**Universidade do Minho**

**Exercício 2 – Extensão à Programação em Lógica e Conhecimento Imperfeito**

Mestrado Integrado em Engenharia Informática - MiEI

Sistemas de Representação de Conhecimento e Raciocínio

(2º Semestre/2015-2016)

A72223 Gustavo da Costa Gomes

A71223 José Carlos da Silva Brandão Gonçalves

A70443 Tiago João Lopes Carvalhais

Local - Braga

Data - 21/04/2016

**Resumo**

Este documento serve de apoio ao segundo exercício prático da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio, cujo objetivo principal é o de motivar para a utilização da linguagem de programação em lógica - *PROLOG*.

Numa primeira fase, serão explicadas as estratégias definidas para representar a informação pretendida, constituindo assim a base de conhecimento para este trabalho. De seguida, serão abordados todos os objetivos propostos no enunciado, bem como as técnicas utilizadas para os solucionar. Foram também desenvolvidas algumas *queries*, a fim de enriquecer este exercício e, para ajudar, foram utilizados alguns predicados auxiliares, pelo que também será efetuada uma explicação pormenorizada acerca dos mesmos. Para além disso, foi-nos proposta também a interação com o sistema, utilizando para tal, uma janela desenvolvida em *JAVA*, que iremos também descrever neste relatório.

Num fase mais terminal deste relatório, serão ainda colocados exemplos de como questionar a base de conhecimento acerca das funcionalidades criadas, bem como os resultados produzidos pela mesma na janela desenvolvida.

**Índice**

**1.** **Introdução** 4

**2.** **Preliminares** 5

**3.** **Código *PROLOG*** 6

**3.1.** **Base de Conhecimento e Representação** 6

**3.2.** **Exemplos de Conhecimento Perfeito** 8

**3.3.** **Exemplos de Conhecimento Negativo** 8

**3.4.** **Exemplos de Conhecimento Imperfeito Incerto** 9

**3.5.** **Exemplos de Conhecimento Imperfeito Impreciso** 10

**3.6.** **Exemplos de Conhecimento Imperfeito Interdito** 11

**3.7.** **Resolução de *Queries*** 12

**3.8.** **Inserção e Remoção de Conhecimento** 13

# 

# **1. Introdução**

O objetivo deste trabalho é a utilização da linguagem de programação em lógica –*PROLOG* - para representar conhecimento perfeito e imperfeito e criar mecanismos que atuem sobre o mesmo, efetuando uma interação com outra linguagem e paradigma de programação.

O conhecimento perfeito caracteriza-se por representar factos certos, ou seja, não deixa dúvidas sobre a veracidade de uma afirmação. Já o conhecimento imperfeito deixa algumas incertezas, isto é, não é seguro que uma dada afirmação seja verdadeira ou falsa, pelo que se introduz um terceiro valor de verdade, que se junta ao verdadeiro e ao falso: o desconhecido. Este tipo de conhecimento possui três variantes, que passamos a explicar de seguida.

O conhecimento imperfeito do tipo incerto ocorre quando não se sabe determinado facto, mas esse pode vir a ser descoberto, sendo que, enquanto não o for, qualquer resposta pode ser verdadeira ou falsa, portanto, será representada como desconhecida. Por exemplo, o Tiago recebeu de salário um valor que ainda ninguém conhece, com exceção do próprio, pelo que poderá revelar a quantia que recebeu assim que o desejar, sendo que, até lá, esse valor permanece desconhecido.

O conhecimento imperfeito do tipo impreciso acontece numa situação em que a dúvida incide num determinado intervalo de incerteza. Por exemplo, não se sabe quanto dinheiro recebeu o José, mas sabe-se que foi um valor superior a 500€ e inferior a 600€. Qualquer valor que esteja incluído dentro deste intervalo será representado como desconhecido e qualquer outro valor será falso.

Por fim, o conhecimento imperfeito do tipo interdito é similar à primeira variante, contudo, neste tipo de situações, é impossível vir-se a esclarecer a dúvida existente, pelo que esta permanecerá permanentemente uma incógnita. A título de exemplo, o Gustavo transportava uma carteira com uma elevada quantidade monetária, que nem o próprio sabia ao certo o seu valor, contudo, esta cai dentro de uma sarjeta e o dinheiro desaparece, sendo que, se ninguém sabia quanto havia na mesma, esse valor será impossível de saber.

