

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](#_Toc4336410)

[1 - Introdução 6](#_Toc4336411)

[2 - Descrição do Trabalho 7](#_Toc4336412)

[2.1 - Representação do Conhecimento 7](#_Toc4336413)

[2.1.1 – Utente 7](#_Toc4336414)

[2.1.2 - Serviço 8](#_Toc4336415)

[2.1.3 - Consulta 8](#_Toc4336416)

[2.1.4 – Data 8](#_Toc4336417)

[2.1.5 – Médico 9](#_Toc4336418)

[2.1.6 – Seguro 9](#_Toc4336419)

[2.2 – Registos e Remoções 9](#_Toc4336420)

[2.2.1 – Registar/Remover Utentes 12](#_Toc4336421)

[2.2.2 – Registar/Remover Serviços 12](#_Toc4336422)

[2.2.3 – Registar/Remover Consultas 13](#_Toc4336423)

[2.2.3.1 – Inserção de uma data no contexto de Consulta 13](#_Toc4336424)

[2.2.4 – Registar/Remover Médicos 14](#_Toc4336425)

[2.2.5 – Registar/Remover Seguros 14](#_Toc4336426)

[2.3 - Identificar as instituições prestadoras de serviços 15](#_Toc4336427)

[2.4 - Identificar utentes/serviços/consultas por critérios de seleção 15](#_Toc4336428)

[2.4.1 – Critérios de Seleção extras 16](#_Toc4336429)

[2.5 – Identificar serviços prestados por instituição, cidade, data ou custos 17](#_Toc4336430)

[2.6 – Identificar os utentes de um serviço ou instituição 18](#_Toc4336431)

[2.7 – Identificar serviços realizados por utente, instituição ou cidade 19](#_Toc4336432)

[2.7.1 – Identificar serviços realizados por utente 19](#_Toc4336433)

[2.7.2 – Identificar serviços realizados por instituição ou cidade 20](#_Toc4336434)

[2.8 - Calcular o custo total dos cuidados de saúde por utente/serviço/instituição/data/médico 20](#_Toc4336435)

[2.8.1 – Custo total dos cuidados de saúde por utente 21](#_Toc4336436)

[2.8.2 - Custo total dos cuidados de saúde por data/serviço/médico 22](#_Toc4336437)

[2.8.3 – Custo total dos cuidados de saúde por Instituição 23](#_Toc4336438)

[2.9 – Guardar e carregar factos através da utilização de um ficheiro de texto 23](#_Toc4336439)

[2.10 – Predicados auxiliares 24](#_Toc4336440)

[3 – Conclusão 25](#_Toc4336441)

# 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, Instituição, Cidade -> {V,F}
* consulta: Data, IdUt, IdServ, Custo, IdMed -> {V,F}
* data: Dia, Mes, Ano -> {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:

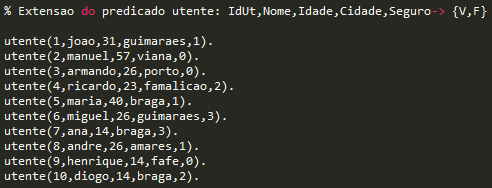


Figura 1 - Extensão do predicado Utente e alguns factos de Utente

## 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:

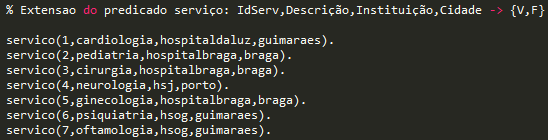


Figura 2 - Extensão do predicado Serviço e alguns factos de Serviço

## 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:

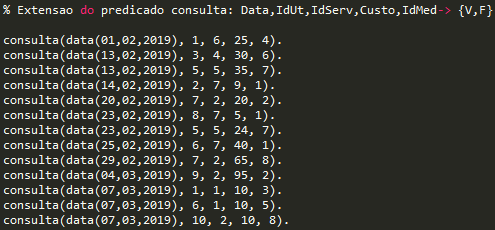


Figura 3 - Extensão do predicado Consulta e alguns factos de Consulta

## 2.1.4 – Data

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

No caso das datas, 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 pertence:

