Braga').
servico(3, 'Oftalmologia', 'Hospital Santa Maria', 'Porto').
servico(4, 'Cardiologia', 'Hospital de Braga', 'Braga').
servico(5, 'Cirurgia Geral', 'Hospital Egas Moniz', 'Lisboa').
servico(6, 'Estomatologia', 'Hospital Egas Moniz', 'Lisboa').
servico(7, 'Genetica Medica', 'Hospital Pulido Valente', 'Lisboa').
servico(8, 'Medicina Nuclear', 'Hospital Pulido Valente', 'Lisboa').
servico(9, 'Curativo', 'Hospital Egas Moniz', 'Lisboa').
servico(10, 'Vacina', 'Hospital de Braga', 'Braga').
servico(11, 'Preparar para exame', 'Hospital Santa Maria', 'Porto').

yes
| ?- identificaInstituicoes(R).
R = ['Hospital Pulido Valente','Hospital Egas Moniz','Hospital de Braga','Hospital Santa Maria'] ? yes
yes
```

Figura 64 - Exemplo 'identificaInstituicoes'

3.5.2. Identificar utentes/serviços/consultas/médicos/enfermeiros/médico e enfermeiro da família/exames por critérios de seleção

Para cada uma das extensões de predicados do nosso sistema, construímos procedimentos que os identificam através dos seus critérios de seleção, ou seja, dos diferentes campos que cada um desse tipo de conhecimento contém.

Para o **utente**, uma vez que possui 6 diferentes campos, construímos 6 diferentes predicados, que são os seguintes: procurar um utente através do id passado como parâmetro (***utenteID***), utentes com um determinado nome (***utenteNome***), utentes de uma determinada idade (***utenteIdade***), utentes de uma determinada cidade (***utenteCidade***), utentes de um género (***utenteGenero***) e utentes de uma determinada família (***utenteFamilia***).

Estes predicados usam o meta-predicado *soluções* (Figura 102) para colocar em R o *match* pretendido.

```
utenteID(Id, R) :- solucoes(utente(Id,N,I,G,IdF,C),
                            utente(Id,N,I,G,IdF,C), R).

utenteNome(N, R) :- solucoes(utente(Id,N,I,G,IdF,C),
                             utente(Id,N,I,G,IdF,C), R).

utenteIdade(I, R) :- solucoes(utente(Id,N,I,G,IdF,C),
                              utente(Id,N,I,G,IdF,C), R).

utenteCidade(C, R) :- solucoes(utente(Id,N,I,G,IdF,C),
                               utente(Id,N,I,G,IdF,C), R).

utenteGenero(G, R) :- solucoes(utente(Id,N,I,G,IdF,C),
                               utente(Id,N,I,G,IdF,C), R).

utenteFamilia(IdF, R) :- solucoes(utente(Id,N,I,G,IdF,C),
                                  utente(Id,N,I,G,IdF,C), R).
```

Figura 65 - Predicados utente por critérios de seleção

Na figura seguinte encontra-se um exemplo prático de aplicação do predicado ***utenteCidade***, onde procuramos todos os utentes que residem em Guimarães.

```
| ?- listing.
utente(1, 'Carlos', 2, 'M', 1, 'Porto').
utente(2, 'Joao', 20, 'M', 1, 'Braga').
utente(3, 'Vilaca', 20, 'M', 2, 'Braga').
utente(4, 'Catarina', 20, 'F', 3, 'Braga').
utente(5, 'Eva', 0, 'F', 3, 'Braga').
utente(6, 'Maria Jose', 56, 'F', 1, 'Guimaraes').
utente(7, 'Gustavo', 28, 'M', 2, 'Lisboa').
utente(8, 'Helena', 25, 'F', 5, 'Lisboa').
utente(9, 'Guilherme', 58, 'M', 1, 'Porto').
utente(10, 'Ana', 34, 'F', 6, 'Guimaraes').

yes
| ?- utenteCidade('Guimaraes',R).
R = [utente(10,'Ana',34,'F',6,'Guimaraes'),utente(6,'Maria Jose',56,'F',1,'Guimaraes')] ? yes
yes
```

Figura 66 - Exemplo 'utenteCidade'

O **serviço** possui 4 diferentes campos que, consequentemente, originam 4 diferentes predicados de seleção: o id (***servicoID***), a descrição (***servicoDescricao***), a instituição (***servicoInstituicao***) e a cidade (***servicoCidade***).

```
servicoID(Id,R) :- solucoes(servico(Id,D,I,C),
                            servico(Id,D,I,C), R).

servicoDescricao(D, R) :- solucoes(servico(Id,D,I,C),
                                   servico(Id,D,I,C), R).

servicoInstituicao(I, R) :- solucoes(servico(Id,D,I,C),
                                     servico(Id,D,I,C), R).

servicoCidade(C, R) :- solucoes(servico(Id,D,I,C),
                                servico(Id,D,I,C), R).
```

Figura 67 - Predicados serviço por critérios de seleção

A **consulta** pode ser identificada através de qualquer um dos seus 5 critérios de seleção, tal como se pode ver na figura seguinte.

```
consultaID(Id,R) :- solucoes(consulta(Id,D,IdU,IdS,C),
                             consulta(Id,D,IdU,IdS,C), R).

consultaData(D,R) :- solucoes(consulta(Id,D,IdU,IdS,C),
                              consulta(Id,D,IdU,IdS,C), R).

consultaIdUtente(IdU, R) :- solucoes(consulta(Id,D,IdU,IdS,C),
                                     consulta(Id,D,IdU,IdS,C), R).

consultaIdServico(IdS, R) :- solucoes(consulta(Id,D,IdU,IdS,C),
                                      consulta(Id,D,IdU,IdS,C), R).

consultaCusto(C, R) :- solucoes(consulta(Id,D,IdU,IdS,C),
                                consulta(Id,D,IdU,IdS,C), R).
```

Figura 68 - Predicados consulta por critérios de seleção

Para as nossas extensões de predicados extra, construímos também procedimentos que identificam cada um deles através dos critérios de seleção existentes.

Na Figura 69 encontram-se os predicados de **médico** pelos vários critérios de seleção e na Figura 70 encontram-se os de **enfermeiro**.

