Universidade do Minho

Programação em Lógica e Invariantes

MIEI

Sistemas de Representação de Conhecimento e Raciocínio
(2º Semestre - 2017/2018)

Ana Paula Carvalho - a61855
João Pires Barreira - a73831
Rafael Braga Costa - a61799

Março 2018

Resumo

De modo a consolidar definitivamente os conhecimentos retidos nestes primeiros passos na Programação em Lógica, foi desenvolvido, sob a forma de um exercício prático, um Sistema de Representação de Conhecimento e Raciocínio com a capacidade de caracterizar um universo de discurso na área da prestação de cuidados de saúde. Para este panorama de conhecimento, dado na forma de utentes, prestadores, cuidados e instituições, são exigidas algumas funcionalidades base e, posteriormente, adicionadas algumas outras que achámos pertinentes.

Todo o exercício foi desenvolvido utilizando a linguagem *PROLOG*.

Índice

1 Introdução

No âmbito da unidade curricular de Sistema de Representação de Conhecimento e Raciocínio foi proposta a resolução de um exercício, de modo a incentivar a utilização da linguagem de programação em lógica *PROLOG* e construir mecanismos de raciocínio para resolução de problemas.

Para o desafio lançado foi solicitado o desenvolvimento da caracterização de um universo de discurso na área de cuidados de saúde. Para o efeito, considerou-se um panorama de conhecimento e implementaram-se funcionalidades tais como o registo e a remoção de utentes, prestadores e cuidados de saúde, identificação por critérios de seleção dos mesmos e cálculo do gasto total de um utente em cuidados de saúde.

Foram também desenvolvidas outras funcionalidades, idealizadas pelo grupo, que considerámos acrescentar valências ao sistema. Tudo isto será minuciosamente demonstrado e justificado ao longo deste relatório.

2 Preliminares

Para o desenvolvimento de uma solução para este problema, revelaram-se imprescindíveis todos os conteúdos lecionados até ao momento nas aulas da unidade curricular, tanto teóricas como práticas.

Ocasionalmente, o grupo recorreu a algumas pesquisas na *web* sobre programação lógica e exemplos da sua utilização, obtendo uma perceção mais clara das suas potencialidades.

Para além do referido, uma distribuição uniforme das tarefas pelos elementos do grupo contribuiu para o sucesso deste exercício

3 Descrição do Trabalho e Análise dos Resultados

3.1 Base de conhecimento

O sistema de representação de conhecimento e raciocínio desenvolvido caracteriza uma área de prestação de cuidados de saúde. Deste modo, considerou-se que o conhecimento é dado pelo seguinte modo:

- utente: #IdUt, Nome, Idade, Morada $\rightsquigarrow \{ \mathbb{V}, \mathbb{F} \}$
- prestador: #IdPrest, Nome, Especialidade, Instituição $\rightsquigarrow \{ \mathbb{V}, \mathbb{F} \}$
- cuidado: Data, #IdUt, #IdPrest, Descrição, Custo $\rightsquigarrow \{ \mathbb{V}, \mathbb{F} \}$
- instituição: #IdIns, Nome, Cidade $\rightsquigarrow \{ \mathbb{V}, \mathbb{F} \}$
- consulta: #IdUt, IdPrest, HoraInício, HoraFim $\rightsquigarrow \{ \mathbb{V}, \mathbb{F} \}$

3.1.1 Utente

Um utente é caracterizado por um identificador (número inteiro único para cada utente), um nome, uma idade e uma morada. Os seguintes predicados representam utentes:

```
utente( 0, 'Jose',      45, 'Rua do Queijo').
utente( 1, 'Maria',     41, 'Rua de Cima').
utente( 2, 'Gertrudes', 26, 'Rua Carlos Antonio').
```

```
utente( 3, 'Paula',      73, 'Rua da Mina').
utente( 4, 'Sebastiao', 83, 'Rua da Poeira').
utente( 5, 'Zeca',       9,  'Rua do Poeira').
utente( 6, 'Jorge',      44, 'Rua da Poeira').
utente( 7, 'Rafaela',    23, 'Rua da Poeira').
utente( 8, 'Anabela',    42, 'Rua de Baixo').
utente( 9, 'Antonio',    57, 'Rua do Forno').
utente(10, 'Zueiro',     33, 'Rua do Mar').
```