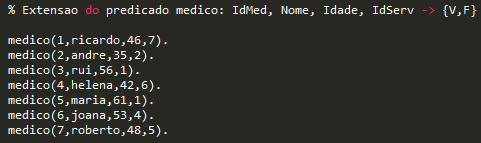


Figura 4 - Extensão do predicado Médico e alguns factos de Médico

## 2.1.6 – Seguro

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

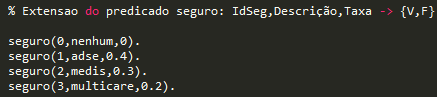


Figura 5 - Extensão do predicado Seguro e alguns factos de Seguro

## 2.2 – Registos e Remoções

Para a realização dos diferentes registos e remoções foi necessário criar predicados auxiliares 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 retrocesso, utiliza o “*retract*” para não permitir a permanência do facto que errado acabado de inserir.

O predicado “remover” funciona da forma contrária, utilizando o “*retract*” em caso de sucesso e o “*assert*” em caso de retrocesso, para voltar a inserir o facto acabado de ser removido.

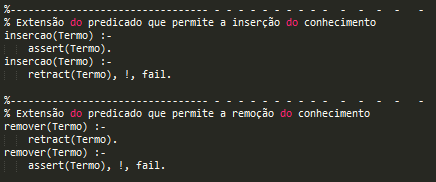


Figura 6 - Extensões dos predicados insercao e remover

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:

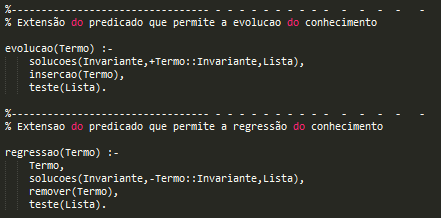


Figura 7 - Extensões dos predicados evolucao e regressao

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.

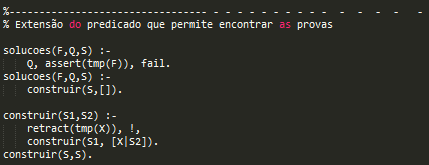


Figura 8 - Extensão do predicado solucoes

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”.

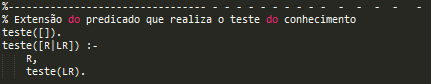


Figura 9 - Extensão do predicado teste

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

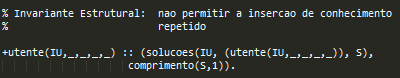


Figura 10 - Exemplo dum Invariante Estrutural para o predicado Utente

Inserção (+): 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 como argumento.

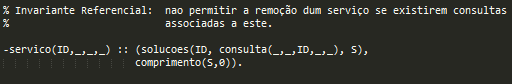


Figura 11 - Exemplo dum Invariante Referencial para o predicado Servico

Remoção (-): 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.

Antes de tudo isto foi necessário criar definições iniciais do SICStus PROLOG para que fosse possível implementar os invariantes:

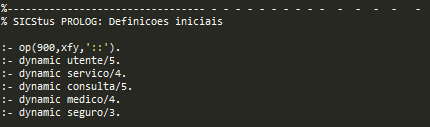


Figura 12 - Definições iniciais do ficheiro

## 2.2.1 – Registar/Remover Utentes

Para o registo e a remoção de Utentes foram criados dois predicados: “registarU” e “removerU”. Estes recebem os parâmetros respetivos ao utente que queremos registar ou remover e utilizam o predicado “evolucao” (no caso de registo) ou “regressao” (no caso de remoção).

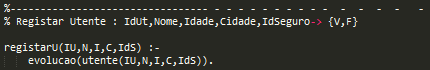


Figura 13 - Extensão do predicado registarU

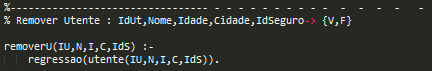


Figura 14 - Extensão do predicado removerU

## 2.2.2 – Registar/Remover Serviços

Para o registo e a remoção de Serviços foram criados dois predicados: “registarServ” e “removerServ”. Estes recebem os parâmetros respetivos ao serviço que queremos registar ou remover e utilizam o predicado “evolucao” (no caso de registo) ou “regressao” (no caso de remoção).

