Sistemas de Representação de Conhecimento e Raciocínio (3º ano de MIEI)

Primeiro Exercício

Relatório de Desenvolvimento

André Gonçalves (a80368) | João Queirós (a82422) | Luís Alves (a80165) | Rafaela Rodrigues (a80516)

31 de Março de 2019

Resumo

Este relatório inicia-se com uma breve introdução, seguida de um conjunto de preliminares que definem o universo sobre o qual o trabalho prático foi desenvolvido. De seguida, são apresentados alguns predicados auxiliares que foram elaborados para ajudar a definição dos predicados principais, que são explicitados de seguida. No fim de cada secção relativa a uma questão do enunciado do trabalho prático, encontram-se demonstrações práticas do uso dos predicados desenvolvidos. Por fim, são apresentadas algumas conclusões sobre a elaboração do caso de estudo abordado.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

Este trabalho foi desenvolvido no âmbito da Unidade Curricular de Sistemas de Representação de Conhecimento e Raciocínio, que se insere no 2º semestre do 3º ano do Mestrado Integrado em Engenharia Informática.

É o primeiro de 3 exercícios propostos que constituem o trabalho prático da UC.

Para este primeiro exercício foi proposto o desenvolvimento de um sistema de representação de conhecimento e raciocínio com capacidade para caracterizar um universo de discurso na área da prestação de cuidados de saúde.

Com este trabalho prático pretende-se o desenvolvimento das capacidades do grupo na utilização da linguagem de programação PROLOG para implementar mecanismos de programação em lógica.

Capítulo 2

Preliminares

2.1 Panorama

O sistema de conhecimento a desenvolver terá por base de conhecimento os seguintes predicados:

- `utente: IdUtente, Nome, Idade, Cidade -> {V,F}`
- `servico: IdServico, Descricao, Instituicao, Cidade -> {V,F}`
- `consulta: Data, IdUtente, IdServico, Custo -> {V,F}`
- `data: Dia, Mes, Ano -> {V,F}`
- `sexo: Sexo, IdUtente -> {V,F}`

2.1.1 Utente

Um Utente terá um ID único, um nome, uma idade (que estará entre 1 e 134, inclusive), e habitará numa cidade. Um Utente poderá efetuar consultas de um dado serviço numa determinada, tendo essa consulta um determinado custo. Assume-se que o mesmo Utente não realizará mais que uma consulta com a mesma descrição na mesma data.

2.1.2 Serviço

Um Serviço terá um ID único, uma descrição do mesmo e será disponibilizado numa instituição de uma dada cidade.
Para além disso, a mesma instituição não poderá disponibilizar mais que um serviço com a mesma descrição.

2.1.3 Consulta

Uma Consulta ocorrerá numa determinada data, sendo efetuada a um Utente com um determinado ID e sendo de um Serviço. Para além disso, tem também associado um Custo, que será superior a 0.

2.1.4 Data

Uma Data é constituída por um dia, mês e ano, sendo que não são tidos em consideração os anos bissextos. Representam-se estes 3 componentes através de números inteiros.

2.1.5 Sexo

Um Sexo poderá ser Masculino, Feminino ou Outro, tendo associado um determinado ID de Utente. O Sexo será representado por 'M', 'F' ou 'O'.

Capítulo 3

Descrição do Trabalho e Análise de Resultados

3.1 Predicados auxiliares

Uma vez que ao longo do trabalho desenvolvido será necessário operar sobre listas, foram definidos alguns predicados que permitem obter informação útil das listas. Enumeram-se de seguida:

1. `pertence: Elemento, Lista -> {V,F}`
2. `semRepetidos: ListaComRepetidos, ListaSemRepetidos -> {V,F}`
3. `countElements: Lista, Tamanho da Lista -> {V,F}`
4. `sumElements: Lista, Soma -> {V,F}`

O primeiro, dada um elemento, indica se ele pertence à lista. O segundo, dada uma lista com repetidos, determina a lista sem repetições de elementos. O terceiro, dada uma lista, determina o tamanho desta. O quarto e último, dada uma lista, determina a soma dos seus elementos.

3.2 Registo e remoção de utentes, serviços e consultas

De forma a poderem serem adicionados novos utentes, serviços e consultas, foi desenvolvido um conjunto de predicados auxiliares que permitem a evolução e involução do conhecimento. Estes apresentam-se na subsecção 3.2.1.
Assim, foi possível desenvolver um conjunto de invariantes que verificam as propriedades enumeradas nas secções 2 para cada um dos componentes.

3.2.1 Evolução e Involução

De forma a podermos evoluir a nossa base de conhecimento dinamicamente, foi necessário que desenvolvêssemos dois predicados basilares: `evolucao` e `involucao`. O primeiro representa a adição de novo conhecimento à base de conhecimento e o segundo a subtração de conhecimento.

Antes de os definirmos, também definimos o predicado `solucoes: Formato, Prova, Lista de Soluções -> {V,F}`. Este predicado foi construído à custa da definição do *SICStus PROLOG* de `findall`.

Tendo o predicado `solucoes` definido, passamos à definição de 2 novos predicados: `insercao` e `remocao`. O primeiro é definido em 3.1. O segundo em 3.2.

