

Diogo Braga

Diogo Silva

João Silva

Ricardo Caçador

Ricardo Veloso

**1º Exercício do Trabalho de Grupo:**

**Programação em Lógica e Invariantes**

**Universidade do Minho**

Departamento de Informática

**1º Exercício do Trabalho de Grupo:**

**Programação em Lógica e Invariantes**

Diogo Braga

Diogo Silva

João Silva

Ricardo Caçador

Ricardo Veloso

**Universidade do Minho**

Departamento de Informática



# Resumo

O trabalho representado neste relatório foi desenvolvido no âmbito da UC de Sistemas de Representação de Conhecimento e Raciocínio por forma a desenvolver competências na utilização da linguagem de programação em lógica - PROLOG.

Este exercício consistiu no desenvolvimento de uma base de conhecimento e raciocínio para caracterizar um universo de discurso na área da prestação de cuidados de saúde.

Este relatório irá explicar todo o processo que envolveu a criação dessa base até ao resultado final.

**Tabela de Conteúdos**

[Resumo 3](#_Toc4167011)

[1 - Introdução 5](#_Toc4167012)

[2 - Descrição do Trabalho 6](#_Toc4167013)

[2.1 - Representação do Conhecimento 6](#_Toc4167014)

[2.1.1 – Utente 6](#_Toc4167015)

[2.1.2 - Serviço 7](#_Toc4167016)

[2.1.3 - Consulta 7](#_Toc4167017)

[2.1.4 – Data 7](#_Toc4167018)

[2.1.5 – Médico 8](#_Toc4167019)

[2.1.6 – Seguro 8](#_Toc4167020)

[2.2 – Registos e Remoções 8](#_Toc4167021)

[2.2.1 – Registar/Remover Utentes 8](#_Toc4167022)

# 1 - Introdução

Esta primeira fase tem como objetivo a criação de um sistema representação de conhecimento e raciocínio que caracterize a estrutura de uma área de prestação de cuidados de saúde. A criação deste sistema é feita através da utilização da linguagem de programação PROLOG.

Para o efeito, foi-nos apresentado um panorama possível para caracterizar o conhecimento bem como um conjunto de funcionalidades que o sistema deve respeitar.

De seguida, iremos apresentar todas as soluções realizadas pelo grupo para a consumação do exercício proposto bem como as extensões de conhecimento implementadas no sistema.

# 2 - Descrição do Trabalho

Para que todo o conhecimento pretendido no enunciado fosse concebido foram desenvolvidos seis predicados:

* utente: IdUt,Nome,Idade,Cidade,Seguro -> {V,F}
* serviço: IdServ,Descrição,Insituição,Cidade -> {V,F}
* consulta: Data,IdUt,IdServ,Custo,IdMed -> {V,F}
* data: D, M, A -> {V,F}
* medico: IdMed, Nome, Idade, IdServ -> {V,F}
* seguro: IdSeg, Descrição, Taxa -> {V,F}

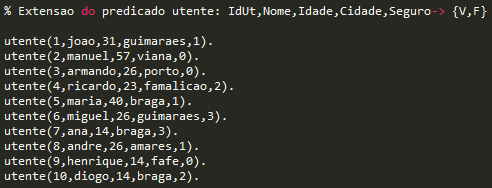
Como se pode verificar, existem algumas discrepâncias no que conta ao panorama inicialmente apresentado no enunciado do projeto. Passaremos de seguida a explicar todo o processo que nos levou a representar o conhecimento desta forma.

## 2.1 - Representação do Conhecimento

Neste subcapítulo serão explicados com detalhe todos os predicados acima apresentados, bem como os factos associados a estes.

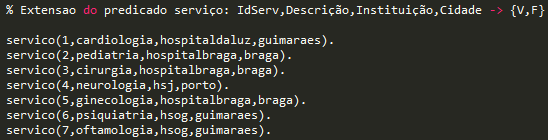
## 2.1.1 – Utente

Um utente é caracterizado pelo seu ID, Nome, Idade, Cidade e Id do seguro (Seguro) a que tem direito:



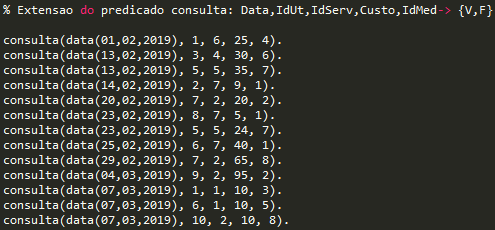
## 2.1.2 - Serviço

Um serviço é caracterizado pelo seu ID, a sua descrição, a instituição onde é prestado e a cidade onde se a encontra a instituição:



## 2.1.3 - Consulta

Uma consulta é caracterizada pela data em que foi realizada, o ID do utente que a frequentou, o ID do Serviço respetivo, o seu custo e o ID do médico que a realizou:



## 2.1.4 – Data

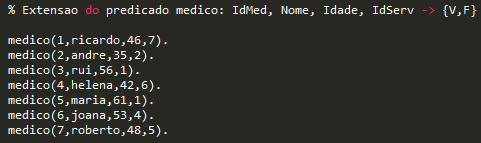
Uma data é caracterizada pelo seu dia, mês e ano.