```
medicoID(Id, R) :- solucoes(medico(Id,N,I,G,IdS),
                            medico(Id,N,I,G,IdS), R).

medicoNome(N, R) :- solucoes(medico(Id,N,I,G,IdS),
                             medico(Id,N,I,G,IdS), R).

medicoIdade(I, R) :- solucoes(medico(Id,N,I,G,IdS),
                              medico(Id,N,I,G,IdS), R).

medicoGenero(G, R) :- solucoes(medico(Id,N,I,G,IdS),
                               medico(Id,N,I,G,IdS), R).

medicoIdServico(IdS, R) :- solucoes(medico(Id,N,I,G,IdS),
                                    medico(Id,N,I,G,IdS), R).
```

Figura 69 - Predicados médico por critérios de seleção

```
enfermeiroID(Id, R) :- solucoes(enfermeiro(Id,N,I,G,IdS),
                                enfermeiro(Id,N,I,G,IdS), R).

enfermeiroNome(N, R) :- solucoes(enfermeiro(Id,N,I,G,IdS),
                                 enfermeiro(Id,N,I,G,IdS), R).

enfermeiroIdade(I, R) :- solucoes(enfermeiro(Id,N,I,G,IdS),
                                  enfermeiro(Id,N,I,G,IdS), R).

enfermeiroGenero(G, R) :- solucoes(enfermeiro(Id,N,I,G,IdS),
                                   enfermeiro(Id,N,I,G,IdS), R).

enfermeiroIdServico(IdS, R) :- solucoes(enfermeiro(Id,N,I,G,IdS),
                                        enfermeiro(Id,N,I,G,IdS), R).
```

Figura 70 - Predicados enfermeiro por critérios de seleção

Um exemplo prático do predicado ***medicoGenero*** encontra-se na seguinte figura, onde procuramos todos os médicos do sexo feminino.

```
| ?- listing.
medico(1, 'Josefa', 47, 'F', 1).
medico(2, 'Manuel', 29, 'M', 6).
medico(3, 'Gaspar', 38, 'M', 4).
medico(4, 'Catia', 43, 'F', 8).
medico(5, 'Luisa', 40, 'F', 2).
medico(6, 'Marta', 38, 'F', 5).

yes
| ?- medicoGenero('F',R).
R = [medico(6,'Marta',38,'F',5),medico(5,'Luisa',40,'F',2),medico(4,'Catia',43,'F',8),
medico(1,'Josefa',47,'F',1)] ? yes
yes
```

Figura 71 - Exemplo 'medicoGenero'

Para o **médico e enfermeiro da família** e para o **exame** o processo é semelhante, Figura 72 e Figura 73, respetivamente.

```
medEnfFamiliaIdFam(IdFam, R) :- solucoes(medEnfFamilia(IdFam,IdMed,IdEnf),
                                         medEnfFamilia(IdFam,IdMed,IdEnf), R).

medEnfFamiliaIdMed(IdMed, R) :- solucoes(medEnfFamilia(IdFam,IdMed,IdEnf),
                                         medEnfFamilia(IdFam,IdMed,IdEnf), R).

medEnfFamiliaIdEnf(IdEnf, R) :- solucoes(medEnfFamilia(IdFam,IdMed,IdEnf),
                                         medEnfFamilia(IdFam,IdMed,IdEnf), R).
```

Figura 72 - Predicados médico e enfermeiro da família por critérios de seleção

```
exameIdConsulta(IdConsulta, R) :- solucoes(exame(IdConsulta,D),
                                           exame(IdConsulta,D), R).

exameDescricao(D, R) :- solucoes(exame(IdConsulta,D),
                                 exame(IdConsulta,D), R).
```

Figura 73 - Predicados exame por critérios de seleção

3.5.3. Identificar serviços prestados por instituição/cidade/datas/custo

Neste tópico, decidimos estender o enunciado proposto e fazer duas diferentes interpretações do mesmo: interpretar serviços prestados como **consultas** e interpretar como o próprio **serviço.**

No caso da consulta, no que diz respeito às **datas** e aos **custos**, os predicados já foram anteriormente construídos, visto que esta informação pode ser retirada diretamente da própria consulta. Na *Figura 68 - Predicados consulta por critérios de seleção* encontra-se então a consulta por data (***consultaData***) e a consulta por custo (***consultaCusto***).

Para a procura através da **instituição** (Figura 74) e da **cidade** (Figura 75), o raciocínio já é um pouco diferente uma vez que inicialmente temos que encontrar os serviços disponíveis através desses critérios, e só depois encontrar as consultas que realizaram esse serviço. Para a última parte mencionada, foi essencial o predicado auxiliar *servicosParaConsultasServicos* (Figura 76), que é o responsável por converter os serviços identificados para consultas.

```
consultasPorInstituicao(Instituicao, Resultado) :-
    servicosPorInstituicao(Instituicao, I1),
    servicosParaConsultasServicos(I1, Resultado).
```

Figura 74 - Predicado 'consultasPorInstituicao'

```
consultasPorCidade(Cidade, Resultado) :-
    servicosPorCidade(Cidade, I1),
    servicosParaConsultasServicos(I1, Resultado).
```

Figura 75 - Predicado 'consultasPorCidade'

```
servicosParaConsultasServicos([],[]).
servicosParaConsultasServicos([servico(Id,D,Ins,C)|XS], R) :-
    servicosParaConsultasServicos(XS, I1),
    consultaIdServico(Id,I2),
    concat(I1, I2, I3),
    eliminaRepetidos(I3, R).
```

Figura 76 – Predicado 'servicosParaConsultasServicos'

Na figura seguinte encontra-se um exemplo prático da aplicação de ***consultasPorCidade***, onde pesquisamos todas as consultas que se realizaram em Braga.