3.1.2 Prestador

Um prestador é caracterizado por um identificador (tal como um utente), um nome, por uma especialidade e por um identificador de uma instituição onde presta os seus serviços. Optou-se por esta metodologia de modo a uma melhor organização do conhecimento, já que assim pode-se expressar com mais detalhe a informação relativa a uma instituição sem a replicar por cada um dos prestadores envolvidos. Os seguintes predicados representam prestadores:

```
prestador( 0, 'Manuel',    'Cardiologia',  1).
prestador( 1, 'Carlos',    'Neurologia',   2).
prestador( 2, 'Aventino', 'Urologia',     3).
prestador( 3, 'Paulo',     'Ortopedia',    4).
prestador( 4, 'Bicas',     'Psiquiatria',  2).
prestador( 5, 'Ines',      'Pediatria',    3).
prestador( 6, 'Manuela',   'Ginecologia',  4).
prestador( 7, 'Sara',      'Oftalmologia', 4).
prestador( 8, 'Sandra',    'Radiografia',  3).
prestador( 9, 'Ruben',     'Fisioterapia', 2).
prestador(10, 'Luisa',     'Dermatologia', 1).
```

3.1.3 Cuidado prestado

Um cuidado prestado é definido à custa de uma data (DD, MM, AAAA), pelos números identificadores de um utente e de um prestador envolvidos no cuidado prestado, por uma descrição e por um custo (obviamente este valor deve ser positivo). Os seguintes predicados representam exemplos de cuidados prestados:

```
cuidado(data(13, 12, 2017), 0, 2, 'Consulta Urologia', 19.99).
cuidado(data(13, 11, 2017), 0, 1, 'Consulta Neurologia', 39.99).
cuidado(data(10, 11, 2017), 1, 6, 'Consulta Ginecologia', 19.99).
cuidado(data(10, 11, 2017), 1, 6, 'Consulta Ginecologia', 59.99).
cuidado(data(11, 12, 2017), 1, 0, 'Consulta Cardiologia', 59.99).
cuidado(data(11, 12, 2017), 2, 0, 'Consulta Cardiologia', 59.99).
cuidado(data(12, 12, 2017), 2, 7, 'Consulta Oftalmologia', 59.99).
cuidado(data(12, 11, 2017), 2, 8, 'Radiografia ao Torax', 49.99).
cuidado(data(12, 12, 2017), 2, 9, 'Sessao de Fisioterapia', 69.99).
cuidado(data(12, 12, 2017), 3, 1, 'Consulta Neurologia', 19.99).
cuidado(data(13, 11, 2017), 3, 0, 'Consulta Cardiologia', 9.99).
cuidado(data(14, 12, 2017), 4, 5, 'Consulta de Exames', 0).
```

```
cuidado(data(15, 12, 2017), 4, 4, 'Consulta Psiquiatria', 15.99).
cuidado(data(15, 11, 2017), 4, 4, 'Consulta Psiquiatria', 14.99).
cuidado(data(15, 12, 2017), 4, 2, 'Consulta Urologia', 19.99).
```

3.1.4 Instituição

Uma instituição é representada através de um número identificador que deve ser único para cada instituição, por um nome e pela cidade onde está localizada. Os seguintes exemplos representam instituições:

```
instituicao(1, 'Hospital de Braga', 'Braga').
instituicao(2, 'Hospital de Guimaraes', 'Guimaraes').
instituicao(3, 'Hospital do Porto', 'Porto').
instituicao(4, 'Hospital de Lisboa', 'Lisboa').
```

3.1.5 Consulta

Finalmente, uma consulta é definida pelos números identificadores que caracterizam um utente e um prestador de serviços envolvidos numa consulta. Uma consulta é também definida por uma hora de início e por uma hora de fim (que pode também ser vista como uma estimativa de uma duração de uma consulta no que toca a poder marcar consultas futuras com um prestador). Tanto a hora de início, como a hora de fim, seguem o formato (DD, MM, AAAA, hh, mm), em que "hh" representa uma hora e "mm" os minutos. Os seguintes predicados representam consultas:

```
consulta(0, 0, data_hora(25, 1, 2018, 14, 20),
    data_hora(25, 1, 2018, 14, 50)).
consulta(1, 0, data_hora(25, 1, 2018, 15, 0),
    data_hora(25, 1, 2018, 15, 30)).
consulta(2, 0, data_hora(25, 1, 2018, 15, 40),
    data_hora(25, 1, 2018, 16, 10)).
consulta(3, 0, data_hora(25, 1, 2018, 16, 20),
    data_hora(25, 1, 2018, 16, 50)).
consulta(4, 0, data_hora(25, 1, 2018, 17, 20),
    data_hora(25, 1, 2018, 18, 0)).
consulta(0, 1, data_hora(25, 1, 2018, 14, 20),
    data_hora(25, 1, 2018, 14, 50)).
consulta(1, 1, data_hora(25, 1, 2018, 15, 0),
    data_hora(25, 1, 2018, 15, 30)).
```

3.2 Adição e remoção de conhecimento

A inserção e remoção de conhecimento é realizada à custa de invariantes que especificam um conjunto de restrições que devem ser verdadeiras após uma inserção/remoção de conhecimento. A inserção de predicados pode ser vista como uma evolução do sistema em termos de conhecimento. De um modo análogo, uma remoção de um predicado pode ser vista como uma involução do sistema. De notar que, tanto as operações de inserção como de remoção são sempre efetuadas. No entado, após cada uma dessas operações é verificada a consistência do sistema com recurso aos invariantes. Caso alguma

operação provoque uma anomalia no sistema, esta perde o seu efeito e o sistema volta ao estado anterior à operação em questão (semelhante à operação de *rollback* numa base de dados).