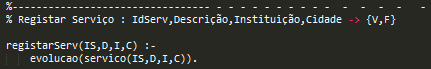


Figura 15 - Extensão do predicado registarServ

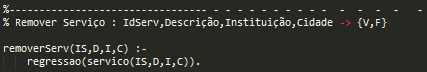


Figura 16 - Extensão do predicado removerServ

## 2.2.3 – Registar/Remover Consultas

Para o registo e a remoção de Consultas foram criados dois predicados: “registarConsulta” e “removerConsulta”. Estes recebem os parâmetros respetivos à consulta que queremos registar ou remover e utilizam o predicado “evolucao” (no caso de registo) ou “regressao” (no caso de remoção).

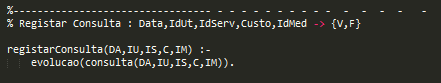


Figura 17 - Extensão do predicado registarConsulta

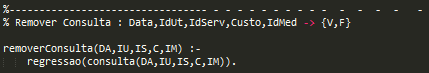


Figura 18 - Extensão do predicado removerConsulta

## 2.2.3.1 – Inserção de uma data no contexto de Consulta

Como já referido no ponto **2.1.4** o predicado data apenas existe no contexto de consulta mas, ainda assim, o grupo definiu algumas regras para a inserção das datas.

Posto isto, foram criados predicados que verificam se a data a inserir é válida. Podemos verificar isso na imagem seguinte:

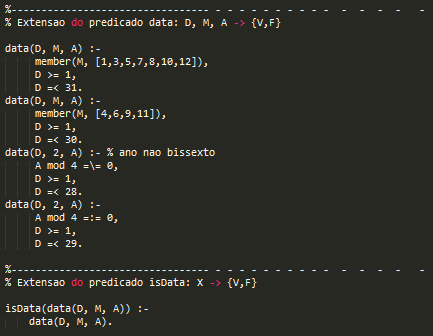


Figura 19 - Extensões dos predicados data e isData

Os predicados na imagem são utilizados para o invariante estrutural da consulta que não permite a inserção duma data que não seja válida:

inserirData.png

Figura 20 - Invariante Estrutural para o predicado Consulta

## 2.2.4 – Registar/Remover Médicos

Para o registo e a remoção de Médicos foram criados dois predicados: “registarM” e “removerM”. Estes recebem os parâmetros respetivos ao médico que queremos registar ou remover e utilizam o predicado “evolucao” (no caso de registo) ou “regressao” (no caso de remoção).

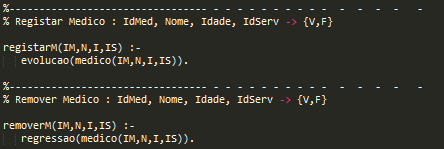


Figura 21 - Extensões dos predicados registarM e removerM

## 2.2.5 – Registar/Remover Seguros

Para o registo e a remoção de seguros foram criados dois predicados: “registarS” e “removerS”. Estes recebem os parâmetros respetivos ao seguro que queremos registar ou remover e utilizam o predicado “evolucao” (no caso de registo) ou “regressao” (no caso de remoção).

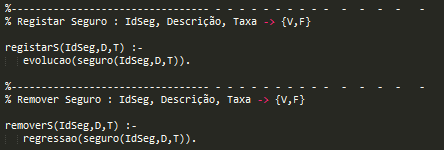


Figura 22 - Extensões dos predicados registarS e removerS

## 2.3 - Identificar as instituições prestadoras de serviços

Neste predicado apenas é utilizado o predicado “solucoes”, que cria uma lista com todas as instituições onde se situam os serviços. De seguida removemos qualquer repetição de instituição que possa existir na lista criada (isto porque vários serviços são realizados na mesma instituição).

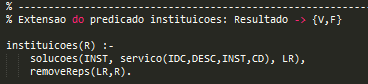


Figura 23 - Extensão do predicado instituicoes

O resultado deste predicado é, por exemplo:

instituicoesR.png

Figura 24 - Exemplo dum teste ao predicado instituicoes

## 2.4 - Identificar utentes/serviços/consultas por critérios de seleção

Todos estes predicados funcionam de forma equivalente em que é dado um critério (por exemplo o nome de um utente, uma cidade ou uma descrição de um serviço) e de seguida são procurados todos os utentes/serviços/consultas que contêm esse critério de seleção através do predicado “solucoes”.