```
| ?- listing.
consulta(3, '05-02-2019', 2, 4, 15).
consulta(4, '13-12-2018', 2, 1, 20).
consulta(9, '02-04-2017', 2, 2, 40).
consulta(10, '04-04-2017', 5, 7, 10).
consulta(11, '12-12-2017', 10, 8, 0).

servico(1, 'Pediatria', 'Hospital Santa Maria', 'Porto').
servico(2, 'Ortopedia', 'Hospital de Braga', 'Braga').
servico(4, 'Cardiologia', 'Hospital de Braga', 'Braga').
servico(9, 'Curativo', 'Hospital Egas Moniz', 'Lisboa').
servico(10, 'Vacina', 'Hospital de Braga', 'Braga').

yes
| ?- consultasPorCidade('Braga',R).
R = [consulta(9,'02-04-2017',2,2,40),consulta(3,'05-02-2019',2,4,15)] ? yes
yes
```

Figura 77 - Exemplo 'consultasPorCidade'

No que diz respeito a predicados de procura de **serviços** através de **instituições**, **cidades**, **datas** e **custos**, podemos desde já mencionar que no caso da instituição e da cidade os predicados já foram construídos na Secção 3.5.2, tal como podemos ver na *Figura 67 - Predicados serviço por critérios de seleção.*

Para as datas e para os custos, tivemos que encontrar inicialmente quais eram as consultas que faziam *match* com o critério de seleção escolhido e só depois converter essas consultas para o serviço que elas prestaram. Na Figura 78 encontra-se o predicado construído para a procura através da data (***servicosPorData***), e na Figura 79 encontra-se o predicado construído para a procura através do custo (***servicosPorCusto***).

```
servicosPorData(Data, R) :-
    solucoes(IdS, consulta(Id,Data,IdU,IdS,C), I1),
    eliminaRepetidos(I1, I2),
    idServicoParaServico(I2, R).
```

Figura 78 - Predicado 'servicosPorData'

```
servicosPorCusto(Custo, R) :-
    solucoes(IdS, consulta(Id,D,IdU,IdS,Custo), I1),
    eliminaRepetidos(I1, I2),
    idServicoParaServico(I2, R).
```

Figura 79 - Predicado 'servicosPorCusto'

Para o caso dos serviços por custo (***servicosPorCusto***), exemplificamos o funcionamento do predicado pesquisando os serviços que já tinham tido um custo de 10.

```
| ?- listing.
consulta(1, '01-01-2018', 1, 1, 10).
consulta(3, '05-02-2019', 2, 4, 15).
consulta(4, '13-12-2018', 2, 1, 20).
consulta(5, '09-09-2017', 3, 5, 10).
consulta(10, '04-04-2017', 5, 7, 10).
consulta(11, '12-12-2017', 10, 8, 0).

servico(1, 'Pediatria', 'Hospital Santa Maria', 'Porto').
servico(4, 'Cardiologia', 'Hospital de Braga', 'Braga').
servico(5, 'Cirurgia Geral', 'Hospital Egas Moniz', 'Lisboa').
servico(7, 'Genetica Medica', 'Hospital Pulido Valente', 'Lisboa').
servico(9, 'Curativo', 'Hospital Egas Moniz', 'Lisboa').
servico(10, 'Vacina', 'Hospital de Braga', 'Braga').

yes
| ?- servicosPorCusto(10,R).
R = [servico(1,'Pediatria','Hospital Santa Maria','Porto'),servico(5,'Cirurgia Geral','Hospital Egas Moniz','Lisboa'),
servico(7,'Genetica Medica','Hospital Pulido Valente','Lisboa')] ? yes
yes
```

Figura 80 - Exemplo 'servicosPorCusto'

3.5.4. Identificar os utentes de um serviço/instituição

O procedimento de utentes que permite identificar os utentes que já usufruíram de um determinado serviço é o ***utentesServico***, que se encontra na Figura 81.

Neste predicado começamos por, através do predicado auxiliar *soluções* (Figura 102), colocar em I1 a lista de todos os ids dos utentes que já compareceram em consultas do serviço passado como parâmetro. Em seguida, transformamos a lista I1 num conjunto I2, sem repetições, visto que um utente poderá ter ido a mais do que uma consulta do mesmo serviço (*Figura 110 - Predicado 'eliminaRepetidos')*.

Assim, temos em I2 a lista dos ids dos utentes, bastando agora converter esses ids para a própria extensão do predicado utente, ou seja, com todas as informações que um utente contém. Para isso, recorremos à função auxiliar *IdUtenteParaUtente*.

```
utentesServico(IdS,R) :-
    solucoes(IdU, consulta(Id,D,IdU,IdS,C), I1),
    eliminaRepetidos(I1,I2),
    idUtenteParaUtente(I2,R).

idUtenteParaUtente([],[]).
idUtenteParaUtente([X|XS], [U|I]) :- idUtenteParaUtente(XS, I), utenteID(X,[U]).
```

Figura 81 - Predicado 'utentesServico'

Na figura seguinte apresentamos um exemplo prático do uso do predicado ***utentesServico***, onde procuramos identificar quais são os utentes que já usufruíram do serviço 1 ('Pedriatria'). Recordamos que na **consulta** o terceiro campo corresponde ao id do utente e o quarto campo corresponde ao id do serviço.

```
| ?- listing.
consulta(1, '01-01-2018', 1, 1, 10).
consulta(2, '23-05-2016', 1, 1, 14).
consulta(3, '05-02-2019', 2, 4, 15).
consulta(4, '13-12-2018', 2, 1, 20).
consulta(5, '09-09-2017', 3, 5, 10).
consulta(6, '26-10-2016', 1, 3, 15).

servico(1, 'Pediatria', 'Hospital Santa Maria', 'Porto').

utente(1, 'Carlos', 2, 'M', 1, 'Porto').
utente(2, 'Joao', 20, 'M', 1, 'Braga').
utente(3, 'Vilaca', 20, 'M', 2, 'Braga').
utente(4, 'Catarina', 20, 'F', 3, 'Braga').

yes
| ?- utentesServico(1,R).
R = [utente(1,'Carlos',2,'M',1,'Porto'),utente(2,'Joao',20,'M',1,'Braga')] ? yes
yes
```

Figura 82 - Exemplo 'utentesServico'

Em segundo lugar, construímos o predicado que permite identificar os utentes que já visitaram uma determinada instituição, passada então como parâmetro à função ***utentesInstituicao***.

Nesse predicado, com o auxílio da função *servicoInstituicao* (Figura 67), convertemos a instituição passada como parâmetro (Inst) para os serviços que são hospedados nessa instituição (que ficam armazenados em I).

Em seguida, essa lista I é passada como argumento à função auxiliar *servicosParaUtentesServicos*, que é responsável por converter esses serviços nos respetivos utentes que usufruíram destes. Para isso, foi necessário recorrer aos procedimentos *utentesServico* (Figura 65), a *concat* (Figura 112) e a *eliminaRepetidos* (Figura 110).