Antes mesmo de se descrever, com detalhe, os processos de evolução e de involução do sistema, assim como todos os invariantes desenvolvidos, faz sentido descrever dois predicados que são absolutamente cruciais para o correto funcionamento destes processos. Um destes predicados denomina-se *solucoes* e é caracterizado da seguinte forma:

```
solucoes(F, Q, S) :- findall(F, Q, S).
```

Em que o predicado *findall* é um predicado nativo da linguagem *PROLOG*. Este predicado procura todas as ocorrências de "F" que satisfaçam "Q". Por sua vez estes resultados são guardados numa lista "S".

Por sua vez, será necessário para os processos de evolução e involução so sistema, determinar o tamanho da lista resultante após a execução do predicado *solucoes*. Este processo é descrito pelo predicado *comprimento* que é definido da seguinte forma:

```
comprimento(S, N) :- length(S, N).
```

Em que *length* é um predicado nativo da linguagem *PROLOG* que dada uma lista "S" determina o seu comprimento em "N".

Através destes dois predicados fundamentais, pode-se então descrever os processos de evolução e involução do sistema, assim como os invariantes desenvolvidos para cada um dos predicados apresentados na secção **3.1**.

3.2.1 Evolução do sistema

O processo de evolução do sistema é caracterizado por um aumento de conhecimento no sistema. Este processo é descrito por:

```
evolucao(Termo) :-
                solucoes(Inv, +Termo::Inv, S),
                insere(Termo),
                teste(S).
```

Em que o predicado *insere* caracteriza-se pela inserção de conhecimento no sistema (à custa do predicado *assert*) e o predicado *teste* pelo teste aos invariantes relativos a esse "Termo" armazenados na lista "S". De notar que este processo insere primeiro o conhecimento e só apenas depois é que realiza testes à custa dos invariantes, pelo que estamos perante um teste à consistência do sistema. Através desta abordagem de representação do sistema, na inserção de conhecimento do mesmo deve ser usado o predicado *evolucao* e não o predicado *assert* já fornecido pelo *PROLOG*. Os predicados *insere* e *teste* são ilustrados em anexo. De notar que, por conveniência, definiu-se que um invariante relativo a uma inserção de um termo qualquer é precedido pelo símbolo de "+" no nome desse mesmo termo.

3.2.2 Involução do sistema

O processo de involução do sistema é muito semelhante ao processo de evolução e é definido do seguinte modo:

```
involucao(Termo) :-
          Termo,
          solucoes(Inv, -Termo::Inv, S),
```

```
        remove(Termo),
        teste(S).
```

Sendo que neste caso é efetuado um processo de remoção (o predicado *remove* está definido em anexo). Definiu-se que um invariante relativo a uma remoção de um termo qualquer é precedido pelo símbolo de "-" no nome desse mesmo termo. De notar que, contrariamente ao processo de evolução, efetua-se inicialmente um teste ao próprio termo. Isto porque não faz sentido a remoção de conhecimento que não exista.

3.2.3 Utentes

A inserção de conhecimento relativa a um utente deve respeitar os seguintes critérios: o identificador de um utente deve ser inteiro e único e a idade deve possuir um valor inteiro pertencente ao intervalo $[0, 125]$. O invariante que traduz estas restrições é o seguinte:

```
+utente(Id, _, Idade, _) :: (
    integer(Id),
    integer(Idade),
    Idade >= 0,
    Idade =< 125,
    solucoes(Id, utente(Id, _, _, _), S),
    comprimento(S, N),
    N == 1
).
```

Por sua vez, a remoção de um utente não deve ser permitida quando ainda existirem cuidados prestados ou consultas relativas ou mesmo:

```
-utente(IdUt, _, _, _) :: (
    solucoes(IdUt, cuidado(_, IdUt, _, _, _), S1),
    solucoes(IdUt, consulta(IdUt, _, _, _), S2),
    comprimento(S1, N1),
    comprimento(S2, N2),
    N1 == 0, N2 == 0
).
```