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

Figura 3.1: Predicado de Inserção

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

Figura 3.2: Predicado de Remoção

Ambos os predicados fazem uso dos mecanismos do *PROLOG* de procura de soluções. Pegando no caso da inserção, se for possível realizar um *assert* de um determinado termo, então foi possível inseri-lo. No entanto, devido ao mecanismo de *backtracking* do *PROLOG*, a segunda cláusula será utilizada caso a cláusula seguinte da `evolucao` falhe. No entanto, a primeira cláusula é sempre verificada, pelo que caso nos encontremos na segunda, é porque não foi possível inserir o termo por violar algum invariante. Por isso, deverá ser removido o termo adicionado (com recurso ao *retract*), sendo que de seguida é feito um *cut* de forma a que o valor lógico da expressão seja falso independentemente do mecanismo de *backtracking*, e assim a evolução do conhecimento seja falsa também. Este raciocínio é igualmente aplicável para a `remocao`.

Como mencionado, a segunda cláusula de `remocao` e de `insercao` apenas deve ser utilizada pelo *PROLOG* caso a inserção ou remoção viole algum invariante. Por isso, será necessária a definição de um predicado que teste uma lista de termos. Foi definido então o predicado em 3.3.

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

Figura 3.3: Predicado de Teste

Tendo estes 3 predicados defindos, passa então a ser possível definir os predicados `evolucao` e `involucao` em 3.4 e 3.5.

```
evolucao: Termo -> {V,F}
evolucao( T ) :- solucoes( I, +T::I, LInv),
         insercao(T),
         teste(LInv).
```

Figura 3.4: Predicado de Evolução do Conhecimento

```
involucao: Termo -> {V,F}
involucao( T ) :- T,
        solucoes(I, -T::I, LInv),
        remocao(T),
        teste(LInv).
```

Figura 3.5: Predicado de Involução do Conhecimento

3.2.2 Registo/Remoção de Utente

Um Utente apenas poderá ser registado se não existir um Utente com o mesmo ID, se o sexo for válido e se a idade for um número inteiro inferior a 135.
Por isso, foi desenvolvido um invariante estrutural e referencial para o predicado utente. O primeiro não permite a inserção de conhecimento repetido sobre um Utente, e o segundo valida a idade no momento de registo. Como exemplo, o invariante referencial encontra-se representado em 3.6.

```
+utente(_,_,Id,_) :: (natural(Id), Id < 135).
```

Figura 3.6: Invariante Referencial: Não permitir a inserção de utentes com idades não válidas

Para o predicado sexo, foi também desenvolvido um invariante estrutural e um referencial. O primeiro não permite a inserção de conhecimento repetido sobre o sexo de um Utente, e o segundo valida o Sexo (para ver quais os valores válidos para Sexo, ver 2.1.5). Como exemplo, o invariante estrutural encontra-se representado em 3.7.

```
+sexo(Sexo,Id) :: (solucoes(Ls, sexo(_,Id),S),
        countElements(S,N),
        N == 1).
```

Figura 3.7: Invariante Estrutural: Não permitir a inserção de sexo para o mesmo Utente

Assim, para registar um utente foi criado o predicado 3.8.

```
registarU: IdUtente, NomeUtente, Idade, Cidade, Sexo -> {V,F}
registarU(Id, Nome, Idade, Cidade, Sexo) :-
evolucao(utente(Id,Nome,Idade,Cidade)), evolucao(sexo(Sexo,Id)).
```

Figura 3.8: Predicado de Registar Utente

Relativamente à remoção de um Utente, apenas poderá ser efetuada caso não existam consultas associadas a ele e caso exista registo do seu sexo.
Por isso, foi desenvolvido um invariante referencial, que verifica a existência de consultas com o ID do Utente que se pretende remover.
Assim, para remover um Utente foi criado o predicado 3.9.

```
apagarU: IdUtente -> {V,F}
apagarU(ID) :- involucao(utente(ID,_,_,_)),
    involucao(sexo(_,ID)).
```

Figura 3.9: Predicado de Remover Utente

3.2.3 Registo/Remoção de Serviço

Um Serviço poderá ser registado se não existir um Serviço com o mesmo ID e se não existir um serviço com uma descrição igual na mesma instituição.
Por isso, foram definidos dois invariantes estruturais para o predicado serviço. O primeiro não permite a inserção de serviços com o mesmo ID e o segundo não permite a inserção de serviços com a mesma descrição na mesma instituição. Como exemplo, o segundo invariante estrutural encontra-se representado em 3.10.

```
+servico(_,Desc,Inst,_) :: (solucoes(Ls, servico(_,Desc,Inst,_), S),
                            countElements(S,N),
                            N == 1).
```

Figura 3.10: Invariante Estrutural: Não permitir a inserção de serviços com a mesma descrição na mesma instituição

Assim, para registar um Serviço foi criado o predicado 3.11.

```
registarS: ID, Descrição, Instituição, Cidade -> {V,F}
registarS(Id,Desc,Inst,Cidade) :- evolucao(servico(Id,Desc,Inst,Cidade)).
```