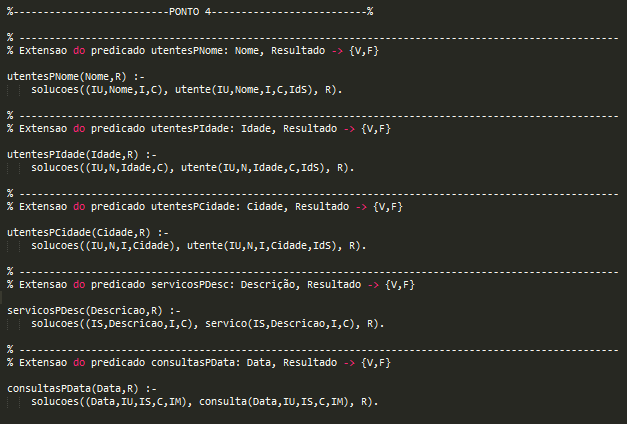


Figura 25 - Extensões de predicados que representam conhecimento selecionado

Apresentamos então, alguns possíveis resultados quando utilizamos estes predicados:

utentesPCidade.png

Figura 26 - Exemplo dum teste ao predicado utentesPCidade

consultasPData.png

Figura 27 - Exemplo dum teste ao predicado consultasPData

servicosPDesc.png

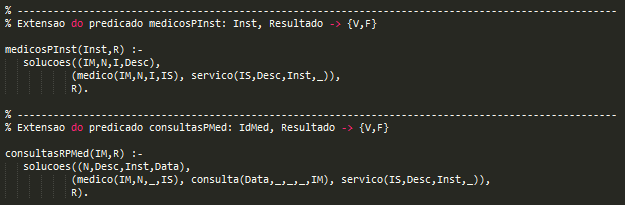
Figura 28 - Exemplo dum teste ao predicado servicosPDesc

## 2.4.1 – Critérios de Seleção extras

Na sequência da criação do predicado “medico” o grupo decidiu criar alguns identificadores por critério de seleção extras. Estes foram:

* medicos por instituição;
* consultas por medico.

Figura 29 - Extensões dos predicados medicosPInst e consultasRPMed



Demonstramos um exemplo de resolução para cada um destes predicados:

medicosPinst.png

Figura 30 - Exemplo dum teste ao predicado medicosPInst

consultasPmedico.png

Figura 31 - Exemplo dum teste ao predicado consultasRPMed

## 2.5 – Identificar serviços prestados por instituição, cidade, data ou custos

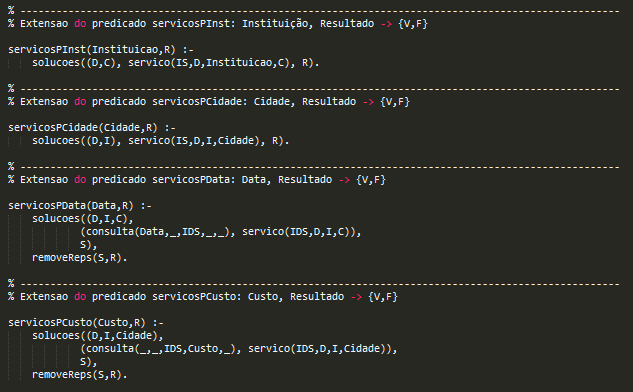
 Para este ponto utilizamos a mesma estratégia que a utilizada no ponto anterior ainda que, no caso da data e dos custos foi necessário verificar o ID do Serviço em todas as consultas que tivessem a data ou o custo pretendido e, de seguida, fazer a conexão com os serviços a que os IDs dos Serviços das consultas encontradas correspondessem. Neste caso também removemos as repetições na possibilidade da existência de um serviço ser prestados várias vezes na mesma data ou que tenha o mesmo custo.

Figura 32 - Extensões dos predicados relacionados com serviços

Alguns exemplos da utilização deste predicado:

servicosPCusto.png

Figura 33 - Exemplo dum teste ao predicado servicosPCusto

servicosPCidade.png

Figura 34 - Exemplo dum teste ao predicado servicosPCidade

## 2.6 – Identificar os utentes de um serviço ou instituição

Neste ponto passamos como argumento a descrição dum serviço ou uma instituição e desta forma obtemos os IDs dos Serviços correspondentes. De seguida utilizamos os IDs dos Serviços para encontrar as consultas já realizadas por esse serviço e destas retiramos os Ids dos Utentes que realizaram essa consulta. Após isto apenas temos que aceder aos Utentes respetivos aos IDs encontrados e inserir os seus IDs e Nomes na lista das soluções. Como no ponto anterior removemos as repetições no caso de elementos iguais na lista de soluções.

