

Diogo Braga

Diogo Rocha

João Silva

Ricardo Caçador

Ricardo Veloso

**2º Exercício do Trabalho de Grupo:**

**Programação em Lógica Estendida e Conhecimento Imperfeito**

**Universidade do Minho**

Departamento de Informática

**2º Exercício do Trabalho de Grupo:**

**Programação em Lógica Estendida e Conhecimento Imperfeito**

Diogo Braga

Diogo Rocha

João Silva

Ricardo Caçador

Ricardo Veloso

**Universidade do Minho**

Departamento de Informática



# Resumo

O seguinte relatório consiste na apresentação do trabalho realizado no âmbito da Unidade Curricular de Sistemas de Representação de Conhecimento e Raciocínio relativo ao segundo exercício requerido, e tem como objetivo a continuação do desenvolvimento de competências na linguagem de programação em lógica PROLOG.

O exercício referido acima consiste num sistema de representação de conhecimento imperfeito com capacidade para ­caracterizar um universo de discurso na área da prestação de cuidados de saúde pela realização de serviços de atos médicos, recorrendo à temática de valores nulos.

Ao longo do presente relatório irá ser explicado todo o processo e todos os mecanismos necessário para dar resposta aos desafios propostos no enunciado desta segunda fase.

**Tabela de Conteúdos**

[Resumo 3](#_Toc6833418)

[1 – Introdução 6](#_Toc6833419)

[2 - Descrição do Trabalho 7](#_Toc6833420)

[2.1 - Representação de Conhecimento Positivo 7](#_Toc6833421)

[2.2 – Representação de Conhecimento Negativo 8](#_Toc6833422)

[2.3 – Representação Conhecimento Imperfeito 10](#_Toc6833423)

[2.3.1 – Conhecimento Imperfeito Incerto 10](#_Toc6833424)

[2.3.2 – Conhecimento Imperfeito Impreciso 10](#_Toc6833425)

[2.3.3 – Conhecimento Imperfeito Interdito 11](#_Toc6833426)

[2.4 – Invariantes de inserção e remoção de conhecimento 12](#_Toc6833427)

[2.5 – Evolução e regressão do conhecimento 14](#_Toc6833428)

[2.6 – Sistema de Inferência 15](#_Toc6833429)

[3 – Conclusão 16](#_Toc6833430)

[------- POR FAZER ------- 16](#_Toc6833431)

**Tabela de Figuras**

# 1 – Introdução

Muito à semelhança da primeira fase, para esta segunda fase foi-nos proposta 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 tendo uma atenção redobrada no que conta ao tema da representação de conhecimento imperfeito.

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

# 2 - Descrição do Trabalho

Devido à ligação que existe com a primeira fase do projeto, é importante relembrar os predicados que foram criados:

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

Tendo em conta a Programação em Lógica Estendida abordada nesta fase, é importante diferenciar que, em complemento com um sistema de inferência, agora os predicados podem resultar num valor de Verdadeiro, Falso ou Desconhecido.

## 2.1 - Representação de Conhecimento Positivo

A representação de conhecimento positivo foi replicada da primeira fase, tendo em conta que já nesse momento tinha existido este tipo de representação de conhecimento.

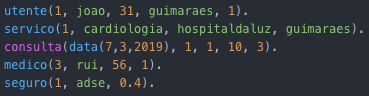
Desta forma, a seguir apresentamos um exemplo de conhecimento positivo, para cada um dos predicados estabelecidos no projeto:

Figura 1 – Exemplos de Conhecimento Positivo

## 2.2 – Representação de Conhecimento Negativo

A representação de conhecimento negativo foi algo que, devido aos novos requisitos provenientes da Lógica Estendida, fez sentido criar no trabalho, pois desta forma tornamos possível representar algo que é falso. Tal é possível representar através da negação explícita, como através da negação forte.

**Negação Explícita**

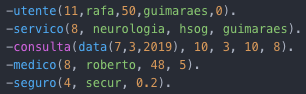
De seguida, são apresentados alguns exemplos de negação explícita dos predicados envolvidos no trabalho, associados à explicação dos mesmos.

Figura 2 – Exemplos de Negação Explícita

Exemplos:

1. Não existe um utente de 50 anos com id 11 e nome rafa, que viva em guimaraes sem seguro.
2. Não existe um serviço de neurologia, com id 8, no hsog em guimaraes.
3. Não existe uma consulta do dia 07/03/2019, do utente com o id 10 para o serviço com o id 3, no qual o custo monetário é 10 e o id do médico é 8.
4. Não existe um médico de ginecologia com o id 8, chamado roberto, e com 48 anos de idade.
5. Não existe um seguro com id 4 e nome secur que possuía uma taxa de retorno de 0.2.

**Negação Forte**

Este género de negação assenta na aplicação do Pressuposto do Mundo Fechado na programação em Lógica Estendida. Com tal aplicação garantimos que algo que não tem prova de ser verdadeiro, é implicitamente falso.