No caso da data, estas apenas são criadas no contexto de consulta daí não existirem factos associados a este predicado a não ser aqueles que se encontram na figura acima.

O predicado data foi criado com o objetivo de dar mais coerência ao sistema através da criação de invariantes que irão ser apresentados mais à frente.

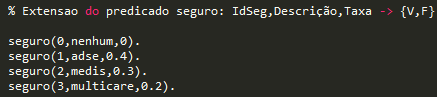
## 2.1.5 – Médico

Um médico é caracterizado pelo seu ID, Nome, Idade e Id do serviço ao qual pretence:



## 2.1.6 – Seguro

Um seguro é caracterizado pelo seu ID, Descrição e taxa de retorno associada a ele:



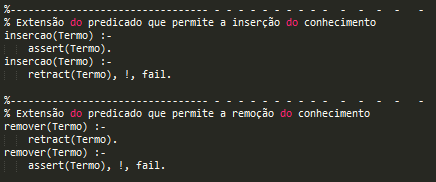
## 2.2 – Registos e Remoções

Para a realização dos diferentes registos e remoções foi necessário criar predicados auxiliaries que permitem a inserção e remoção de conhecimento de uma forma coesa e correta.

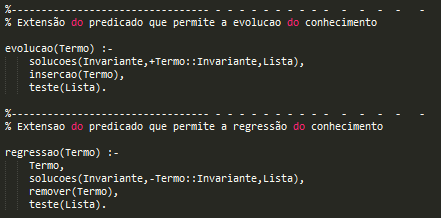
Em primeiro lugar os predicados “insercao” e “remover” que utilizam os predicados “*assert*” e “*retract*”.

O predicado “insercao” utiliza o “*assert*”, no caso de sucesso, para inserir um novo facto na base de conhecimento e, em caso de insucesso, utiliza o “*retract*” para não permitir a permanência do facto que tentamos inserir.

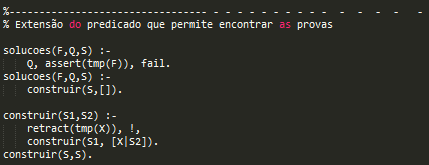
O predicado “remover” funciona da forma contrária, utilizando o “*retract*” em caso de sucesso e o “*assert*” em caso de insucesso.



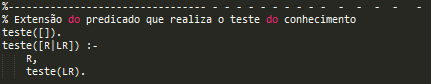
No entanto, estes predicados não são o suficiente para inserir ou remover conhecimento. É necessário obedecer a algumas regras que o grupo definiu. Por exemplo, não é permitido inserir um utente quando o id desse utente que estamos a tentar inserir já existe na base do conhecimento. Daí a criação dos seguintes predicados:



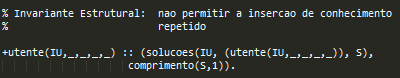
Estes predicados criam uma lista com todos os invariantes respetivos ( - para os invariantes de remoção e + para os invariantes de inserção) utilizando o predicado “solucoes” que funciona como um motor de pesquisa de provas.



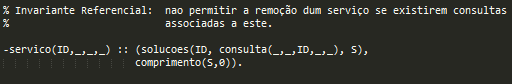
De seguida inserem (utilizando o predicado “insercao”) o termo pretendido na base do conhecimento e testam se o conhecimento presente é coeso e correto através do predicado “teste”.



Para um melhor esclarecimento sobre os invariantes criados pelo grupo mostramos alguns exemplos destes para os diferentes predicados:



Neste caso, estamos a verificar se a lista de IDs criada pelo predicado “solucoes” tem apenas tamanho um. Assim temos a certeza que só existe um utente com o IdUtente passado com argumento.



Aqui, verificamos se o serviço em causa tem alguma consulta diretamente ligada a si. Caso isso aconteça não faz sentido remover um serviço que já esteja associado a uma determinada consulta.

## 2.2.1 – Registar/Remover Utentes