Figura 3.11: Predicado de Registar Serviço

Quanto à sua remoção, apenas poderá ser efetuada se não existir qualquer consulta cujo ID seja o ID do serviço a remover. Por isso, foi definido um invariante referencial que impede essa remoção em 3.12.

```
-servico(Id,_,_,_) :: nao(consulta(_,_,Id,_)).
```

Figura 3.12: Invariante Referencial: Não permitir a remoção de serviços se existirem consultas desse serviço

3.2.4 Registo/Remoção de Consulta

Uma Consulta poderá ser registada caso não exista uma consulta com os mesmos parâmetros, o seu custo seja superior a 0, esteja associada a um utente que existe na base de conhecimento, esteja associada a um serviço que existe na base de conhecimento e tenha uma data válida. Por isso, foram definidos 2 invariantes estruturais e 3 invariantes referenciais. Como exemplo, o invariante estrutural que não permite a inserção de consultas de utentes que não existem na base de conhecimento encontra-se defindo em 3.13.

```
+consulta(_,IdU,_,_) :: utente(IdU,_,_,_).
```

Figura 3.13: Invariante Referencial: Não permitir a inserção de novas consultas se não existir Utente

Assim, para registar uma Consulta, foi definido o predicado 3.14.

```
registarC: Data, IDUtente, IDServiço, Custo -> {V,F}
registarC(Data,IdU,IdS,Custo) :- evolucao(consulta(Data,IdU,IdS,Custo)).
```

Figura 3.14: Predicado de Registar Consulta

Quanto à sua remoção, poderá ser removida com recurso ao predicado 3.15.

```
apagarC: Data, IDUtente, IDServiço, Custo -> {V,F}
apagarC(Data,ID,IDS,C) :- involucao(consulta(Data,ID,IDS,C)).
```

Figura 3.15: Predicado de Remover Consulta

3.2.5 Análise de Resultados

De forma a podermos verificar os predicados desenvolvidos nas secções anteriores, decidimos efetuar alguns testes com recurso ao *SICStus PROLOG*. Assim, a base de conhecimento inicial é a que se encontra na figura 3.16.

```
| ?- listing(utente).
utente(1, 'Andre', 20, 'Esposende').
utente(2, 'Joao', 20, 'Braga').
utente(3, 'Luis', 21, 'Braga').
utente(4, 'Rafaela', 20, 'Braga').
utente(5, 'Joaquim', 23, 'Lisboa').
utente(6, 'Anabela', 27, 'Portalegre').
utente(7, 'Vitorino', 30, 'Lisboa').
utente(8, 'Zeferino', 37, 'Faro').
utente(9, 'Miguelito', 42, 'Outeiro').
utente(10, 'Jacinto', 43, 'Faro').
utente(11, 'Orlando', 50, 'Braga').
utente(12, 'Nestor', 45, 'Esposende').
utente(13, 'Moura', 55, 'Viana do Castelo').
utente(14, 'Barros', 33, 'Paredes de Coura').
utente(15, 'Proenca', 60, 'Figueira da Foz').
```

Figura 3.16: Base inicial do conhecimento de Utente

De seguida, foi possível adicionar um novo utente, uma vez que nenhum dos parâmetros violava os invariantes definidos, tal como está demonstrado na figura 3.17.

```
| ?- registarU(16,'Esteves',32,'Paredes de Coura','M').
yes
| ?- listing(utente).
utente(1, 'Andre', 20, 'Esposende').
utente(2, 'Joao', 20, 'Braga').
utente(3, 'Luis', 21, 'Braga').
utente(4, 'Rafaela', 20, 'Braga').
utente(5, 'Joaquim', 23, 'Lisboa').
utente(6, 'Anabela', 27, 'Portalegre').
utente(7, 'Vitorino', 30, 'Lisboa').
utente(8, 'Zeferino', 37, 'Faro').
utente(9, 'Miguelito', 42, 'Outeiro').
utente(10, 'Jacinto', 43, 'Faro').
utente(11, 'Orlando', 50, 'Braga').
utente(12, 'Nestor', 45, 'Esposende').
utente(13, 'Moura', 55, 'Viana do Castelo').
utente(14, 'Barros', 33, 'Paredes de Coura').
utente(15, 'Proenca', 60, 'Figueira da Foz').
utente(16, 'Esteves', 32, 'Paredes de Coura').
```

Figura 3.17: Base do conhecimento após registo de Utente

No entanto, se tentarmos adicionar um utente com um ID já utilizado, não é possível realizar essa operação, como demonstrado na figura 3.18.

```
| ?- registarU(14,'Estevão',17,'Bragança','F').
no
| ?- listing(utente).
utente(1, 'Andre', 20, 'Esposende').
utente(2, 'Joao', 20, 'Braga').
utente(3, 'Luis', 21, 'Braga').
utente(4, 'Rafaela', 20, 'Braga').
utente(5, 'Joaquim', 23, 'Lisboa').
utente(6, 'Anabela', 27, 'Portalegre').
utente(7, 'Vitorino', 30, 'Lisboa').
utente(8, 'Zeferino', 37, 'Faro').
utente(9, 'Miguelito', 42, 'Outeiro').
utente(10, 'Jacinto', 43, 'Faro').
utente(11, 'Orlando', 50, 'Braga').
utente(12, 'Nestor', 45, 'Esposende').
utente(13, 'Moura', 55, 'Viana do Castelo').
utente(14, 'Barros', 33, 'Paredes de Coura').
utente(15, 'Proenca', 60, 'Figueira da Foz').
utente(16, 'Esteves', 32, 'Paredes de Coura').
```