3.2.4 Prestadores

A inserção de um prestador só deve ser permitida se não existir conhecimento de um prestador com o mesmo número de identificação, se o número de indentificação for inteiro e se existir previamente o conhecimento de uma instituição que contenha o mesmo número de indentificação da instituição fornecida pelo prestador, ou seja:

```
+prestador(Id, _, _, IdIns) :: (
    integer(Id),
    solucoes(Id, prestador(Id, _, _, _), S1),
    solucoes(IdIns, instituicao(IdIns, _, _), S2),
    comprimento(S1, N1),
    comprimento(S2, N2),
    N1 == 1,
```

```
    N2 == 1
).
```

No processo de remoção de um prestador, tal como para um utente, não é permitida a remoção de prestador ao qual ainda exista conhecimento de cuidados prestados ou de consultas que envolvem o mesmo:

```
-prestador(IdPrest, _, _, _) :: (
    solucoes(IdUt, cuidado(_, _, IdPrest, _, _), S1),
    solucoes(IdPrest, consulta(_, IdPrest, _, _), S2),
    comprimento(S1, N1),
    comprimento(S2, N2),
    N1 == 0, N2 == 0
).
```

3.2.5 Cuidados prestados

Relativamente a um cuidado prestado, é considerado a inserção de um conhecimento duplicado quando todos os seus argumentos são completamente iguais a um outro cuidado prestado já presente no sistema. Além disso, o primeiro argumento de um predicado deve ser uma data válida, o segundo argumento deve corresponder a um indentificador de um utente já presente no sistema e o terceiro argumento a identificador de um prestador já presente no sistema. Finalmente, o custo de um cuidado deve um número positivo:

```
+cuidado(Data, IdUt, IdPrest, Descr, Custo) :: (
    number(Custo), Custo >= 0, data(Data),
    solucoes(IdUt, utente(IdUt, _, _, _), S1),
    solucoes(IdPrest, prestador(IdPrest, _, _, _), S2),
    solucoes((Data, IdUt, IdPrest, Descricao, Custo),
             cuidado(Data, IdUt, IdPrest, Descricao, Custo),
             S3),
    comprimento(S1, N1),
    comprimento(S2, N2),
    comprimento(S3, N3),
    N1 == 1, N2 == 1, N3 == 1
).
```

O predicado *data* determina se uma data é correta ou não e está definido em anexo.

Após a remoção de um cuidado deve ser garantido que não haja conhecimento do mesmo predicado, ou seja:

```
-cuidado(Data, IdUt, IdPrest, Descr, Custo) :: (
    solucoes((Data, IdUt, IdPrest, Descricao, Custo),
             cuidado(Data, IdUt, IdPrest, Descricao, Custo),
             S),
    comprimento(S, N),
    N == 0
).
```

3.2.6 Instituições

Na inserção de uma instituição apenas se deve garantir que o indeficador de uma institituição não exista e que o seu valor é inteiro:

```
+instituicao(Id, _, _) :: (
    integer(Id),
    solucoes(Id, instituicao(Id, _, _), S),
    comprimento(S, N),
    N == 1
).
```

Por sua vez, não se deve remover uma instituição quando há conhecimentos de prestadores associados à mesma:

```
+instituicao(Id, _, _) :: (
    solucoes(Id, prestador(_, _, _, Id), S),
    comprimento(S, N),
    N == 0
).
```

3.2.7 Consultas

A inserção de uma consulta consiste no invariante mais complexo desenvolvido no sistema. Além de não ser permitida a inserção de uma consulta repetida (com todos os seus argumentos iguais a uma outra consulta já existente), os seus primeiros dois argumentos devem corresponder, respetivamente, a um utente e a um prestador que já existam. Além disso o dois últimos argumentos devem ser horas válidas. Para a determinação deste último critério foi desenvolvido um predicado *data_hora* que à custa do predicado *data* determina a validade de uma hora. Este predicado está exemplificado em anexo. O invariante que caracteriza estas restrições é o seguinte:

```
+consulta(IdU, IdP, HI, HF) :: (
    data_hora(HI), data_hora(HF),
    solucoes(IdU, utente(IdU, _, _, _), S1),
    solucoes(IdP, prestador(IdP, _, _, _), S2),
    solucoes((IdU, IdP, HI, HF), consulta(IdU, IdP, HI, HF), S3),
    solucoes((IdU1, IdP, HI1, HF1), consulta(IdU1, IdP, HI1, HF1), S4),
    comprimento(S1, N1),
    comprimento(S2, N2),
    comprimento(S3, N3),
    N1 == 1, N2 == 1, N3 == 1,
    remove_consulta(S4, (IdU, IdP, HI, HF), L),
    nao_colide(HI, HF, L)
).
```