```
utentesInstituicao(Inst,R) :-
    servicoInstituicao(Inst, I),
    servicosParaUtentesServicos(I, R).

servicosParaUtentesServicos([],[]).
servicosParaUtentesServicos([servico(Id,D,Ins,C)|XS], R) :-
    servicosParaUtentesServicos(XS, I1),
    utentesServico(Id,I2),
    concat(I1, I2, I3),
    eliminaRepetidos(I3, R).
```

Figura 83 - Predicado 'utentesInstituicao'

Assim, na figura seguinte encontra-se um exemplo prático da aplicação do predicado ***utentesInstituicao***, que pesquisa todos os utentes que já visitaram a instituição 'Hospital Pulido Valente'. Mais uma vez, recordamos que o terceiro campo da **consulta** corresponde ao id do utente e o quarto campo corresponde ao id do serviço.

```
| ?- listing.
consulta(7, '29-02-2017', 3, 3, 30).
consulta(8, '10-03-2016', 4, 6, 20).
consulta(9, '02-04-2017', 2, 2, 40).
consulta(10, '04-04-2017', 5, 7, 10).
consulta(11, '12-12-2017', 10, 8, 0).

servico(6, 'Estomatologia', 'Hospital Egas Moniz', 'Lisboa').
servico(7, 'Genetica Medica', 'Hospital Pulido Valente', 'Lisboa').
servico(8, 'Medicina Nuclear', 'Hospital Pulido Valente', 'Lisboa').

utente(5, 'Eva', 0, 'F', 3, 'Braga').
utente(6, 'Maria Jose', 56, 'F', 1, 'Guimaraes').
utente(10, 'Ana', 34, 'F', 6, 'Guimaraes').

yes
| ?- utentesInstituicao('Hospital Pulido Valente',R).
R = [utente(10,'Ana',34,'F',6,'Guimaraes'),utente(5,'Eva',0,'F',3,'Braga')] ? yes
yes
```

Figura 84 - Exemplo 'utentesInstituicao'

3.5.5. Identificar serviços realizados por utente/instituição/cidade

Para identificar todos serviços que um determinado utente já usufruiu construímos o predicado ***servicosPorUtente*** (Figura 85).

Numa primeira fase, começamos por identificar todos os ids dos serviços que o utente com o id passado como parâmetro usufruiu, com a ajuda do predicado *soluções* (Figura 102), colocando o resultado em I1. Em seguida, são eliminadas todas as repetições (*Figura 110 - Predicado 'eliminaRepetidos')* e, por fim, convertemos os ids dos serviços encontrados nos respetivos serviços, com a ajuda do predicado auxiliar *idServicoParaServico*.

```
servicosPorUtente(IdUtente, R) :-
    solucoes(IdS, consulta(Id,D,IdUtente,IdS,C), I1),
    eliminaRepetidos(I1, I2),
    idServicoParaServico(I2, R).

idServicoParaServico([],[]).
idServicoParaServico([X|XS], [U|I]) :- idServicoParaServico(XS, I), servicoID(X,[U]).
```

Figura 85 - Predicado 'servicosPorUtente'

Na figura seguinte podemos ver um exemplo de aplicação do predicado anteriormente descrito, onde encontramos todos os serviços que o utente com id 1 usufruiu.

```
| ?- listing.
consulta(1, '01-01-2018', 1, 1, 10).
consulta(2, '23-05-2016', 1, 1, 14).
consulta(3, '05-02-2019', 2, 4, 15).
consulta(4, '13-12-2018', 2, 1, 20).
consulta(5, '09-09-2017', 3, 5, 10).
consulta(6, '26-10-2016', 1, 3, 15).

servico(1, 'Pediatria', 'Hospital Santa Maria', 'Porto').
servico(2, 'Ortopedia', 'Hospital de Braga', 'Braga').
servico(3, 'Oftalmologia', 'Hospital Santa Maria', 'Porto').
servico(4, 'Cardiologia', 'Hospital de Braga', 'Braga').

utente(1, 'Carlos', 2, 'M', 1, 'Porto').

yes
| ?- servicosPorUtente(1,R).
R = [servico(1,'Pediatria','Hospital Santa Maria','Porto'),servico
(3,'Oftalmologia','Hospital Santa Maria','Porto')] ? yes
yes
```

Figura 86 - Exemplo 'servicosPorUtente'

Para identificar os serviços através da instituição (*Figura 87 - Predicado 'servicoPorInstituicao'*) e da cidade (*Figura 88 - Predicado 'servicoPorCidade'*), uma vez que ambos os critérios se encontram diretamente na extensão do predicado **serviço**, basta utilizar o predicado auxiliar *soluções* (Figura 102) para encontrar os serviços pretendidos.

```
servicosPorInstituicao(IdI, R) :-
    solucoes(servico(IdS,D,IdI,C), servico(IdS,D,IdI,C), R).
```

Figura 87 - Predicado 'servicoPorInstituicao'

```
servicosPorCidade(Cidade, R) :-
    solucoes(servico(IdS,D,IdI,Cidade), servico(IdS,D,IdI,Cidade), R).
```

Figura 88 - Predicado 'servicoPorCidade'

3.6. Custos

3.6.1. Calcular o custo total dos cuidados de saúde por utente/serviço/instituição/data

Todas as entidades que prestam serviços à população requerem imprescindivelmente funcionalidades de cálculo de custos.

Em consequência, numa área da prestação de cuidados saúde podem ser calculados os **custos** através de vários critérios de seleção: por **utente**, por **serviço**, por **instituição** e por **data**.

É a consulta que tem um custo associado e, em seguida, vamos apresentar e descrever como é que obtivemos o custo total através de cada um dos critérios pedidos.

No caso do cálculo de custos por **utente**, construímos o predicado ***totalUtente*** (Figura 89), onde através do id do utente passado como parâmetro recorremos ao predicado auxiliar *soluções* (Figura 102) para obtermos a lista de todos os custos que o determinado utente obteve com consultas e, posteriormente, através do predicado auxiliar *soma* (Figura 111), somamos esses custos e obtivemos o total de custos.

```
totalUtente(IdUtente, Total) :-
    solucoes(Custo, consulta(_,_,IdUtente,_,Custo), Lista),
    soma(Lista, Total).
```

Figura 89 - Predicado 'totalUtente'

Na figura seguinte, podemos visualizar o total de custos do utente com o id 2.