Figura 3.18: Base do conhecimento após tentativa falhada de registo de Utente

Relativamente aos serviços, a base inicial de conhecimento é a que se encontra na figura 3.16.

```
| ?- listing(servico).
servico(1, 'Ortodontia', 'Hospital de S.Marcos', 'Braga').
servico(2, 'Medicina Geral', 'Clinica de Santa Tecla', 'Braga').
servico(3, 'Oftalmologia', 'Hospital Privado XPTO', 'Barcelos').
servico(4, 'Ortopedia', 'Hospital de Ourique', 'Lisboa').
servico(5, 'Psicologia', 'Hospital da Nossa Senhora do 20', 'Guimaraes').
servico(6, 'Medicina Dentaria', 'Hospital de S.Marcos', 'Braga').
servico(7, 'Oncologia', 'Hospital da Luz', 'Lisboa').
servico(8, 'Psiquiatria', 'Clinica de Santa Tecla', 'Braga').
servico(9, 'Dermatologia', 'Hospital Privado XPTO', 'Barcelos').
```

Figura 3.19: Base inicial do conhecimento de Serviço

De seguida, foi adicionado um novo serviço que não violava nenhum dos invariantes definidos, pelo que

obteve sucesso, como é possível verificar na figura 3.20.

```
yes
| ?- registarS(10,'Medicina Familiar','Hospital de S.Marcos','Braga').
yes
| ?- listing(servico).
servico(1, 'Ortodontia', 'Hospital de S.Marcos', 'Braga').
servico(2, 'Medicina Geral', 'Clinica de Santa Tecla', 'Braga').
servico(3, 'Oftalmologia', 'Hospital Privado XPTO', 'Barcelos').
servico(4, 'Ortopedia', 'Hospital de Ourique', 'Lisboa').
servico(5, 'Psicologia', 'Hospital da Nossa Senhora do 20', 'Guimaraes').
servico(6, 'Medicina Dentaria', 'Hospital de S.Marcos', 'Braga').
servico(7, 'Oncologia', 'Hospital da Luz', 'Lisboa').
servico(8, 'Psiquiatria', 'Clinica de Santa Tecla', 'Braga').
servico(9, 'Dermatologia', 'Hospital Privado XPTO', 'Barcelos').
servico(10, 'Medicina Familiar', 'Hospital de S.Marcos', 'Braga').
```

Figura 3.20: Base do conhecimento após registo de Serviço

No entanto, se tentarmos remover um serviço que tem consultas associadas (como é o caso do serviço com ID = 5), a base de conhecimento mantém-se inalterada, como é demonstrado na figura 3.21.

```
| ?- apagarS(5).
no
| ?- listing(servico).
servico(1, 'Ortodontia', 'Hospital de S.Marcos', 'Braga').
servico(2, 'Medicina Geral', 'Clinica de Santa Tecla', 'Braga').
servico(3, 'Oftalmologia', 'Hospital Privado XPTO', 'Barcelos').
servico(4, 'Ortopedia', 'Hospital de Ourique', 'Lisboa').
servico(6, 'Medicina Dentaria', 'Hospital de S.Marcos', 'Braga').
servico(7, 'Oncologia', 'Hospital da Luz', 'Lisboa').
servico(8, 'Psiquiatria', 'Clinica de Santa Tecla', 'Braga').
servico(9, 'Dermatologia', 'Hospital Privado XPTO', 'Barcelos').
servico(10, 'Medicina Familiar', 'Hospital de S.Marcos', 'Braga').
servico(5, 'Psicologia', 'Hospital da Nossa Senhora do 20', 'Guimaraes').
```

Figura 3.21: Base do conhecimento após tentativa falhada de remoção de Serviço

Quanto às consultas, a base inicial de conhecimento é a que está representada na figura 3.22.

```
| ?- listing(consulta).
consulta(data(19,3,2019), 1, 3, 25).
consulta(data(23,4,2019), 2, 1, 3).
consulta(data(3,3,2019), 1, 3, 33).
consulta(data(7,7,2019), 3, 2, 15).
consulta(data(7,7,2019), 1, 2, 15).
consulta(data(8,3,2019), 5, 7, 30).
consulta(data(25,4,2019), 8, 8, 8).
consulta(data(3,3,2023), 15, 5, 40).
consulta(data(4,9,2019), 12, 7, 50).
consulta(data(24,12,2019), 9, 5, 20).
consulta(data(1,1,2020), 13, 6, 70).
consulta(data(4,5,2019), 14, 3, 25).
consulta(data(8,8,2020), 10, 4, 35).
consulta(data(1,1,2020), 11, 6, 60).
consulta(data(18,6,2010), 8, 6, 30).
```