De notar nas duas últimas restrições definidas pelo invariante à custa dos predicados *remove_consulta* e *nao_colide*. Uma outra característica relativa a uma consulta que se considerou foi o facto de não se permitir adicionar uma consulta com um prestador que colida com a hora de outra consulta já marcada com o mesmo. Neste caso o predicado *nao_colide* seria, por si só, suficiente para o teste desta restrição. Este predicado recebe duas horas (uma de início e outra de fim) e uma lista que contém pares

de horas de início e de fim de outras consultas. Através desta informação testa se as duas primeiras horas recebidas colidem com algum dos pares presentes nessa lista. O predicado é o seguinte:

```
nao_colide(D1, D2, []) :- depois(D2, D1).
nao_colide(D1, D2, [(D3, D4)|T]) :-
    depois(D2, D1),
    depois(D4, D3),
    depois(D1, D4),
    nao_colide(D1, D2, T).
nao_colide(D1, D2, [(D3, D4)|T]) :-
    depois(D2, D1),
    depois(D4, D3),
    depois(D3, D2),
    nao_colide(D1, D2, T).
```

Em que o predicado *depois* recebe duas horas e testa se a primeira é posterior à segunda. Este predicado encontra-se definido em anexo.

No entanto, relembrando a evolução do sistema, são efetuados testes à consistência do sistema e não testes de verificação antes de uma inserção. Ou seja, a nova consulta que se prentende inserir já está presente no sistema antes do teste deste invariante pelo que o teste de colisão das horas de fim e de início irá sempre falhar visto que já estão previamente inseridas na lista de horas de consultas de um determinado prestador. Para resolver esta situação, desenvolveu-se o predicado *remove_consulta* que dada uma lista de consultas remove a que é exatamente igual à consulta que se pretente inserir. Além disso a lista resultante é filtrada de modo a que apenas contenha os horários de todas as consultas e não toda a informação destas. O predicado é o seguinte:

```
remove_consulta([], _, []).
remove_consulta([(U, P, I, F)|T], (U, P, I, F), L) :-
    remove_consulta(T, (U, P, I, F), L).
remove_consulta([(U1, P1, I1, F1)|T], (U2, P2, I2, F2),
    [(I1, F1)|L]) :-
    remove_consulta(T, (U2, P2, I2, F2), L).
```

Aplicando o predicado *nao_colide* ao resultado deste predicado obtem-se o resultado esperado da inserção de uma nova consulta.

Relativamente à remoção de uma consulta deve-se apenas garantir que não existe conhecimento da consulta que se pretende remover:

```
-consulta(IdU, IdP, HI, HF) :: (
    solucoes((IdU, IdP, HI, HF), consulta(IdU, IdP, HI, HF), S),
    comprimento(S, N),
    N == 0
).
```

3.3 Implementação dos predicados de consulta e análise de resultados

Após a criação dos predicados que permitem a evolução e a involução de conhecimento do sistema, assim como o desenvolvimento dos demais invariantes, prodeceu-se então à crição de predicados de consulta sobre a estrutura de conhecimento. Estes predicados foram maioritariamente desenvolvidos à

custa do predicado *solucoes*, sendo que em alguns casos recorreu-se a alguns predicados auxiliares. As próximas secções descrevem com detalhe os diferentes predicados de consulta desenvolvidos, ilustrando para cada um deles um exemplo de um resultado de uma consulta através dos mesmos.

3.3.1 Procura de utentes

Esta consulta permite obter utentes pelos seguintes critérios: identificador, nome, idade e/ou morada). Os resultados são devolvidos numa lista. Quantos mais critérios forem fornecidos mais específica se torna a consulta e, por sua vez, menor será a quantidade de resultados que a satisfazem. O predicado que traduz esta consulta é o seguinte:

```
procura_utentes(Id, N, Idade, M, L) :-
    solucoes((Id, N, Idade, M), utente(Id, N, Idade, M), L).
```

Em que "Id" corresponde ao identificador de um utente, "N" ao nome de um utente, "M" à morada de um utente e "L" ao resultado da consulta.

Como exemplo, e considerando os exemplos de utentes fornecidos na secção **3.1** considere-se o resultado da seguinte consulta:

```
| ?- procura_utentes(Id, Nome, Idade, 'Rua da Poeira', L).
L = [(4,'Sebastiao',83,'Rua da Poeira'),(6,'Jorge',44,'Rua da Poeira'),(7,'Rafaela',23,'Rua da Poeira')]
```

Figure 1: Consulta de utentes por morada

3.3.2 Procura de instituiçoes

Face à abordagem utilizada para a definição do conhecimento sobre uma instituição, a obtenção de todas as instituições torna-se bastante simples:

```
instituicoes(L) :-
    solucoes((Id, Nome, Cidade), instituicao(Id, Nome, Cidade), L).
```

Em que "L" representa a lista de todas as instituições envolvidas. A figura seguinte ilustra um resultado possível desta mesma consulta:

```
| ?- instituicoes(L).
L = [(1,'Hospital de Braga','Braga'),(2,'Hospital de Guimaraes','Guimaraes'),(3,'Hospital do Porto','Porto
'),(4,'Hospital de Lisboa','Lisboa')] ?
```