```
| ?- listing.
consulta(1, '01-01-2018', 1, 1, 10).
consulta(3, '05-02-2019', 2, 4, 15).
consulta(4, '13-12-2018', 2, 1, 20).
consulta(5, '09-09-2017', 3, 5, 10).
consulta(9, '02-04-2017', 2, 2, 40).
consulta(11, '12-12-2017', 10, 8, 0).

utente(2, 'Joao', 20, 'M', 1, 'Braga').

yes
| ?- totalUtente(2,R).
R = 75 ? yes
yes
```

Figura 90 - Exemplo 'totalUtente'

No cálculo de custos por **serviço**, o raciocínio desenvolvimento foi equivalente. Assim, o predicado construído foi o seguinte.

```
totalServico(IdServ, Total) :-
    solucoes(Custo, consulta(_,_,_,IdServ,Custo), Lista),
    soma(Lista, Total).
```

Figura 91 - Predicado 'totalServico'

Exemplificamos a utilização desse predicado, obtendo o total de custos que já existiram com o serviço com o id 3, 'Oftalmologia'.

```
| ?- listing.
consulta(3, '05-02-2019', 2, 4, 15).
consulta(4, '13-12-2018', 2, 1, 20).
consulta(5, '09-09-2017', 3, 5, 10).
consulta(6, '26-10-2016', 1, 3, 15).
consulta(7, '29-02-2017', 3, 3, 30).

servico(3, 'Oftalmologia', 'Hospital Santa Maria', 'Porto').

yes
| ?- totalServico(3,R).
R = 45 ? yes
yes
```

Figura 92 - Exemplo 'totalServico'

No caso do cálculo através do nome da **instituição**, uma vez a consulta não possui diretamente esse campo, foi necessário identificar os ids dos serviços que atuam na instituição passada como parâmetro (com a ajuda do predicado *soluções*) e, em seguida, através do predicado *somaServicos*, que, com recurso ao predicado mencionado anteriormente (*totalServico*), somou todos os custos de cada uma das consultas que prestaram os serviços em questão.

```
totalInstituicao(NomeInstituicao, Total) :-
    solucoes(IdServico, servico(IdServico,_,NomeInstituicao,_), Lista),
    somaServicos(Lista, Total).

somaServicos([], 0).
somaServicos([X|XS], Total) :-
    somaServicos(XS, Acumulada),
    totalServico(X, TotalServico),
    Total is TotalServico + Acumulada.
```

Figura 93 - Predicado 'totalInstituicao'

Na figura seguinte podemos ver o total de gastos no Hospital de Braga.

```
| ?- listing.
consulta(3, '05-02-2019', 2, 4, 15).
consulta(4, '13-12-2018', 2, 1, 20).
consulta(5, '09-09-2017', 3, 5, 10).
consulta(6, '26-10-2016', 1, 3, 15).
consulta(7, '29-02-2017', 3, 3, 30).
consulta(9, '02-04-2017', 2, 2, 40).

servico(1, 'Pediatria', 'Hospital Santa Maria', 'Porto').
servico(2, 'Ortopedia', 'Hospital de Braga', 'Braga').
servico(4, 'Cardiologia', 'Hospital de Braga', 'Braga').
servico(7, 'Genetica Medica', 'Hospital Pulido Valente', 'Lisboa').
servico(10, 'Vacina', 'Hospital de Braga', 'Braga').

yes
| ?- totalInstituicao('Hospital de Braga',R).
R = 55 ? yes
yes
```

Figura 94 - Exemplo 'totaInstituicao'

Para o custo total através da **data**, o desenvolvimento do predicado seguiu a mesma linha de raciocínio do custo através do utente e do serviço.

```
totalData(Data, Total) :-
    solucoes(Custo, consulta(_,Data,_,_,Custo), Lista),
    soma(Lista, Total).
```

Figura 95 - Predicado 'totalData'

3.7. Mais extras

Para que o nosso sistema de representação de conhecimento e raciocínio fosse o mais completo possível, desenvolvemos vários requisitos extra. Em consequência disso, ao longo do desenvolvimento do presente relatório, fomos apresentando e descrevendo todos os componentes extra implementados.

Ao nível da **extensão do conhecimento a tratar**, tal como referido na Secção 3.1, adicionamos os predicados médico, enfermeiro, médico e enfermeiro da família e exame. Adicionamos também o género e o id da família ao predicado utente e um id à consulta. Assim, em relação às funcionalidades de adição e de remoção, implementamos também procedimentos para registar e para remover conhecimento de médicos, enfermeiros, médicos e enfermeiros da família e exames, como se pode ver na Secção 3.3.1 e 3.4.1. Em consequência, construímos também os respetivos invariantes (Secção 3.2).

Para todo o conhecimento estendido, implementamos ainda os respetivos procedimentos de identificação através de todos os critérios de seleção possíveis, tal como se pode ver na Secção 3.5.2.

No tópico que diz respeito às identificações, fizemos duas interpretações na questão *Identificar serviços prestados por instituição/cidade/datas/custo*, Secção 3.5.3, onde construímos procedimentos tanto para encontrar os serviços, como para encontrar as consultas.

Ao nível das **faculdades de raciocínio**, implementamos um predicado que permite ordenar os utentes por ordem ascedente de dinheiro gasto em consultas, que abordaremos na Secção 3.7.3. Implementamos ainda um predicado que nos permite guardar o estado, mais precisamente os factos do nosso sistema, em ficheiro, descrito mais adiante na Secção 3.7.2.

Por fim, salientamos que todo o nosso código se encontra organizado segundo o paradigma modular, separando as responsabilidades por vários módulos, tal como referimos na secção seguinte.

3.7.1. Modularidade de Código

Tal como já mencionado, criamos vários ficheiros com diferentes responsabilidades para que o nosso trabalho se encontre o mais modular possível. Deste modo, dispomos de um ficheiro *funçõesAuxiliares.pl*, com todas os predicados auxiliares necessários, *identificações.pl* responsável pelos predicados que dizem respeito às identificações requeridas e *calculoDeCustos.pl* com todos os predicados de custos. Temos ainda um ficheiro *state.pl* que armazena todos os factos do nosso sistema e *stateControl.pl*, responsável pelos predicados que permitem guardar o conhecimento atual no sistema. Por fim, temos o nosso ficheiro principal, *exercicio1.pl*, que inclui todos esses ficheiros, tal como se pode ver na Figura 96, e que dispõe das funcionalidades de adição e remoção de conhecimento e dos invariantes.