Figura 3.22: Base do conhecimento de Consulta

De seguida, foi realizada uma tentativa de adicionar uma consulta realizada numa data inválida. A base de conhecimento permaneceu alterado, como visível na figura 3.23.

```
| ?- registarC(data(31,4,2018),7,9,23).
no
| ?- listing(consulta).
consulta(data(19,3,2019), 1, 3, 25).
consulta(data(23,4,2019), 2, 1, 3).
consulta(data(3,3,2019), 1, 3, 33).
consulta(data(7,7,2019), 3, 2, 15).
consulta(data(7,7,2019), 1, 2, 15).
consulta(data(8,3,2019), 5, 7, 30).
consulta(data(25,4,2019), 8, 8, 8).
consulta(data(3,3,2023), 15, 5, 40).
consulta(data(4,9,2019), 12, 7, 50).
consulta(data(24,12,2019), 9, 5, 20).
consulta(data(1,1,2020), 13, 6, 70).
consulta(data(4,5,2019), 14, 3, 25).
consulta(data(8,8,2020), 10, 4, 35).
consulta(data(1,1,2020), 11, 6, 60).
consulta(data(18,6,2010), 8, 6, 30).
```

Figura 3.23: Base do conhecimento após tentativa de registo de Consulta

Por fim, foi removida uma consulta da base de conhecimento, sendo que esta ficou tal como visível na figura 3.24.

```
| ?- apagarC(data(18,6,2010),8,6,30).
yes
| ?- listing(consulta).
consulta(data(19,3,2019), 1, 3, 25).
consulta(data(23,4,2019), 2, 1, 3).
consulta(data(3,3,2019), 1, 3, 33).
consulta(data(7,7,2019), 3, 2, 15).
consulta(data(7,7,2019), 1, 2, 15).
consulta(data(8,3,2019), 5, 7, 30).
consulta(data(25,4,2019), 8, 8, 8).
consulta(data(3,3,2023), 15, 5, 40).
consulta(data(4,9,2019), 12, 7, 50).
consulta(data(24,12,2019), 9, 5, 20).
consulta(data(1,1,2020), 13, 6, 70).
consulta(data(4,5,2019), 14, 3, 25).
consulta(data(8,8,2020), 10, 4, 35).
consulta(data(1,1,2020), 11, 6, 60).
```

Figura 3.24: Base do conhecimento após remoção de Consulta

3.3 Identificar instituições prestadoras de serviços

Tendo definido o predicado `solucoes`, a identificação de todas as instituições prestadoras de serviços é bastante simples, e está descrita em 3.25.

```
todasInst: ListaInstituições -> {V,F}
todasInst(S) :- solucoes(Inst,servico(_,_,Inst,_),X),
          semRepetidos(X,S).
```

Figura 3.25: Predicado de todas as instituições

Assim, é possível obter todas as instituições na base de conhecimento se existir prova de que existem serviços prestados nessas instituições, sendo que o resultado final é apresentado sem repetição de instituições.

3.3.1 Análise de Resultados

De forma a testarmos este nosso predicado, tendo como base de conhecimento aquela exposta em 3.2.5, obtivemos os resultados pretendidos, tal como demonstrado na figura 3.26.

```
| ?- todasInst(X).
X = ['Hospital de Ourique','Hospital da Nossa Senhora do 20','Hospital de S.Mar
cos','Hospital da Luz','Clinica de Santa Tecla','Hospital Privado XPTO'] ?
yes
```

Figura 3.26: Lista de todas as instituições

3.4 Identificar utentes/serviços/consultas por critérios de seleção

Uma vez mais, tendo o predicado `soluções` definido, a identificação de utentes por critérios de seleção é bastante trivial. Por isso, definimos três predicados que apresentam uma lista de nomes no segundo argumento. São os que se apresentam em 3.27.

```
utentePorSexo: Sexo do Utente, Lista de Utentes  -> {V,F}
utentePorSexo(Sexo,L) :- solucoes(Nome, (sexo(Sexo,Id),utente(Id,Nome,_,_) ),L).

utentePorCidade: Cidade do Utente, Lista de Utentes  -> {V,F}
utentePorCidade(C,L) :- solucoes(Nome,utente(_,Nome,_,C),L).

utentePorIdade: Idade do Utente, Lista de Utentes  -> {V,F}
utentePorIdade(I,L) :- solucoes(Nome,utente(_,Nome,I,_),L).
```

Figura 3.27: Predicados de identificação de utentes por critérios de seleção

Uma vez que a secção seguinte aborda diversos critérios de seleção de serviços, optou-se por definir um predicado similar ao 3.25. O predicado desenvolvido encontra-se em 3.28, e lista a descrição de todos os serviços no sistema, bem como a instituição em que são prestados.

```
todosServicos: Lista de Servicos  -> {V,F}
todosServicos(L):- solucoes((S,I), servico(_,S,I,_),L).
```