Ao longo deste trabalho prático, irão surgir vários exemplos destes tipos de conhecimento exemplificados acima, adaptados à temática sugerida pela equipa docente da unidade curricular.

# **2. Preliminares**

Posto isto, passemos, então, a um breve resumo sobre a temática do segundo exercício do trabalho prático. Mais uma vez e, à semelhança do que foi o primeiro exercício, o tema incide no registo de eventos em instituições de saúde. Contrariamente à fase anterior, em que foi o grupo a decidir que predicados apresentar e como os representar, foi a equipa docente a definir os mesmos e a maneira como são representados. Sendo assim, iremos desde já passar a uma breve explicação sobre estes.

Um utente é composto por um identificador, um nome, uma idade e uma morada, sendo que o primeiro deverá ser único, ou seja, não poderá existir mais nenhum utente com o mesmo número, podendo contudo possuir o mesmo nome, a mesma idade e até a mesma morada.

Um serviço é constituído por um identificador, uma designação, uma instituição e uma cidade. À semelhança do utente, também um serviço deve possuir um identificador único, podendo os outros parâmetros ser iguais.

Por fim, a consulta possui uma data, um número do utente que a efetuou, um identificador do respetivo serviço em que se insere e um custo. Contrariamente aos dois predicados anteriores, uma consulta, como não possui um identificador próprio, a sua unicidade terá em conta os atributos da data, número de utente e número de serviço, pelo que, nenhuma consulta pode ter a mesma data, utente e serviço ao mesmo tempo, sendo o custo irrelevante neste caso.

O grupo decidiu incluir no código do trabalho prático, dois exemplos de conhecimento perfeito, conhecimento imperfeito incerto, impreciso e interdito e um exemplo de conhecimento negativo, para cada um dos predicados. Para se efetuarem questões ao sistema, a partir de agora, deve-se utilizar a função *demo*, que será explicada mais à frente.

**3. Código *PROLOG***

**3.1. Base de Conhecimento e Representação**

Para a representação dos vários tipos de conhecimento existentes neste trabalho prático, foram criados na base de conhecimento factos para utentes, serviços e consultas. Como já foi dito na secção anterior, um utente é caracterizado pelo seu identificador, nome, idade e morada, estando escrito da seguinte forma:

utente(1, manuel\_faria, 24, rua\_das\_papoilas).

Ao observarmos este facto, podemos referir que o utente número 1 tem o nome de manuel\_faria, tem 24 anos e reside na rua\_das\_papoilas.

Já um serviço requer um identificador, designação, instituição e cidade, sendo representado da seguinte forma:

servico(1, oncologia, hospital\_braga, braga).

Desta forma, podemos afirmar que o serviço número 1 é de oncologia e existe na instituição hospital\_braga, na cidade de braga.

Por fim, uma consulta possui uma data, número de utente, identificador de serviço e custo, sendo representada desta forma:

consulta(18-4-2015, 1, 1, 25).

Podemos, então, referir que esta consulta foi efetuada na data de 18-4-2015, pelo utente número 1, no serviço 1 e custou 25 unidades monetárias.

Contudo, deparamo-nos agora com um problema: como representar conhecimento desconhecido? Deixamos, neste momento, de estar no pressuposto do mundo fechado, onde o conhecimento ou é verdadeiro ou falso, e entramos num pressuposto de mundo aberto, com a introdução de um terceiro valor de verdade: o desconhecido. Iremos agora necessitar de colocar questões ao sistema, de modo a que o conjunto de respostas englobe estes três valores de verdade, pelo que tivemos de definir uma nova função: a *demo*.

demo(Q, verdadeiro) :- Q.

demo(Q, falso) :- -Q.

demo(Q, desconhecido) :- nao(Q), nao(-Q).

Como podemos verificar, se uma questão tem prova na base de conhecimento, então o valor de verdade será verdadeiro. Caso exista uma prova de que essa questão é falsa, então o valor de verdade retornado será falso. Caso não exista nem uma prova positiva, nem uma prova negativa dessa questão, então o resultado será desconhecido.