```
:- include('state.pl').
:- include('funcoesAuxiliares.pl').
:- include('identificacoes.pl').
:- include('stateControl.pl').
:- include('calculoDeCustos.pl').
```

Figura 96 - 'Includes' presentes no ficheiro exercicio1.pl

3.7.2. Guardar Estado

Para que caso pretendamos adicionar ou remover conhecimento através dos predicados que implementamos para o efeito, e queremos que esse conhecimento fique persistentemente armazenado no nosso sistema, construímos um predicado ***saveState***, ilustrado na Figura 97, que guarda no ficheiro *state.pl* todos os factos que se encontram de momento no sistema.

Para isso, recorremos a três funções já existentes em PROLOG, *open*, *write* e *close*. O *open* abre o ficheiro que guardará os factos em modo de escrita e a Stream responsável pelo *input/output*. O *write* vai escrevendo para a Stream o conteúdo que pretendemos armazenar no ficheiro e, por fim, o *close* fecha a Stream e o seu conteúdo é então escrito no ficheiro desejado.

Cada um dos tipos de conhecimento existentes possui um procedimento que armazena os seus próprios factos da forma que nos é conveniente. Esses predicados encontram-se ilustrados na Figura 98.

```
saveState :-
    open('state.pl', write, Stream),
    write(Stream, '% utente: #IdUt,Nome,Idade,Género,#IdFamilia,Cidade -> {V,F}\n'),
    saveUtente(Stream),
    write(Stream, '\n% serviço: #IdServ,Descrição,Instituição,Cidade -> {V,F}\n'),
    saveServico(Stream),
    write(Stream, '\n% consulta: #IdConsulta,Data,#IdUt,#IdServ,Custo -> {V,F}\n'),
    saveConsulta(Stream),
    write(Stream, '\n% medico: #IdMedico,Nome,Idade,Género,#IdServ -> {V,F}\n'),
    saveMedico(Stream),
    write(Stream, '\n% enfermeiro: #IdEnfermeiro,Nome,Idade,Género,#IdServ -> {V,F}\n'),
    saveEnfermeiro(Stream),
    write(Stream, '\n% medEnfFamilia: #IdFamilia,#IdMedico,#IdEnfermeiro -> {V,F}\n'),
    saveMedEnfFamilia(Stream),
    write(Stream, '\n% exame: #IdConsulta,Descrição -> {V,F}\n'),
    saveExame(Stream),
    close(Stream).
```

Figura 97 - Predicado 'saveState'

```
saveUtente(Stream) :- utente(A,B,C,D,E,F),
        write(Stream, 'utente('),write(Stream, A), write(Stream, ',\''),
        write(Stream, B), write(Stream, '\','), write(Stream, C),
        write(Stream, ',\''), write(Stream, D), write(Stream, '\','),
        write(Stream, E), write(Stream, ',\''),
        write(Stream, F), write(Stream, '\').\n'),
    fail; true.

saveServico(Stream) :- servico(A,B,C,D),
        write(Stream, 'servico('), write(Stream, A), write(Stream, ',\''),
        write(Stream, B), write(Stream, '\',\''), write(Stream, C),
        write(Stream, '\',\''), write(Stream, D), write(Stream, '\').\n'),
    fail; true.

saveConsulta(Stream) :- consulta(A,B,C,D,E),
        write(Stream, 'consulta('), write(Stream, A),
        write(Stream, ',\''), write(Stream, B), write(Stream, '\','),
        write(Stream, C), write(Stream, ','), write(Stream, D),
        write(Stream, ','), write(Stream, E), write(Stream, ').\n'),
    fail; true.

saveMedico(Stream) :- medico(A,B,C,D,E),
        write(Stream, 'medico('), write(Stream, A), write(Stream, ',\''),
        write(Stream, B), write(Stream, '\','), write(Stream, C),
        write(Stream, ',\''), write(Stream, D), write(Stream, '\','),
        write(Stream, E), write(Stream, ').\n'),
    fail; true.

saveEnfermeiro(Stream) :- enfermeiro(A,B,C,D,E),
        write(Stream, 'enfermeiro('),write(Stream, A), write(Stream, ',\''),
        write(Stream, B), write(Stream, '\','), write(Stream, C),
        write(Stream, ',\''), write(Stream, D), write(Stream, '\','),
        write(Stream, E), write(Stream, ').\n'),
    fail; true.

saveMedEnfFamilia(Stream) :- medEnfFamilia(A,B,C),
        write(Stream, 'medEnfFamilia('),write(Stream, A), write(Stream, ','),
        write(Stream, B), write(Stream, ','), write(Stream, C),
        write(Stream, ').\n'),
    fail; true.

saveExame(Stream) :- exame(A,B),
        write(Stream, 'exame('),write(Stream, A), write(Stream, ',\''),
        write(Stream, B), write(Stream, ').\n'),
    fail; true.
```

Figura 98 - Predicados auxiliares de 'saveState'

Apresentamos agora um exemplo prático de utilização desta funcionalidade.

```
| ?- listing.
utente(8, 'Helena', 25, 'F', 5, 'Lisboa').
utente(9, 'Guilherme', 58, 'M', 1, 'Porto').

yes
| ?- novoUtente(10,'Ana',34,'F',6,'Guimaraes').
yes
| ?- saveState.
yes
| ?- ^C
Prolog interruption (h for help)? e

SRCR/exercicio1 on  master [!] took 13s
→ sicstus -l exercicio1.pl
% compiling /Users/cam/Code/UM/SRCR/exercicio1/exercicio1.pl...
%  including /Users/cam/Code/UM/SRCR/exercicio1/state.pl...
%  included /Users/cam/Code/UM/SRCR/exercicio1/state.pl in module user, 0 msec 10256 bytes
%  including /Users/cam/Code/UM/SRCR/exercicio1/funcoesAuxiliares.pl...
%  included /Users/cam/Code/UM/SRCR/exercicio1/funcoesAuxiliares.pl in module user, 50 msec 596272 bytes
%  including /Users/cam/Code/UM/SRCR/exercicio1/identificacoes.pl...
%  included /Users/cam/Code/UM/SRCR/exercicio1/identificacoes.pl in module user, 10 msec 35072 bytes
%  including /Users/cam/Code/UM/SRCR/exercicio1/stateControl.pl...
%  included /Users/cam/Code/UM/SRCR/exercicio1/stateControl.pl in module user, 0 msec 23680 bytes
%  including /Users/cam/Code/UM/SRCR/exercicio1/calculoDeCustos.pl...
%  included /Users/cam/Code/UM/SRCR/exercicio1/calculoDeCustos.pl in module user, 0 msec 6336 bytes
% compiled /Users/cam/Code/UM/SRCR/exercicio1/exercicio1.pl in module user, 100 msec 943152 bytes
SICStus 4.3.0 (x86_64-darwin-13.1.0): Thu May  8 05:39:06 PDT 2014
Licensed to studentSP4.3di.uminho.pt
| ?- listing.
utente(8, 'Helena', 25, 'F', 5, 'Lisboa').
utente(9, 'Guilherme', 58, 'M', 1, 'Porto').
utente(10, 'Ana', 34, 'F', 6, 'Guimaraes').