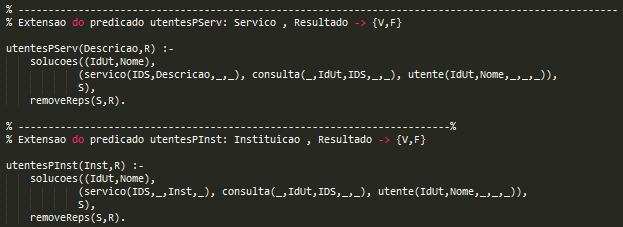


Figura 35 - Extensões dos predicados utentesPServ e utentesPInst

Alguns exemplos de resultados da utilização destes predicados:

utentesPservico.png

Figura 36 - Exemplo dum teste ao predicado utentesPServ

utentesPInst.png

Figura 37 - Exemplo dum teste ao predicado utentesPInst

## 2.7 – Identificar serviços realizados por utente, instituição ou cidade

Para uma melhor explicação desta funcionalidade decidimos dividir o ponto em dois subcapítulos devido à existência de diferenças no processo de identificação de serviços realizados por utente em relação aos processos de identificação de serviços realizados numa instituição/cidade.

## 2.7.1 – Identificar serviços realizados por utente

Neste caso vamos percorrer todas as consultas que contenham o ID do utente pretendido e retiramos o ID do serviço correspondente. De seguida guardamos numa lista, através do predicado “solucoes”, a descrição, instituição e cidade dos serviços. Para o caso de existirem repetições nessa lista usamos mais uma vez o predicado “removeReps” para que não se repita conhecimento.

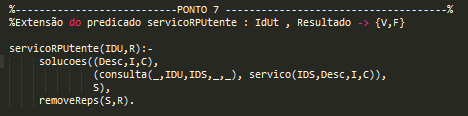


Figura 38 - Extensão do predicado servicoRPUtente

servicoRPUtenteS.pngUm exemplo de solução para este predicado é:

Figura 39 - Exemplo dum teste ao predicado servicoRPUtente

## 2.7.2 – Identificar serviços realizados por instituição ou cidade

Neste caso percorremos todas as consultas retirando os IDs dos serviços e depois guardamos numa lista, através do predicado “solucoes”, a descrição e a instituição/cidade que tenham como característica a instituição ou a cidade passada como argumento. Aqui também removemos as repetições no caso da existência de conhecimento repetido na lista.

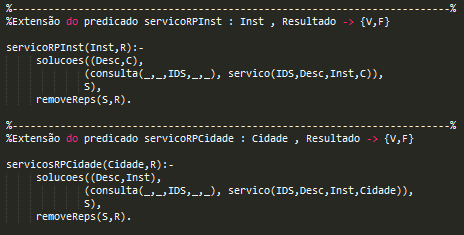


Figura 40 - Extensões dos predicados servicoRPInst e servicosRPCidade

Exemplos de soluções para estes predicados:

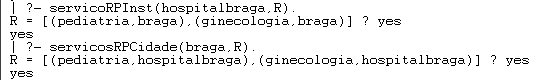


Figura 41 - Exemplo de testes aos 2 predicados

## 2.8 - Calcular o custo total dos cuidados de saúde por utente/serviço/instituição/data/médico

Tal como no ponto anterior, este ponto vai ser dividido em vários subcapítulos devido às diversas abordagens para responder ao ponto da forma mais correta possível.

## 2.8.1 – Custo total dos cuidados de saúde por utente

Para este predicado utilizamos uma abordagem mais complexa do que aquela que, talvez, era pretendida.

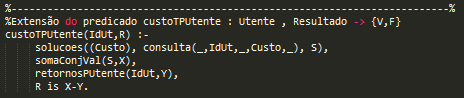


Figura 42 - Extensão do predicado custoTPUtente

Em primeiro lugar, retiramos todos os custos das consultas associados às consultas realizadas por o Utente pretendido.