Figura 3.28: Predicado de identificação de todos os serviços

Relativamente às consultas, foram desenvolvidos dois predicados. O primeiro, dada uma data, lista todas as as consultas efetuadas nessa data, num par (Nome, Descrição do Serviço prestado). Assume-se para este predicado que nenhum utente efetua mais que uma consulta do mesmo serviço na mesma data. O segundo predicado, dada uma data e um serviço, determina o número de consultas prestadas desse serviço nessa data. Ambos os predicados encontram-se explícitos em 3.29.

```
consultasPorDia: Dia, Lista de Consultas  -> {V,F}
consultasPorDia(D,L) :- solucoes((Nome,Desc),
        (consulta(D,IdU,IdS,_) ,
        utente(IdU,Nome,_,_),
        servico(IdS,Desc,_,_)) , L).

consultasServiço: Data, Serviço, NumConsultas  -> {V,F}
consultasServico(D,S,N) :-  servico(IdS,S,_,_),
        solucoes(Id , consulta(D,Id,IdS,_), L),
        countElements(L,N).
```

Figura 3.29: Predicado de identificação de consultas por critérios de seleção

3.4.1 Análise de Resultados

De forma a testarmos os predicados desenvolvidos, tendo como base de conhecimento aquela definida em 3.2.5, foi possível obter os resultados pretendidos, tal como definido nas figuras apresentadas de seguida.

```
| ?- utentePorSexo('F',X).
X = ['Rafaela','Anabela'] ?
yes
```

Figura 3.30: Resultado do predicado utentePorSexo

Na figura 3.30, é possível observar o resultado de inquirir quais os utentes do sexo feminino na base de conhecimento. São as utentes Rafaela e Anabela.

```
| ?- utentePorCidade('Esposende',X).
X = ['Andre','Nestor'] ?
yes
```

Figura 3.31: Resultado do predicado utentePorCidade

Na figura 3.31, é possível observar o resultado de inquirir quais os utentes da cidade de Esposende na base de conhecimento. São os utentes André e Nestor.

```
| ?- utentePorIdade(20,X).
X = ['Andre','Joao','Rafaela'] ?
yes
```

Figura 3.32: Resultado do predicado utentePorIdade

Na figura 3.32, é possível observar o resultado de inquirir quais os utentes que têm 20 anos. São os utentes André, João e Rafaela.

```
| ?- todosServicos(X).
X = [('Ortodontia','Hospital de S.Marcos'),('Medicina Geral','Clinica de Santa
Tecla'),('Oftalmologia','Hospital Privado XPTO'),('Ortopedia','Hospital de Ouri
que'),('Psicologia','Hospital da Nossa Senhora do 20'),('Medicina Dentaria','Ho
spital de S.Marcos'),('Oncologia','Hospital da Luz'),('Psiquiatria','Clinica de
 Santa Tecla'),('Dermatologia','Hospital Privado XPTO')] ?
yes
```

Figura 3.33: Resultado do predicado todosServicos

Na figura 3.33, é possível observar o resultado de inquirir quais os serviços registados na base de conhecimento. Os resultados são apresentados da forma (Serviço, Instituição).

```
| ?- consultasPorDia(data(7,7,2019),X).
X = [('Luis','Medicina Geral'),('Andre','Medicina Geral')] ?
yes
```

Figura 3.34: Resultado do predicado consultasPorDia

Na figura 3.34, é possível observar o resultado de inquirir quais as consultas efetuadas a 7 de julho de 2019. Foi efetuada uma consulta de Medicina Geral ao utente Luís e uma consulta de Medicina Geral ao utente André.

```
| ?- consultasServico(data(7,7,2019),'Medicina Geral',X).
X = 2 ?
yes
```

Figura 3.35: Resultado do predicado consultasServico

Na figura 3.35, é possível observar o resultado de inquirir qual o número de serviços de Medicina Geral prestados no dia 7 de julho de 2019. Foram realizadas 2 consultas de Medicina Geral.

3.5 Identificar serviços prestados por instituição/cidade/datas/custo

De forma a ser possível identificar serviços prestados por cidade, data e custo, foram desenvolvidos três predicados, que se apresentam em 3.36. Relativamente a serviços realizados por instituição, esse predicado será desenvolvido na secção 3.7.

```
servicosPorCidade: Cidade, Lista de Servicos  -> {V,F}
servicosPorCidade(C,L) :- solucoes(S, servico(_,S,_,C),R),
        semRepetidos(R,L).

servicosPorData: Data, Lista de Servicos  -> {V,F}
servicosPorData(Data,L) :-
        solucoes(Desc,(consulta(Data,_,IdServico,_),
        servico(IdServico,Desc,_,_)),R),
        semRepetidos(R,L).

servicosPorCusto: Custo, Lista de Serviços -> {V,F}
servicosPorCusto(Custo,L) :- solucoes(Desc,
        (consulta(_,_,IdServico,Custo) ,
        servico(IdServico,Desc,_,_)), R),
        semRepetidos(R,L).
```

Figura 3.36: Predicado de identificação de serviços prestados por cidade, data e custo

Assim, no primeiro predicado, dada uma cidade, são apresentadas as descrições de todos os serviços disponíveis nessa cidade, sendo removidos serviços repetidos através do predicado `semRepetidos`.

No predicado `servicosPorData`, dada uma data, apresenta todos os serviços prestados nessa data. Uma vez mais, são removidas todas as ocorrências repetidas, uma vez que se trata de informação redundante.

Por fim, o predicado `servicosPorCusto` permite que, dado um custo, seja apresentada uma lista de serviços que tenham sido prestados em consultas com esse custo.