Figure 2: Consulta de todas as instituições

3.3.3 Procura de utentes relacionados com um prestador, especialidade ou instituição

Esta consulta exige que se pesquise por utentes que estejam envolvidos em cuidados médicos prestados por um qualquer prestador ou mesmo apenas numa determinada especialidade ou instituição:

```
procura_utentes(IdPres, Esp, Ins, L) :-
    solucoes((IdU, Nome, Idade, Morada),
             (prestador(IdPres, _, Esp, Ins),
              cuidado(_, IdU, IdPres, _, _),
              utente(IdU, Nome, Idade, Morada)),
             S),
    sem_repetidos(S, L).
```

Em que "IdPres" representa o identificador de um determindado prestador, "Esp" uma especialidade, "Ins" o identificador de uma instituição e "L" a lista de resultados desta mesma consulta. O predicado *sem_repetidos* foi criado com o intuito de remover elementos repetidos de uma lista e está ilustrado em anexo. Esta operação foi necessária de já que diferentes prestadores podem efetuar cuidados médicos aos mesmos utentes. Assim, a lista resultante deixa de produzir resultados duplicados.

```
| ?- procura_utentes(IdP, Esp, 1, L).
L = [(1,'Maria',41,'Rua de Cima'),(2,'Gertrudes',26,'Rua Carlos Antonio'),(3,'Paula',73,'Rua da Mina'),(5,
'Zeca',9,'Rua do Poeira'),(6,'Jorge',44,'Rua da Poeira'),(10,'Zueiro',33,'Rua do Mar'),(8,'Anabela',42,'Ru
a de Baixo')] ?
```

Figure 3: Procura de utentes na instituição com o identificador "1"

A figura anterior ilustra o resultado da consulta em questão sobre a instituição com o número de identifição 1 (Hospital de Braga). Podemos comparar estes resultados através dos factos apresentados na secção **3.1** em que facilmente se verifica que os utentes que consultaram prestadores do Hospital de Braga estão presentes na lista de resultados. (Nota: por uma questão de legibilidade deste mesmo relatório optou-se por não colocar todos os cuidados médicos nem todas as consultas).

3.3.4 Procura de cuidados por data, instituição ou cidade

Como se trata de procurar por cuidados médicos associados a uma determinada data, intituição e/ou cidade, foi necessário para esta consulta obter soluções a partir de prestadores, instituições e cuidados:

```
procura_cuidados_por_dic(Data, Inst, Cidade, L) :-
    solucoes((Data, IdU, IdPres, Desc, Custo),
             (prestador(IdPres, _, _, IdInst),
                        instituicao(IdInst, Inst, Cidade),
              cuidado(Data, IdU, IdPres, Desc, Custo)),
             L).
```

Em que "Data" representa a data de um cuidado, "Inst" o nome de uma instituição onde forem prestados cuidados e "L" a lista de resultados.

```
| ?- procura_cuidados_por_dic(Data, Ints, 'Guimaraes', L).
L = [(data(13,11,2017),0,1,'Consulta Neurologia',39.99),(data(12,12,2017),3,1,'Consulta Neurologia',19.99)
,(data(7,12,2017),7,1,'Consulta Neurologia',89.99),(data(23,11,2017),10,1,'Consulta de Exames',5.99),(data
(15,12,2017),4,4,'Consulta Psiquiatria',15.99),(data(15,11,2017),4,4,'Consulta Psiquiatria',14.99),(data(1
2,12,2017),2,9,...,...)] ?
```

Figure 4: Procura de cuidados efetuados na cidade de Guimarães

3.3.5 Procura de cuidados por utente, prestador ou instituição

Como se trata de procurar por cuidados médicos associados a um determinado utente, prestador e/ou intituição, foi necessário para esta consulta obter soluções a partir de utentes, prestadores e instituições:

```
procura_cuidados_por_upi(IdU, IdPres, Inst, L) :-
    solucoes((Data, IdU, IdPres, Desc, Custo),
             (prestador(IdPres, _, _, IdInst),
                        instituicao(IdInst, Inst, _),
              cuidado(Data, IdU, IdPres, Desc, Custo)),
             L).
```

Em que "IdU" representa o identificador de um utente, "IdPres" o identificador de um prestador, "Inst" o nome de uma instituição onde forem prestados cuidados e "L" a lista de resultados.

```
| ?- procura_cuidados_por_upi(1, P, I, L).
L = [(data(11,12,2017),1,0,'Consulta Cardiologia',59.99),(data(10,11,2017),1,6,'Consulta Ginecologia',19.9
9),(data(10,11,2017),1,6,'Consulta Ginecologia',59.99)] ?
```