Também podemos querer saber se um conjunto de questões são verdadeiras, falsas ou desconhecidas, pelo que optamos pela criação de uma função da família da *demo*, denominada *demoExtendido*.

demoExtendido(Q1 e Q2, R) :- demo(Q1, R1), demoExtendido(Q2, R2),

e(R1, R2 ,R).

demoExtendido(Q1, R1) :- demo(Q1, R1).

O operador e, explicado em anexo, funciona como uma espécie de tabela de verdade que irá verificar se, entre um conjunto de duas questões, o resultado é verdadeiro, falso ou desconhecido. O programa recebe uma questão Q1 e um conjunto de questões Q2 e irá começar por chamar a função *demo* para calcular o valor de verdade da primeira questão e guarda o resultado em R1, sendo que, seguidamente será chamada a própria função de forma recursiva para calcular o valor de verdade do conjunto de questões e no fim juntam-se os dois resultados e determina-se o valor lógico resultante.

Para além disso, devemos também referir quando um predicado deve devolver falso. Essa situação deve ocorrer quando não existe prova de que a questão é verdadeira e de quando não exista uma exceção que indique que um certo valor dessa questão seja nulo, para evitar que em situações em que o resultado deva ser desconhecido, não devolva falso.

-utente(IdU, N, I, M) :- nao(utente(IdU, N, I, M)),

nao(exception(utente(IdU, N, I, M))).

-servico(IdS, D, I, C) :- nao(servico(IdS, D, I, C)),

nao(exception(servico(IdS, D, I, C))).

-consulta(D, IdU, IdS, C) :- nao(consulta(D, IdU, IdS, C)),

nao(exception(consulta(D, IdU, IdS, C))).

**3.2. Exemplos de Conhecimento Perfeito**

Como referirmos anteriormente, o conhecimento perfeito caracteriza-se por ter um grau de certeza de 100%, pelo que é algo que podemos afirmar como sendo verdadeiro. Seguem em baixo todos os predicados que foram inseridos na base de conhecimento e que se enquadram neste tipo de conhecimento.

utente(1, manuel\_faria, 24, rua\_das\_papoilas).

utente(2, carlos\_sousa, 45, rua\_dos\_malmequeres).

servico(1, oncologia, hospital\_braga, braga).

servico(2, pediatria, hospital\_porto, porto).

consulta(18-4-2015, 1, 1, 25).

consulta(25-2-2016, 2, 3, 30).

**3.3. Exemplos de Conhecimento Negativo**

Nos exemplos anteriores demonstramos vários casos de factos positivos, ou seja, conhecimento que podemos afirmar que é verdadeiro, mas também podemos inserir factos negativos, ou seja, dizer claramente que algo é mentira. Para tal, utilizámos um carater de negação no início dos predicados, como está explícito em baixo.

-utente(5, mario\_cardoso, 33, rua\_das\_tristezas).

-servico(6, psiquiatria, hospital\_militar, braga).

-consulta(22-7-2015, 5, 5, 25).

**3.4. Exemplos de Conhecimento Imperfeito Incerto**

Na outra face da moeda temos o conhecimento imperfeito, que é caracterizado pela falta de certeza acerca de um predicado, sendo que o do tipo incerto tem a particularidade de possuir valores desconhecidos, definidos previamente como sendo valores nulos. Exemplos deste tipo estão presentes em baixo.

utente(3, joao\_seabra, xpto1, rua\_da\_alegria).

utente(4, tiago\_barbosa, 37, xpto2).

exception(utente(Id, N, I, M)) :- utente(Id, N, xpto1, M).

exception(utente(Id, N, I, M)) :- utente(Id, N, I, xpto2).

nulo(xpto1).

nulo(xpto2).

Como se pode verificar, no caso do primeiro utente, a idade é desconhecida, sendo esta definida como um valor nulo, de nome xpto1. Como queremos que a função *demo* caracterize este predicado como sendo desconhecido, devemos acrescentar uma exceção que diga que, se for encontrado esse valor nulo, diga que este predicado apresenta como valor de verdade o desconhecido. Seguem em baixo mais exemplos para os restantes predicados, usando o mesmo raciocínio.

servico(3, cirurgia, xpto3, lisboa).

servico(4, radiologia, ipo\_porto, xpto4).

exception(servico(IdS, D, I, C)) :- servico(IdS, D, xpto3, C).

exception(servico(IdS, D, I, C)) :- servico(IdS, D, I, xpto4).

nulo(xpto3).

nulo(xpto4).

consulta(xpto5, 3, 1, 25).

consulta(30-5-2014, 4, 2, xpto6).

exception(consulta(D, IdU, IdS, C)) :- consulta(xpto5, IdU, IdS, C).

exception(consulta(D, IdU, IdS, C)) :- consulta(D, IdU, IdS, xpto6).

nulo(xpto5).

nulo(xpto6).