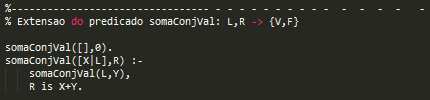


Figura 43 - Extensão do predicado someaConjVal

De seguida somamos esses valores todos através do predicado “somaConjVal” e guardamos o valor numa variável ‘X’.

O próximo passo consiste em utilizar o predicado “retornosPUtente”. Alguns utentes poderão ser reembolsados com uma parte do valor das consultas de acordo com o seguro que estes detêm.

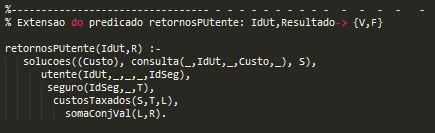


Figura 44 - Extensão do predicado retornosPUtente

Este predicado guarda todos os custos associados a um utente numa lista “S” e encontra o seguro deste. Após esta fase, cria-se uma nova lista através do predicado “custosTaxados” que utiliza a taxa do seguro do utilizador como multiplicador de cada custo presente na lista “S”. No final soma todos os valores da nova lista.

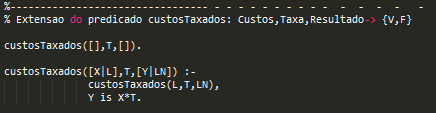


Figura 45 - Extensão do predicado custosTaxados

Após todo este processo do cálculo dos retornos dum utente, guardamos o valor desse cálculo numa variável “Y”.

Por fim o resultado será a subtração da variável X (custos totais) pela variável Y (custos taxados).

Para o utente com ID1 foram realizadas duas consultas, uma com um custo associado de 25 e outra com um custo associado de 10 logo os custos das consultas é 25+10=35. O retorno será 25\*0.4+10\*0.4=14, sendo que a taxa associada ao utente é de 0.4 (ADSE). Podemos verificar o resultado:

custoTPUtenteS.png

Figura 46 - Exemplo de teste ao predicado custoTPUtente

## 2.8.2 - Custo total dos cuidados de saúde por data/serviço/médico

Nestes dois predicados vamos a todas as consultas que tenham sido realizadas na data pretendida ou pelo serviço pretendido ou pelo devido medico e retiramos todos os custos destas.

Depois, simplesmente, somamos esses valores.

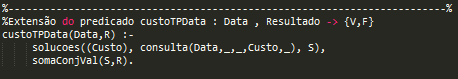


Figura 47 - Extensão do predicado custoTPData

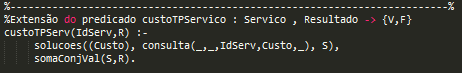


Figura 48 - Extensão do predicado custoTPServ

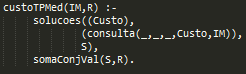


Figura 49 - Extensão do predicado custoTPMed

Exemplos:

custoTPDataS.png

Figura 50 - Exemplo de teste ao predicado custoTPData

custoTPServicoS.png

Figura 51 - Exemplo de teste ao predicado custoTPServ

## 2.8.3 – Custo total dos cuidados de saúde por Instituição

Aqui verificamos os serviços que são realizados na instituição pretendida e usamos o ID destes para encontrar todas as consultas realizadas por estes. De seguida retiramos os custos destas consultas e somamos tudo.

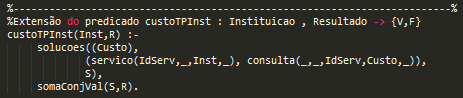


Figura 52 - Extensão do predicado custoTPInst

Exemplo:

custoTPInstS.png

Figura 53 - Exemplo de teste ao predicado custoTPInst

## 2.9 – Guardar e carregar factos através da utilização de um ficheiro de texto

## Aquando da realização de testes com o *sicstus Prolog* reparámos que todo o conhecimento que retirávamos ou colocávamos era apenas conhecimento que se mantinha em tempo de execução. No fim da execução do interpretador todas as alterações que tínhamos feito eram apagadas.

## Desta forma achamos que podia ser conveniente guardar de algum modo todos os factos referentes aos diversos predicados, para que conseguíssemos realizar testes com uma maior quantidade de dados, e de facto a nossa base de conhecimento evoluir sem que fosse perdida nenhuma informação.