Figure 5: Procura de cuidados prestados ao utente com o identificador "1"

3.3.6 Prestadores de um utente

Para a obtenção de todos os prestadores de um utente devem-se cruzar os conhecimentos envolvidos nos cuidados prestados a um utente e seus respetivos prestadores:

```
prestadores_de_utente(IdU, L) :-
    solucoes((IdPres, Nome, Esp, Inst),
             (prestador(IdPres, Nome, Esp, Inst),
              cuidado(_, IdU, IdPres, _, _)),
             S),
    sem_repetidos(S, L).
```

Em que "IdU" representa o identificador de um utente e "L" a lista de resultados. É necessário usar o predicado *sem_repetidos* já que o mesmo utente pode receber vários cuidados do mesmo prestador.

```
| ?- prestadores_de_utente(1, L).
L = [(0,'Manuel','Cardiologia',1),(6,'Manuela','Ginecologia',4)] ?
```

Figure 6: Procura dos prestadores do utente com o identificador "1"

3.3.7 Instituições de um utente

Para obter todas as instituições onde um utente requereu cuidados médicos é necessário cruzar a informação entre os prestadores, cuidados e instituições associadas a um utente. Já que um utente pode atender a uma instituição várias vezes é necessário eliminar resultados duplicados.

```
instituicoes_de_utente(IdU, L) :-
    solucoes((Inst, Nome, Cidade),
             (prestador(IdPres, _, _, Inst),
              cuidado(_, IdU, IdPres, _, _),
                      instituicao(Inst, Nome, Cidade)),
             R),
    sem_repetidos(R, L).
```

Em que "IdU" representa o identificador de um utente e "L" a lista de resultados.

```
| ?- instituicoes_de_utente(1, L).
L = [(1,'Hospital de Braga','Braga'),(4,'Hospital de Lisboa','Lisboa')] ?
```

Figure 7: Instituições associadas ao utente com o identificador "1"

3.3.8 Obtenção do custo total por critérios de seleção

Os custos totais podem ser obtidos através dos seguintes critérios: utente, prestador, especialidade e data de um cuidado. Assim sendo é necessário cruzar informação relativa a cuidados e prestadores.

```
custo_total(IdU, IdP, Esp, Data, C) :-
    solucoes(Custo,
             (prestador(IdP, _, Esp, _),
              cuidado(Data, IdU, IdP, _, Custo)),
             R),
    sum(R, C).
```

Em que "IdU" representa o identificador de um utente, "IdP" o identificador de um prestador, "Esp" uma especialidade e "C" o resultado do custo total. O predicado *solucoes* devolve uma lista com todos os custos envolvidos nos diferentes cuidados prestados pelo foi desenvolvido o predicado *sum* que efetua o somatório de uma lista. Este predicado encontra-se definido em anexo.

```
| ?- custo_total(U, P, 'Cardiologia', Data, C).
C = 179.94 ?
```

Figure 8: Custo total de cuidados relativos a cardiologia

3.3.9 Procura de consultas por critérios de seleção

Uma consulta pode ser procurada através dos seguintes critérios: utente, prestador, hora de início e/ou hora de fim.

```
procura_consultas(IdU, IdP, HI, HF, L) :-
    solucoes((IdU, IdP, HI, HF), consulta(IdU, IdP, HI, HF), L).
```

Em que "IdU" representa o identificador de um utente, "IdP" o identificador de um prestador, "HI" a hora de início de uma consulta, "HF" a hora de fim de uma consulta e "L" a lista de resultados.

```
| ?- procura_consultas(U, 0, HI, HF, L).
L = [(0,0,data_hora(25,1,2018,14,20),data_hora(25,1,2018,14,50)),(1,0,data_hora(25,1,2018,15,0),data_hora(
25,1,2018,15,30)),(2,0,data_hora(25,1,2018,15,40),data_hora(25,1,2018,16,10)),(3,0,data_hora(25,1,2018,16,
20),data_hora(25,1,2018,16,50)),(4,0,data_hora(25,1,2018,17,20),data_hora(25,1,2018,18,0))] ?
```

Figure 9: Consultas associadas ao prestador com o identificador "0"

3.3.10 Procura de consultas que ocorram numa determinada data

Considerou-se que seria também proveitoso pesquisar por consultas que ocorrem num determinado dia. O predicado responsável por este tipo de pesquisa é o seguinte:

```
procura_consultas(data(D, M, A), L) :-
    solucoes((IdU, IdP, data_hora(D, M, A, H1, M1),
             data_hora(D, M, A, H2, M2)),
             consulta(IdU, IdP, data_hora(D, M, A, H1, M1),
                 data_hora(D, M, A, H2, M2)),
             L).
```