3.5.1 Análise de Resultados

Uma vez mais, de forma a testarmos os predicados desenvolvidos, tivemos em consideração a base de conhecimento definida em 3.2.5. Foi então possível obter os resultados pretendidos, tal como definido nas figuras apresentadas de seguida.

```
| ?- servicosPorCidade('Braga',X).
X = ['Ortodontia','Medicina Geral','Medicina Dentaria','Psiquiatria'] ?
yes
```

Figura 3.37: Resultado do predicado servicosPorCidade

Na figura 3.37, é possível observar o resultado de inquirir quais os serviços prestados na cidade de Braga. São estes Ortodontia, Medicina Geral, Medicina Dentária e Psiquiatria.

```
| ?- servicosPorData(data(7,7,2019),X).
X = ['Medicina Geral'] ?
yes
```

Figura 3.38: Resultado do predicado servicosPorData

Na figura 3.38, é possível observar o resultado de inquirir quais os serviços prestados a 7 de julho de 2019. Foram prestados serviços de Medicina Geral.

```
| ?- servicosPorCusto(15,X).
X = ['Medicina Geral'] ?
yes
```

Figura 3.39: Resultado do predicado servicosPorCusto

Na figura 3.39, é possível observar o resultado de inquirir quais os serviços prestados que tiveram um custo de 15 unidades monetárias. Apenas consultas de Medicina Geral tiveram consultas com esse custo.

3.6 Identificar os utentes de um serviço/instituição

De forma a ser possível identificar os utentes de um dado serviço ou instituição, foram desenvolvidos dois predicados, ambos tendo por base o predicado `solucoes`. Encontram-se na figura 3.40.

```
utentesPorServico: Serviço, Lista de Utentes -> {V,F}
utentesPorServico(IdS,L) :- solucoes(Nome, (consulta(_,IdU,IdS,_) ,
    servico(IdS,_,_,_) , utente(IdU,Nome,_,_)), R),
                semRepetidos(R,L).

utentesPorInstituicao: Instituicao, Lista de Utentes -> {V,F}
utentesPorInstituicao(Ins,L) :- solucoes(Nome,
    (consulta(_,IdU,IdS,_) , servico(IdS,_,Ins,_) ,
    utente(IdU,Nome,_,_)), R),
    semRepetidos(R,L).
```

Figura 3.40: Predicado de identificação de utentes por serviço e instituição

Assim, o primeiro predicado, dado um ID de serviço, devolve uma lista com os nomes dos utentes a quem foram realizadas consultas desse serviço, sem repetição de nomes.
O segundo predicado, dada uma instituição, é devolvida uma lista com os nomes dos utentes a quem foram realizadas consultas nessa instituição, também sem haver repetição de nomes.

3.6.1 Análise de Resultados

Uma vez mais, para verificarmos a funcionalidade dos predicados desenvolvidos, utilizamos a base de conhecimento exposta em 3.2.5.

```
| ?- utentesPorServico(3,X).
X = ['Andre','Barros'] ?
yes
```

Figura 3.41: Resultado do predicado utentesPorServico

Na figura 3.41, é inquirido o nome dos utentes do serviço de ID = 3. São os utentes André e Barros.

```
| ?- utentesPorInstituicao('Hospital de S.Marcos',X).
X = ['Joao','Moura','Orlando','Zeferino'] ?
yes
```

Figura 3.42: Resultado do predicado utentesPorInstituicao

Na figura 3.42, é inquirido o nome dos utentes da instituição Hospital de S. Marcos. São os utentes João, Moura, Orlando e Zeferino.

3.7 Identificar serviços realizados por utente/instituição/cidade

De forma a ser possível identificar os serviços realizados por um utente ou numa instituição, foram desenvolvidos dois predicados similares aos desenvolvidos em 3.5. No entanto, foi também desenvolvido um predicado diferente, que poderá receber uma variação de 3 parâmetros. Estes 3 predicados estão representados na figura 3.43.

```
servicosPorUtente: Utente, Lista de Serviços -> {V,F}
servicosPorUtente(IdU,L) :- solucoes(Desc, (consulta(_,IdU,IdS,_) ,
    servico(IdS, Desc,_,_)), R),
    semRepetidos(R,L).

servicosPorInstituição: Instituicao, Lista de Serviços -> {V,F}
servicosPorInstituicao(Inst,L) :- solucoes(Desc,
    (consulta(_,IdU,IdS,_) , servico(IdS, Desc,_,_)), R),
    semRepetidos(R,L).

servicosUIC: IdUtente, Instituicao, Cidade, Lista de Serviços -> {V,F}
servicosUIC(IdU,Inst,Cid,L) :- solucoes((Nome,Desc,Inst,Cid) ,
    (consulta(_,IdU,IdS,_), servico(IdS,Desc,Inst,Cid),
    utente(IdU,Nome,_,_)), R),
    semRepetidos(R,L).
```

Figura 3.43: Predicado de identificação de serviços por utente, instituição e variações

No primeiro predicado, dado o ID de um utente, é devolvida uma lista dos serviços utilizados por esse utente, sem repetição de nomes.

No predicado `servicosPorInstituicao`, dada uma instituição, devolve os serviços realizados nessa instituição, sem repetição de nomes.