**3.5. Exemplos de Conhecimento Imperfeito Impreciso**

O conhecimento imperfeito do tipo impreciso é caracterizado pela sua resposta estar contida dentro de um determinado intervalo de incerteza, sendo que qualquer valor dentro do mesmo será representado como desconhecido e qualquer outro como falso. Abaixo seguem alguns exemplos.

exception(servico(5, cardiologia, hospital\_faro, faro)).

exception(servico(5, urologia, hospital\_faro, faro)).

Á semelhança do caso anterior, foram criadas exceções para referir que o serviço número 5, existente na instituição hospital\_faro na cidade de faro. O problema é que não se sabe ao certo qual a designação do serviço, mas sabe-se que, ou é de cardiologia ou de urologia. Ao colocar como exceção, o sistema irá reconhecer esta questão como desconhecida, desde que as designações sejam aquelas que foram referidas, caso contrário irá devolver como resposta o valor falso.

-servico(6, psiquiatria, hospital\_militar, braga).

exception(servico(6, psiquiatria, hospital\_militar, lisboa)).

exception(servico(6, psiquiatria, hospital\_militar, porto)).

No exemplo acima, verificamos que o serviço número 6 é de psiquiatria e é efetuado na instituição hospital\_militar, contudo não se sabe exatamente a cidade onde se situa a mesma, mas sabe-se que não é em braga, podendo estar situada em lisboa ou no porto.

Seguem em baixo exemplos para os restantes casos.

-utente(5, mario\_cardoso, 33, rua\_das\_tristezas).

exception(utente(5, mario\_cardoso, 34, rua\_das\_tristezas)).

exception(utente(5, mario\_cardoso, 35, rua\_das\_tristezas)).

exception(utente(6, joel\_vaz, 56, rua\_das\_estrelas)).

exception(utente(6, joel\_vaz, 56, rua\_das\_luas)).

-consulta(22-7-2015, 5, 5, 25).

exception(consulta(20-7-2015, 5, 5, 25)).

exception(consulta(21-7-2015, 5, 5, 25)).

exception(consulta(9-9-2013, 6, 2, 30)).

exception(consulta(9-9-2013, 6, 2, 20)).

**3.6. Exemplos de Conhecimento Imperfeito Interdito**

Este tipo de conhecimento caracteriza-se de forma semelhante ao incerto, pois há um certo atributo que é desconhecido, contudo este nunca será conhecido, pelo que é necessário que o sistema garanta que não será possível a inserção de um valor para esse atributo, como está exemplificado em baixo.

exception(consulta(7-12-2015, 4, 3, C)).

exception(consulta(D, 1, 4, 10)).

+consulta(7-12-2015, 4, 3, C) :: (findall((7-12-2015, 4, 3, C),

consulta(7-12-2015, 4, 3, C), S),

length(S, T), T == 0).

+consulta(D, 1, 4, 10) :: (findall((D, 1, 4, 10),

consulta(D, 1, 4, 10), S),

length(S, T), T == 0).

Como podemos verificar, foram criadas mais exceções, para garantir que o predicado inserido seja reconhecido como desconhecido. Contudo, para garantir que a informação desconhecida não seja atualizada com inserção de conhecimento, foram criados invariantes para garantir que, ao inserir o mesmo predicado, mas com um valor certo, neste caso, para o custo da consulta, este não seja inserido na base de conhecimento e se mantenha a incerteza.

Em baixo seguem os restantes exemplos.

exception(utente(7, jose\_esteves, I, rua\_das\_alcatifas)).

exception(utente(8, otavio\_correia, 79, M)).

+utente(7, jose\_esteves, I, rua\_das\_alcatifas) :: (findall((7,

jose\_esteves, I, rua\_das\_alcatifas), utente(7, jose\_esteves, I, rua\_das\_alcatifas), R), length(R, T), T == 0).