Em que "data(D, M, A)" representa um dia e "L" a lista de resultados.

```
| ?- procura_consultas(data(25,1,2018), L).
L = [(0,0,data_hora(25,1,2018,14,20),data_hora(25,1,2018,14,50)),(1,0,data_hora(25,1,2018,15,0),data_hora(
25,1,2018,15,30)),(2,0,data_hora(25,1,2018,15,40),data_hora(25,1,2018,16,10)),(3,0,data_hora(25,1,2018,16,
20),data_hora(25,1,2018,16,50)),(4,0,data_hora(25,1,2018,17,20),data_hora(25,1,2018,18,0)),(0,1,data_hora(
25,1,2018,14,20),data_hora(25,1,2018,14,50)),(1,1,data_hora(...),data_hora(...)),(2,1,.....),(3,.....)
,(.....)|...] ?
```

Figure 10: Consultas que ocorrem no dia 25-1-2018

Facilmente se verifica que estes resultados são os esperados já que todas as consultas definidas na secção **3.1** ocorreram no dia pesquisado.

4 Conclusões e Sugestões

Após o término deste exercício prático, congratulámo-nos por termos conseguido implementar as funcionalidades básicas exigidas e as demais acrescidas por iniciativa do grupo.

Fomos capazes de aplicar os conhecimentos lecionados durante as aulas da unidade curricular e reaproveitar exemplos e frações de código apresentadas nas lições de cariz prático, que considerámos úteis.

Pudemos, com êxito, consolidar e exercitar os nossos conhecimentos em *PROLOG* e obter competências um pouco mais robustas quanto a programação em lógica, atingindo, portanto, o objetivo deste trabalho.

Como sugestão para trabalho futuro – e em virtude da ampla diversidade de serviços relacionados com a área médica –, poderiam ser implementadas novas funcionalidades como, por exemplo, a inserção de receitas médicas, gerenciamento de pessoal de enfermagem, etc.

5 Anexos

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


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


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


data(D, M, A) :-
    A >= 0,
    pertence(M, [1,3,5,7,8,10,12]),
    D >= 1,
    D =< 31.
data(D, M, A) :-
    A >= 0,
    pertence(M, [4,6,9,11]),
    D >= 1,
    D =< 30.
data(D, 2, A) :-
    A >= 0,
    A mod 4 =\= 0,
    D >= 1,
    D =< 28.
data(D, 2, A) :-
    A >= 0,
    A mod 4 =:= 0,
    D >= 1,
    D =< 29.
data(data(D, M, A)) :- data(D, M, A).
```

A terceira clausula do predicado *data* trata da exceção do mês de Fevereiro em que um ano bissexto tem 366 dias por oposição aos outros anos que possuem 365 dias. A última clausula permite que o teste de validade de uma data. De notar que "D" corresponde ao dia, "M" ao mês e "A" ao ano de uma data.

```
data_hora(D, M, A, H, Min) :-
    data(D, M, A),
    H >= 0,
    H =< 23,
    Min >= 0,
    Min =< 59.
data_hora(data_hora(D, M, A, H, Min)) :- data_hora(D, M, A, H, Min).
```

O predicado *data_hora* representa uma hora no formato (Dia, Mês, Ano, Hora, Minuto) e é definido à custa do predicado *data*. Novamente, a última clausula deste predicado determina a validade de uma determinada hora.

```
depois(data_hora(_, _, A1, _, _), data_hora(_, _, A2, _, _)) :-
    A1 > A2.
depois(data_hora(_, M1, A1, _, _), data_hora(_, M2, A2, _, _)) :-
    A1 >= A2,
    M1 > M2.
depois(data_hora(D1, M1, A1, _, _), data_hora(D2, M2, A2, _, _)) :-
    A1 >= A2,
    M1 >= M2,
    D1 > D2.
depois(data_hora(D1, M1, A1, H1, _), data_hora(D2, M2, A2, H2, _)) :-
    A1 >= A2,
    M1 >= M2,
    D1 >= D2,
    H1 > H2.
depois(data_hora(D1, M1, A1, H1, Min1),
    data_hora(D2, M2, A2, H2, Min2)) :-
    A1 >= A2,
    M1 >= M2,
    D1 >= D2,
    H1 >= H2,
    Min1 > Min2.


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

O predicado *pertence* determina se um dado elemento "X" pertence a uma lista.

```
sem_repetidos([],[]).
sem_repetidos([H|T], R) :-
    pertence(H,T),
    sem_repetidos(T,R).
sem_repetidos([H|T], [H|R]) :-
    nao(pertence(H,T)),
    sem_repetidos(T,R).
```

Este predicado retira elementos repetidos de uma lista. Para o efeito, usa o predicado *pertence* para determinar se um elemento é repetido.

```
sum([], 0).
sum([H|T], R) :- sum(T, L), R is H + L.
```