De seguida, é apresentada a extensão do predicado que define a negação forte do predicado utente. As extensões dos restantes predicados são realizadas da mesma forma, só que com o predicado respetivo.



Figura 3- Negação Forte do predicado utente

Esta negação do utente parametrizado na figura é validada caso não exista prova que o utente seja conhecimento verdadeiro, e caso não exista prova duma exceção associada a esse mesmo utente.

## 2.3 – Representação Conhecimento Imperfeito

## 2.3.1 – Conhecimento Imperfeito Incerto

Incerto.pngPara representar conhecimento imperfeito incerto vamos recorrer a um exemplo prático definido pelo grupo:

Figura 4 - Exemplo de conhecimento imperfeito incerto

Podemos verificar que, de facto, existe um serviço com id nº9 correspondente à área da fisiatria e foi realizado em Braga mas, relativamente à instituição onde este foi realizado a informação é desconhecida. Neste caso, foi escolhido “xpto021” no lugar da instituição prestadora do serviço para representar um caso de exceção.

Resumidamente, o conhecimento imperfeito incerto representa algo desconhecido de um conjunto indeterminado de hipóteses.

## 2.3.2 – Conhecimento Imperfeito Impreciso

Para representar conhecimento imperfeito impreciso vamos, tal e qual como no ponto antecedente, recorrer a um exemplo:

impreciso.png

Figura 5 - Exemplo de conhecimento imperfeito impreciso

É possível averiguar que existe um seguro 5 denominado de “Allianz” mas não sabemos a taxa de dedução do valor das consultas, apenas sabemos que se encontra entre 0.2 (20%) e 0.25 (25%).

Portanto, o conhecimento imperfeito impreciso representa algo desconhecido de um conjunto limitado de soluções.

## 2.3.3 – Conhecimento Imperfeito Interdito

Por fim, para representar conhecimento imperfeito interdito vamos também recorrer a um exemplo construído pelo grupo:

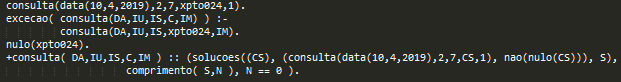


Figura 6 - Exemplo de conhecimento imperfeito interdito

Temos portanto uma consulta realizada no dia 10 de Abril de 2019, pelo médico com id igual a 1 (Ricardo) que presta o serviço do tipo 7 (Oftalmologia) e requisitada pelo utente com id igual a 2 (André). Fora tudo isto, nunca será possível determinar o valor dela.

Para representar este tipo de conhecimento é necessário definir uma exceção (“xpto024”) e atribuir-lhe um valor nulo. Além disso, é também preciso impedir a adição de conhecimento a esta consulta em que o custo seja não nulo.

Finalmente, o conhecimento imperfeito interdito representa algo desconhecido e não permitido conhecer.

## 2.4 – Invariantes de inserção e remoção de conhecimento

Para que o sistema funcione corretamente, foi necessário manipular alguns invariantes que controlam a inserção e remoção de conhecimento.

Não deve ser, por exemplo, permitido inserir um utente com um ID que já esteja associado a outro utente já presente na base do conhecimento.

repetido.png

Figura 7 - Invariante que não permite a inserção de conhecimento repetido

São então procurados utentes com o ID do utente que pretendemos inserir e criámos uma lista com estes. Caso a lista tenha o comprimento igual a 1, podemos inserir o utente na base do conhecimento.

Também não deve ser possível inserir consultas relativas a clientes não existentes.

referencial.png

Figura 8 - Invariante que não permite a inserção de consultas relativas a um utente inexistente

Para que a consulta seja adicionada à base do conhecimento é imperativo que exista um utente com ID respetivo ao ID do utente da consulta.

Outro exemplo de invariante é a impossibilidade de inserir conhecimento negativo repetido.

negativo.png

Figura 9 - Invariante que não permite a inserção de conhecimento negativo repetido

Este invariante tem o mesmo método de processo que o primeiro invariante apresentado.

Relativamente às remoções os invariantes criados foram:

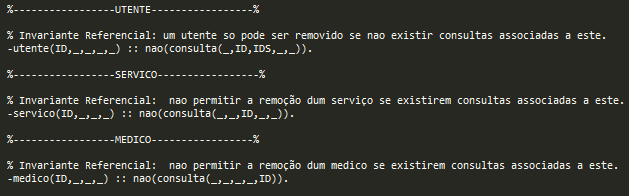


Figura 10 - Invariantes para a remoção de conhecimento

Todos estes predicados não permitem remover conhecimento que esteja associado, de qualquer forma, a outro tipo de conhecimento. Por exemplo, se um utente tiver consultas associadas a ele não podemos removê-lo.