+utente(8, otavio\_correia, 79, M) :: (findall((8, otavio\_correia, 79, M), utente(8, otavio\_correia, 79, M), R), length(R, T), T == 0).

exception(servico(7, psicologia, hospital\_amadora\_sintra, C)).

exception(servico(IdS, oftalmologia, hospital\_sao\_joao, porto)).

+servico(7, psicologia, hospital\_amadora\_sintra, C) :: (findall((7, psicologia, hospital\_amadora\_sintra, C), servico(7, psicologia,

hospital\_amadora\_sintra, C), S), length(S, T), T == 0).

+servico(IdS, oftalmologia, hospital\_sao\_joao, porto) :: (findall((IdS, oftalmologia, hospital\_sao\_joao, porto), servico(IdS, oftalmologia, hospital\_sao\_joao, porto), S), length(S, T), T == 0).

**3.7. Resolução de *Queries***

Procedemos também à realização de três *queries*, um pouco à semelhança do primeiro exercício, a fim de enriquecer um pouco o trabalho realizado e para o utilizador da interface poder interagir com o sistema para poder consultar alguns dados estatísticos. As três funções estão representadas em baixo.

mediaIdades(R) :- findall(I, utente(IdU, N, I, M), S), media(S, R).

A função acima calcula a média de idades de todos os utentes. Começa por encontrar uma todos os que estão registados na base de conhecimento e, de seguida, constrói uma lista com as idades dos mesmos e calcula a média desses valores. Convém notar que as idades só irão ser contabilizadas na contagem, apenas no caso de serem valores não nulos, ou seja, valores que sejam conhecidos.

servicosInstituicao(I, R) :- findall((IdS, D),

servico(IdS, D, I, C), R).

Esta função determina a lista de serviços que são efetuados numa dada instituição, sendo que o programa irá construir uma lista com todos os serviços registados e devolvê-la como *output*.

getConsultas(IdU, Lc, Tp) :- findall((D, C),

consulta(D, IdU, IdS, C), S),

filtraConsultas(S, Lc, Tp).

A função acima tem como função devolver uma lista com as consultas de um utilizador e o custo total das mesmas. Começa por determinar uma lista com as consultas registadas com o número do utilizador e depois chama a função filtraConsultas, que irá filtrar as consultas do utente que não possuem um valor de custo ou data desconhecidos.

**3.8. Inserção e Remoção de Conhecimento**

Como restrições à inserção e remoção de informação na base de conhecimento, decidimos proceder à criação de vários invariantes que nos garantam que as seguintes condições são verificadas:

* Não existir mais do que um utente com o mesmo identificador;
* Não existir mais do que um serviço com o mesmo identificador;
* Não existir mais do que uma consulta com a mesma data, número de cliente e número de serviço;
* Ao inserir uma consulta com um número de utente e serviço, estes já devem estar representados na base de conhecimento;
* Para se remover um serviço, este não deverá possuir consultas associadas;
* Para se remover um utente, este não deverá possuir consultas associadas;
* Não é permitido adicionar exceções repetidas.

Mediante estas condições, procedemos, então, à criação dos seguintes invariantes:

+utente(IdU, N, I, M) :: (findall((IdU, N2, I2, M2),

utente(IdU, N2, I2, M2), S),

length(S, T), T == 1).

+servico(IdS, D, I, C) :: (findall((IdS, D2, I2, C2),

servico(IdS, D2, I2, C2), S),

length(S, T), T == 1).

+consulta(D, IdU, IdS, C) :: (findall((D, IdU, IdS, C2),

consulta(D, IdU, IdS, C2), S),

length(S, T), T == 1).

+consulta(D, IdU, IdS, C) :: (findall(IdU, utente(IdU, N, I, M), S),

length(S, T), T == 1).

+consulta(D, IdU, IdS, C) :: (findall(IdS,

servico(IdS, Descr, I, Cid), S),

length(S, T), T == 1).

-servico(IdS, D, I, C) :: (findall(IdS,

consulta(Data, IdU, IdS, Custo), S),

length(S, T), T == 0).

-utente(IdU, N, I, M) :: (findall(IdU,

consulta(Data, IdU, IdS, Custo), S),

length(S, T), T == 0).

+exception(Q) :: (findall(Q, exception(Q), S), length(S, T), T == 1).