yes
```

Figura 99 - Exemplo 'saveState'

Como se pode ver na Figura 99, inicialmente, o sistema possui apenas 2 utentes e, com recurso à função *novoUtente,* é registado um novo utente. Em seguida, invocamos então o predicado ***saveState***. Terminando a sessão do *sicstusus* e voltando a iniciá-la, vemos que agora o conhecimento se encontra efetivamente alargado, isto é, o nosso sistema possui 3 utentes.

3.7.3. Ordenar por quem gastou mais dinheiro

O utilizador pode consultar quais os N utentes com mais dinheiro gasto em consultas através do uso do predicado $utentesComMaisDinheiroGasto(N, Resultado)$ ilustrado em seguida.

```
% Ordenar por quem gastou mais dinheiro

utentesComMaisDinheiroGasto(Tamanho, Resultado) :-
    solucoes(Id, utente(Id,N,I,G,F,C), I1),
    utenteParaParUtenteGastoTotal(I1, I2),
    mergeSortKeyValuePairASC(I2,I3),
    take(I3,Tamanho,I4),
    idUtenteParaUtente(I4,Resultado).
```

Figura 100 - Predicado 'utentesComMaisDinheiroGasto'

Este predicado reúne todos os identificadores de utentes presentes na base de conhecimento, calcula para cada um deles o seu gasto total e, em seguida, ordena recorrendo ao predicado descrito na *Figura 113 - Predicado 'mergeSortKeyValuePairASC'* para ordenar os utentes por gasto total por ordem ascendente.

Por fim, é apenas necessário pegar nos N primeiros elementos da lista através do predicado descrito na *Figura 114 - Predicado 'take'* e converter os identificadores de utentes para utentes através do predicado *utenteID* (Figura 65 - Predicados utente por critérios de seleção).

```
| ?- utentesComMaisDinheiroGasto(5,R).
R = [(utente(2,'Joao',20,'M',1,'Braga'),75),(utente(3,'Vilaca',20,'M',2,'Braga'),40)
,(utente(1,'Carlos',2,'M',1,'Porto'),39),(utente(4,'Catarina',20,'F',3,'Braga'),20),
(utente(5,'Eva',0,'F',3,'Braga'),10)] ? yes
yes
```

Figura 101 - Exemplo "utentesComMaisDinheiroGasto"

4. Conclusões e Sugestões

No final deste exercício, a equipa conclui que foi desenvolvido com sucesso um sistema de caraterização de um universo na área da prestação de cuidados de saúde. Este sistema inclui diversos tipos de conhecimento como utentes, consultas, exames, entre outros.

Foram ainda implementadas todas as funcionalidades solicitadas que permitem uma correta manipulação da base de conhecimento (quer adição quer remoção de conhecimento) e a aplicação de uma série de *queries* sobre ela.

Em futuras evoluções deste trabalho poderá, por exemplo, ser estendido o conhecimento, através da inclusão de, por exemplo, auxiliares de saúde, equipamentos médicos, entre muitos outros. Seria ainda possível que futuras versões permitissem pesquisas mais refinadas na base de conhecimento, acrescentando novos predicados de identificação.

A boa preparação da equipa e o estudo prévio, quer do motor de inferência quer do caso de estudo, possibilitaram que todo o processo decorresse dentro da normalidade, sem que tenham surgido problemas de dimensão considerável.

5. Referências

5.1. Referências Bibliográficas

[Analide, 2011] ANALIDE, Cesar, NOVAIS, Paulo, Neves, José,

"Sugestões para a Redacção de Relatórios Técnicos",

Relatório Técnico, Departamento de Informática, Universidade do Minho, Portugal, 2011.

[Bratko, 2000] BRATKO, Ivan,

"PROLOG: Programming for Artificial Intelligence",

United States, Pearson Education (us), 2000.

5.2. Referências Eletrónicas

CATEGORIA: Exames médicos. Disponível em: https://pt.wikipedia.org/wiki/Categoria:Exames_m%C3%A9dicos. Acesso em: 11 mar. 2019.

LISTA de Especialidades Médicas. Disponível em: https://pt.wikipedia.org/wiki/Lista_de_especialidades_m%C3%A9dicas. Acesso em: 11 mar. 2019.

ORDEM dos Enfermeiros: Especialidades. Disponível em: https://www.ordemenfermeiros.pt/faqs/especialidades/. Acesso em: 11 mar. 2019.

PROLOG. Disponível em: https://pt.wikibooks.org/wiki/Prolog. Acesso em: 13 mar. 2019.

SICStus Prolog. Disponível em: https://sicstus.sics.se/sicstus/docs/latest4/html/sicstus.html/. Acesso em: 15 mar. 2019.

6. Anexos

6.1. Funções Auxiliares

Para o desenvolvimento dos procedimentos requeridos pelo nosso sistema de representação de conhecimento e raciocínio, recorremos a várias funções auxiliares que foram extremamente úteis no decorrer de todo o processo.

Em seguida, apresentamos essas funções desenvolvidas, que fomos referindo ao longo do desenvolvimento do presente relatório.

O meta-predicado **soluções** que coloca em R a lista com as soluções do tipo X para todo o XS que encontrar.

```
% Encontra todas as soluções
solucoes(X, XS, R) :- XS, assert(tmp(X)), fail.
solucoes(X, XS, R) :- solucoesAux([], R).

solucoesAux(L, R) :- retract(tmp(X)), !, solucoesAux([X|L], R).
solucoesAux(R, R).
```

Figura 102 – Meta-predicado 'solucoes'

O predicado **comprimento**, que coloca em R o comprimento da lista passada como argumento.