## 2.5 – Evolução e regressão do conhecimento

Uma vez que nesta fase estamos a lidar com conhecimento perfeito e imperfeito, os predicados de inserção e regressão da fase anterior não conseguem evoluir ou regredir a base de conhecimento da forma que se esperaria.

Desta forma, foi imperativo que criássemos predicados capazes de lidar com este novo problema. A nossa proposta de solução para esta problemática passou por dividir cada tipo de conhecimento e criar um predicado para cada um deles.

A explicação que se segue contempla apenas o predicado Utente. A evolução e regressão para os restantes predicados foram assentes nas mesmas ideias.

## 2.5.1 – Evolução de Conhecimento Perfeito

Quando inserimos conhecimento perfeito, precisamos de ter em conta se este é positivo ou negativo, e se existe alguma imprecisão ou incerteza na base de conhecimento associado a este. No caso de ser algum conhecimento interdito a estar associado ao novo conhecimento apenas é necessário verificar os invariantes definidos.

O predicado **evolucaoPerfeito: Utente -> {V,F,D}**, trata da inserção de conhecimento perfeito, tanto positivo como negativo, na base de conhecimento. Este predicado é apresentado a seguir.

Uma imagem com texto

Descrição gerada automaticamente

Figura 11 - Definição do predicado evolucaoPerfeito

Como se pode reparar na imagem acima, o caso em que o conhecimento a adicionar for **verdadeiro**, não é tratado nem admissível uma vez que não é permitido ter conhecimento repetido na base de conhecimento.

Para o caso em que o conhecimento a ser inserido é **desconhecido** ou **falso**, este é tratado da mesma forma. Primeiro é removido todo o conhecimento impreciso e incerto associado este, caso exista, e de seguida é usado o predicado **evolução** proveniente da primeira fase que insere ou não o novo facto. Por último é inserido na base de conhecimento um facto **conhecimentoPerfeito**, que marca através do **ID** do utente o tipo de conhecimento associado a si.

Os predicados auxiliares para contruir o **evolucaoPerfeito** são apresentados a seguir:

Uma imagem com garrafa

Descrição gerada automaticamente

Figura 12 - Definição do predicado auxiliar removerConhecimentoImpreciso

Uma imagem com texto

Descrição gerada automaticamente

Figura 13 - Definição do predicado auxiliar removerConhecimentoIncerto

Os predicados acima definidos removem da base de conhecimento situações de incerteza e imprecisão associado ao facto a ser inserido.

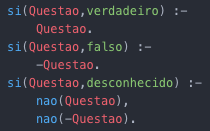
## 2.6 – Sistema de Inferência

Tendo em conta a diferente abordagem desta fase, que tem por base a Programação em Lógica Estendida, foi conclusivo que os predicados agora podem resultar num valor de Verdadeiro, Falso ou Desconhecido, demonstrando assim que os contradomínios dos novos predicados são mais abrangentes.

Devido a tal, foi necessário criar um Sistema de Inferência diferente que fosse capaz de analisar os três tipos de respostas que os novos predicados podem efetuar.

* Verdadeiro, caso a prova seja verdadeira
* Falso, caso a prova seja falsa
* Desconhecido, caso não exista prova

O sistema de inferência analisa, portanto, todas as questões da seguinte forma:



Sendo que o predicado não é definido da seguinte forma:



Figura 14 - Predicado nao

Este predicado tenta procurar uma prova para a questão parametrizada, e caso este falhe, é concluído que não existe prova.

De forma a estarmos perante um Sistema de Inferência mais capacitado, foram desenvolvidas extensões para ser possível responder a mais do que uma questão de cada vez.

As operações lógicas abordadas foram:

* Conjunção
* Disjunção
* Implicação
* Equivalência

Desta forma é possível realizar duas ou mais questões de uma só vez no Sistema de Inferência, caso estas possam ser conjugadas com os operadores lógicos referidos.

Para tal, foi construído o seguinte meta-predicado que :

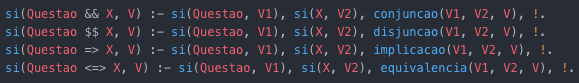


Figura 15 - Meta-predicado Operações Lógicas

Em complemento com este meta-predicado foi estabelecido, para cada um dos valores lógicos, os resultados que estes podem adquirir consoante os valores resultantes dos predicados parametrizados.

# 3 – Conclusão

Este trabalho prático teve como objetivo solidificar os conhecimentos relativos à representação de conhecimento imperfeito, recorrendo à utilização de valores nulos. Foi assim possível entender a utilização destes conhecimentos para casos práticos.

Em termos de desafios, o principal terá sido representar corretamente a evolução de conhecimento perfeito e imperfeito, tendo-se relevado algo complicado.

Em suma, o grupo considera que realizou um bom trabalho e que respondeu de uma forma coesa e correta a todas as funcionalidades requisitadas no enunciado do projeto. Estamos, assim, prontos para avançar para a fase seguinte.