Por fim, o último predicado permite que dada uma combinação de ID de utente, instituição e/ou cidade, seja devolvida uma lista de serviços realizados nessa combinação de parâmetros.

3.7.1 Análise de Resultados

Tal como nas secções de Análise de Resultados anteriores, a base de conhecimento que serve de caso de teste aos predicados desenvolvidos é a que se encontra em 3.2.5. Os resultados da aplicação dos predicados são os que se apresentam de seguida.

```
| ?- servicosPorUtente(1,X).
X = ['Oftalmologia','Medicina Geral'] ?
yes
```

Figura 3.44: Resultado do predicado servicosPorUtente

Na figura 3.44, é inquirido o nome dos serviços prestados ao utente com o ID = 1. Foram prestadas consultas de Oftalmologia e Medicina Geral.

```
| ?- servicosPorInstituicao('Hospital de S.Marcos',X).
X = ['Ortodontia','Medicina Geral','Psiquiatria','Oncologia','Psicologia','Ofta
lmologia','Ortopedia','Medicina Dentaria'] ?
yes
```

Figura 3.45: Resultado do predicado servicosPorInstituicao

Na figura 3.45, é inquirido o nome dos serviços prestados na instituição Hospital de S. Marcos. São prestadas consultas de Ortodontia, Medicina Geral, Psiquiatria, Oncologia, Psicologia, Oftalmologia, Ortopedia e Medicina Dentária.

```
| ?- servicosUIC(1,X,Y,Z).
Z = [('Andre','Oftalmologia','Hospital Privado XPTO','Barcelos'),('Andre','Medi
cina Geral','Clinica de Santa Tecla','Braga')] ?
yes
```

Figura 3.46: Resultado do predicado servicosUIC

Na figura 3.46, é inquirido um conjunto de informações sobre os serviços prestados ao utente com ID = 1. Foram prestados 2 serviços, em 2 instituições e 2 cidades distintas, sendo o resultado apresentado na forma (Nome de Utente, Serviço, Instituição, Cidade).

3.8 Calcular o custo das consultas por utente/serviço/instituição/data

Por fim, para calcular os custos totais de cuidados de saúde por parte de utentes, serviços, instituições e/ou datas, foram desenvolvidos 2 predicados. O primeiro permite a identificação do custo total dos cuidados de saúde, dado um ID de utente, de serviço, uma instituição e/ou uma data. O segundo, dada uma data, devolve a soma dos custos das consultas que ocorreram nessa data. Ambos os predicados encontram-se em 3.47.

```
custoUSID: IdUtente, Serviço, Instituicao, Data, Custo Total -> {V,F}
custoUSID(IdU,IdS,Inst,Data,R) :- solucoes(C, (consulta(Data,IdU,IdS,C) ,
    servico(IdS,_,Inst,_)), L) ,
    sumElements(L,R).

ganhoPorDia: Data, Valor  -> {V,F}
ganhoPorDia(D,T) :-  solucoes(Valor , consulta(D,_,_,Valor), L),
               sumElements(L,T).
```

Figura 3.47: Predicados de cálculo de custos de consultas

O primeiro predicado devolve a soma dos custos das consultas que verifiquem os critérios escolhidos (quer seja ID de utente, de serviço, instituição, data ou uma combinação destes), enquanto que o segundo predicado se limita a devolver a soma dos custos de consultas que tenham ocorrido num determinado dia.

3.8.1 Análise de Resultados

Tal como nas restantes secções de Análise de Resultados, a base de conhecimento a partir da qual foram elaborados predicados teste é a mesma que a exposta em 3.2.5. Os resultados da aplicação dos predicados desenvolvidos anteriormente são os que se apresentam de seguida.

```
| ?- custoUSID(U,6,I,D,R).
R = 160 ?
yes
```

Figura 3.48: Resultado do predicado custoUSID

Na figura 3.48, é inquirido o custo total de consultas realizadas em relação ao serviço com ID = 6. O custo total é de 160.

```
| ?- ganhoPorDia(data(7,7,2019),X).
X = 30 ?
yes
```

Figura 3.49: Resultado do predicado ganhoPorDia

Na figura 3.49, é inquirido o custo total de consultas realizadas a 7 de julho de 2019. O custo total é de 30.

Capítulo 4

Conclusões

Com este trabalho foi possível colocar em prática os conhecimentos adquiridos ao longo das aulas relativos à Programação em Lógica. Consideramos que foram desenvolvidos predicados suficientes para cumprir com as expectativas definidas no enunciado. Para além disso, pudemos também transpor predicados sugeridos tanto nas aulas teóricas como nas aulas práticas para este caso prático, tais como o `não` e `soluções`.

Apesar disso, consideramos que este relatório acabou por ficar demasiado extenso, uma vez que consideramos necessária a documentação efetiva e respetiva demonstração dos principais predicados desenvolvidos. Ainda assim, não foram incluídos alguns predicados auxiliares para manipulação de listas, bem como todos os invariantes definidos.

Por fim, consideramos que tendo realizado este trabalho, o grupo se sente mais confortável para o desenvolvimento ou aprofundamento deste caso prático na linguagem PROLOG.