```
% Comprimento de uma lista
comprimento([], 0).
comprimento([_|XS], R) :- comprimento(XS, S), R is 1 + S.
```

Figura 103 - Predicado 'comprimento'

O meta-predicado **inserção** que coloca Q na base de conhecimento no caso de sucesso, retornando *yes*, e retira Q no caso de haver retrocesso, retornando *no*. O meta-predicado **remoção**, que faz o oposto do meta-predicado **inserção**, ou seja, elimina Q da base de conhecimento no caso de sucesso e adiciona Q no caso de haver retrocesso, retornando *no*.

```
% Inserção de conhecimento
insercao(Q) :- assert(Q).
insercao(Q) :- retract(Q), !, fail.
```

Figura 104 – Meta-predicado 'insercao'

```
% Remoção de conhecimento
remocao(Q) :- retract(Q).
remocao(Q) :- assert(Q), !, fail.
```

Figura 105 – Meta-predicado 'remocao'

O meta-predicado **teste**, que testa se todos os predicados passados como parâmetro são verdadeiros.

```
% Testa se todos os predicados são verdadeiros
teste([]).
teste([I|L]) :- I, teste(L).
```

Figura 106 – Meta-predicado 'teste'

O meta-predicado **evolução**, responsável por adicionar novo conhecimento na base de conhecimento, verificando se todos os invariantes estruturais e referenciais de adição do conhecimento que se pretende adicionar (T) continuam verdadeiros. Só em caso de sucesso é que o conhecimento fica efetivamente registado, em caso contrário, o conhecimento inicialmente inserido é transparentemente removido.

```
% Insere novo conhecimento na base de conhecimento
evolucao(T) :- solucoes(I, +T::I, Linv),
               insercao(T),
               teste(Linv).
```

Figura 107 – Meta-predicado 'evolucao'

O meta-predicado **involução**, responsável por remover conhecimento da base de conhecimento, verificando se todos os invariantes estruturais e referenciais de remoção do conhecimento que se pretende remover (T) continuam verdadeiros. Só em caso de sucesso é que o conhecimento é permanentemente removido, em caso contrário, o conhecimento é novamente inserido.

```
% Retira conhecimento da base de conhecimento
involucao(T) :- solucoes(I, -T::I, Linv),
                remocao(T),
                teste(Linv).
```

Figura 108 – Meta-predicado 'involucao'

O predicado **pertence** que averigua se A pertence à lista passada como parâmetro e o predicado **elimina repetidos**, que elimina todos os elementos repetidos em X, colocando em R a nova lista, respetivamente.

```
% Averigua se elemento pertence a uma lista
pertence(A,[A|XS]).
pertence(A,[X|XS]) :- pertence(A,XS).
```

Figura 109 - Predicado 'pertence'

```
% Elimina os elementos repetidos numa lista
eliminaRepetidos(X, R) :- eliminaRepAux(X,[],R).

eliminaRepAux([],Acc,Acc).
eliminaRepAux([X|XS],Acc,R) :- pertence(X,Acc), eliminaRepAux(XS,Acc,R).
eliminaRepAux([X|XS],Acc,R) :- eliminaRepAux(XS,[X|Acc],R).
```

Figura 110 - Predicado 'eliminaRepetidos'

O predicado **soma**, que soma todos os elementos da lista passada como argumento, colocando em Total o resultado.

```
% Somatório
soma([],0).
soma([X|XS],Total) :- soma(XS, Acumulado), Total is X + Acumulado.
```

Figura 111 - Predicado 'soma'

O predicado ***concat***, que concatena duas listas, colocando o resultado no terceiro argumento da função.

```
% Concatenar 2 listas
concat([], R, R).
concat([X|XS1], R, [X|XS2]) :- concat(XS1, R, XS2).
```

Figura 112 - Predicado 'concat'

O predicado ***mergeSortKeyValuePairASC*** faz ordenação de uma lista usando como elemento de comparação a segunda componente do tuplo.

```
% Merge Sort
splitlist(L, [], L, 0).
splitlist([H|T], [H|A], B, N) :- Nminus1 is N-1, splitlist(T, A, B, Nminus1).

halfhalf(L, A, B) :- length(L, Len), Half is Len//2, splitlist(L, A, B, Half).

merge(A, [], A).
merge([], B, B).
merge([(X0,X1)|Ta], [(Y0,Y1)|Tb], R) :- X1 > Y1, merge(Ta, [(Y0,Y1)|Tb], M), R = [(X0,X1)|M].
merge([(X0,X1)|Ta], [(Y0,Y1)|Tb], R) :- X1 =< Y1, merge(Tb, [(X0,X1)|Ta], M), R = [(Y0,Y1)|M].

mergeSortKeyValuePairASC([], []).
mergeSortKeyValuePairASC([E], [E]).
mergeSortKeyValuePairASC([(X0,X1), (Y0,Y1)], [(X0,X1), (Y0,Y1)]) :- X1 > Y1.
mergeSortKeyValuePairASC([(X0,X1), (Y0,Y1)], [(Y0,Y1), (X0,X1)]) :- X1 =< Y1.
mergeSortKeyValuePairASC(L, R) :-
    halfhalf(L, A, B),
    mergeSortKeyValuePairASC(A, Asort),
    mergeSortKeyValuePairASC(B, Bsort),
    merge(Asort, Bsort, R).
```

Figura 113 - Predicado 'mergeSortKeyValuePairASC'

O predicado **take**, que coloca no terceiro parâmetro da função os primeiros N elementos da lista passada como primeiro parâmetro.

```
% Pega nos N primeiros de uma lista
take(Lista,0,[]).
take([X|XS], N1, [X|I]) :- N0 is N1-1, take(XS,N0,I).
```

Figura 114 - Predicado 'take'

E, por fim, o predicado **idade válida** (idade >= 0), **custo válido** (custo >= 0) e **género válido** (masculino, ‘M’, ou feminino, ‘F’).

```
% Idade válida (>= 0)
idadeValida(I) :- I >= 0.

% Custo válido (>= 0)
custoValido(C) :- C >= 0.

% Género é masculino ('M') ou feminino ('F')
generoValido('M').
generoValido('F').
```

Figura 115 - Predicado 'idadeValida', 'custoValido' e 'generoValido'