## Apoiado nesta ideia o grupo desenvolveu dois novos predicados. O primeiro fica encarregue de guardar em ficheiro todos os factos existentes na base de conhecimento até ao momento.

* guardaFactos: Ficheiro -> {V,F}

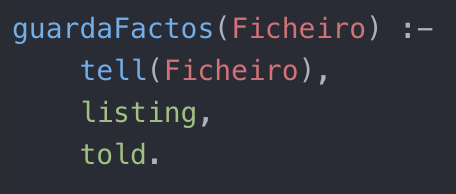


Figura 54 - Extensão do predicado guardaFactos

Neste predicado usamos dois predicados *built-in* do PROLOG, *tell*/1 e *told*/0. O primeiro predicado abre o Ficheiro e torna-o o atual output. Depois de escrever os dados do predicado *listing* no Ficheiro, realiza-se o predicado *told* que fecha o atual output (Ficheiro).

Ora possuindo um predicado que guarda todos os factos em ficheiro necessitámos doutro predicado que carregue esses factos do ficheiro onde se encontram guardados.

* carregaFactos: Ficheiro -> {V,F}

Uma imagem com texto

Descrição gerada automaticamente

Figura 55 - Extensão do predicado carregaFactos

Para este predicado utilizamos 5 predicados *built-in* do PROLOG. O primeiro, *seeing/1*, guarda em InputAtual o input atual do interpretador. O segundo, *see/1*, torna o Ficheiro o atual input do interpretador. O terceiro, *repeat/0*, é utilizado frequentemente para reproduzir ciclos de falha, neste caso para ler termos do input atual do interpretador em ciclo até que seja atingido *end\_of\_file*. O predicado utilizado para ler termos foi o *read/1*. Por último utilizamos o predicado *seen/0*, que fecha o input atual do interpretador.

Para finalizar é utlizado novamente o predicado *see* para tornar o input atual o input anterior à realização do predicado *carregaFactos*, que ficou guardado em InputAtual.

Para dar uso a estes dois predicados junto com o relatório e com o ficheiro relativo ao exercício enviamos um ficheiro *GRUPO10\_FACTOS.txt* onde residem alguns factos que podem ser usados para testar os predicados.

## 2.10 – Predicados auxiliares

Foram criados alguns predicados auxiliares (alguns já apresentados anteriormente) para simplificar a realização do exercício proposto. Para que seja possível entender a utilização destes em alguns predicados, apresentamos uma imagem ilustrativa de todos os predicados que ainda não foram representados nas secções anteriores.

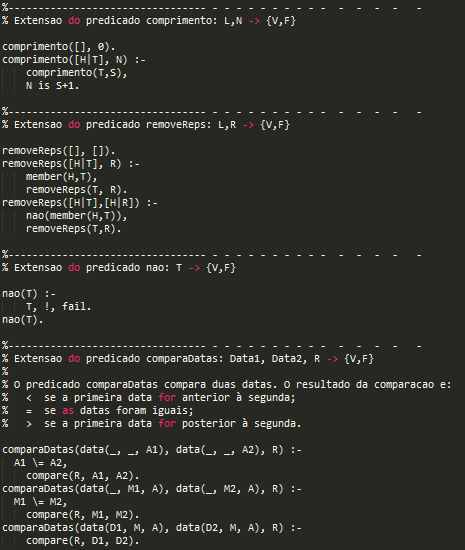


Figura 56 - Extensões de predicados auxiliares

# 3 – Conclusão

O grupo considera que realizou um trabalho bastante aprofundado tendo respondido às questões propostas no enunciado do exercício de uma forma simples e correta. O facto de termos criado um número significativo de funcionalidades extra faz-nos acreditar que DEVIAMOS TER 20 CARALHO.

Como trabalho futuro alguns aspetos no que conta à implementação do sistema de seguros poderiam ser melhorados criando por exemplo situações de utentes isentos de pagamento.

De uma forma geral, sentimos que os conceitos de sistemas de representação de conhecimento e raciocínio pedidos neste primeiro exercício ficaram bem consolidados e solidificamos a nossa capacidade de utilizar a linguagem PROLOG.