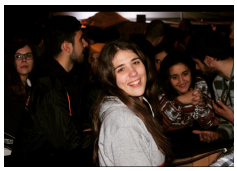
# Sistemas de Representação de Conhecimento e Raciocínio

MIEI - 3º ano - 2º semestre
Universidade do Minho

## Trabalho Prático



Sara Pereira
A73700


Rui Vieira
A74658


Filipe Fortunato
A75008


Frederico Pinto
A73639


y

# *Conteúdo*

**1 Introdução** **2**

**2 Introdução** **3**

**3 Introdução** **4**
3.1 Análise Teórica . . . . . 5
3.2 Mundo Aberto . . . . . 5
3.2.1 Conhecimento Imperfeito . . . . . 5
3.3 Lógica ternária . . . . . 6
3.4 Evolução do Conhecimento . . . . . 6
3.5 Implementação da Solução . . . . . 6
3.5.1 Representação do Conhecimento Imperfeito . . . . . 7
3.5.2 Evolução de Conhecimento Imperfeito . . . . . 9
3.6 Conclusão . . . . . 10

# 1. *Introdução*

Na primera fase deste trabalho foi-nos proposto o desenvolvimento de uma base de conhecimento de uma instituição médica. Posto isto, esta base de dados apenas representava conhecimento concreto, sendo impossível a representação de conhecimento imperfeito. Nesta segunda fase partimos deste princípio para a resolução deste problema. No final, é esperado que a base de conhecimento seja capaz de representar conhecimento incerto, impreciso e interdito.

# 2. *Introdução*

Na primera fase deste trabalho foi-nos proposto o desenvolvimento de uma base de conhecimento de uma instituição médica. Posto isto, esta base de dados apenas representava conhecimento concreto, sendo impossível a representação de conhecimento imperfeito. Nesta segunda fase partimos deste princípio para a resolução deste problema. No final, é esperado que a base de conhecimento seja capaz de representar conhecimento incerto, impreciso e interdito.

# 3. *Introdução*

Na primera fase deste trabalho foi-nos proposto o desenvolvimento de uma base de conhecimento de uma instituição médica. Posto isto, esta base de dados apenas representava conhecimento concreto, sendo impossível a representação de conhecimento imperfeito. Nesta segunda fase partimos deste princípio para a resolução deste problema. No final, é esperado que a base de conhecimento seja capaz de representar conhecimento incerto, impreciso e interdito.

## 3.1 Análise Teórica

No exercício anterior a base de conhecimento assenta no **Pressuposto do Mundo Fechado**, i.e, toda a informação que não existe mencionada na base de dados é considerada falsa. Mas, na realidade não é isto que acontece, não é considerada falsa uma informação que não se conhece, mas sim desconhecida. Nem faria sentido assumir que as entidades presentes na base de dados sejam as únicas existentes. Desta maneira, seguimos este princípio para o desencvolvimento da base de conhecimento atual. Como diz o **Pressuposto do Mundo Aberto** podem existir outros factos ou conclusões verdadeiros para além daqueles representados na base de conhecimento.

## 3.2 Mundo Aberto

Partindo deste pressuposto, vamos fazer uma distinção entre dois tipos de negação: forte e por falha. Assim, entende-se por negação por falha (representada pelo termo **nao**) algo que não esteja presente na base de conhecimento e negação forte (representada por ¬) pela identificação de informação falsa. Portanto, a partir deste princípio, para uma certa questão **q** podem ser obtidas 3 respostas diferentes:

- `verdadeiro se` $\exists$`X : q(X)`
- `falso se` $\exists$`X :` $\neg$`q(X)`
- `desconhecido se` $\neg\exists$`X : q(X)` $\vee$ $\neg$`q(X)`

### 3.2.1 Conhecimento Imperfeito

Este tipo de conhecimento depende das possíveis respostas a uma questão, mais concretamente no caso de a mesma ser **desconhecida**. Logo, existem 3 tipos de conhecimento imperfeito:

- **Incerto** - representa valores nulos que não pertencem a nenhum conjunto determinado de valores, sendo portanto completamente desconhecidos.
- **Impreciso** - representa valores nulos que pertencem a um conjunto determinado de valores, i.e, existe uma noção do que é falso mas não se conhece qual a verdadeira resposta.
- **Interdito** - representa valores nulos desconhecidos e que não serão permitidos conhecer.

Ao mesmo tempo, pode ser necessária a aplicação do pressuposto do mundo fechado em programação estendida. Isto acontece quando estamos perante uma situação a identificar na definição dos casos que são considerados extraordinários à concretização da informação negativa, i.e:

$\neg p(x) \leftarrow \exists x : p(x) \vee excecao(p(x))$

Como não há regra sem exceção, pode existir o caso de haver um predicado que seja falso na maior parte das situações, havendo exceções em que não existe prova de que seja falso.

## 3.3 Lógica ternária

Com a adição do novo pressuposto à lógica anterior foi necessário criar inferências de maneira a aumentar o número de valores de verdade, passando a ser estes: **verdadeiro, falso, desconhecido**.

| $\wedge$ | **V** | **F** | **D** |
|---|---|---|---|
| **V** | V | F | D |
| **F** | F | F | D |
| **D** | D | D | D |

Tabela 3.1: Tabela da conjunção de valores

| $\vee$ | **V** | **F** | **D** |
|---|---|---|---|
| **V** | V | V | D |
| **F** | V | F | D |
| **D** | D | D | D |

Tabela 3.2: Tabela da conjunção de valores

## 3.4 Evolução do Conhecimento

Este tópico retrata parte da evolução da representação do conhecimento, visto que, anteriormente, tratávamos de casos de conhecimento perfeito e , agora, deparamo-nos com a adaptação da antiga base de conhecimento para esta conseguir retratar conhecimento imperfeito. Este processo consistia na criação de invariantes de maneira a garantir a consistência da base de dados em conjunto com os predicados **evolução** e **involução** que garantiam que nada do que era inserido violava os mesmos invariantes. Agora, perante a lógica extendida e o **pressuposto do mundo aberto**, recorrendo à temática dos três tipos de conhecimento imperfeito, foram adicionados valores nulos, à base de conhecimento, que podem ser representados por exceções. Ao mesmo tempo, também, foram criados invariantes que garantam que o conhecimento não pode evoluir.

## 3.5 Implementação da Solução

Como já foi referido anteriormente, e com a adição de conhecimento imperfeito, passamos da lógica clássica à lógica extendida, onde para além das respostas **verdadeiro** ou **falso** podemos deparar-nos com o **desconhecido**. Assim, foi desenvolvido, com base na lógica clássica um interpretador de questões, mas antes disso foi preciso definir os três tipos de respostas:

**Verdadeiro** : $\exists x : p(x)$
**Falso** : $\exists x : \neg p(x)$
**Desconhecido**$\neg \exists x : p(x) \vee \neg p(x)$

Logo, o interpretador de questões trata-se de um predicado que se baseia na informação acima para associar respostas a questões:

```
demo(Q,verdadeiro) :- Q.
demo(Q,falso) :- -Q.
demo(Q,desconhecido) :- nao(Q), nao(-Q).
```

Este predicado é capaz de avaliar uma única questão de cada vez, por isso o grupo achou necessária a criação de predicados que permitissem a avaliação de múltiplas questões, quer pela conjunção, quer pela disjunção das mesmas. Assim, foram criados os seguintes predicados:

```
demoConj([],verdadeiro).
demoConj([X],R) :- demo(X,R).
```

$demoConj([Q1\|Q2], R) : -demo(Q1, R1), demoConj(Q2, R2), conjuncao(R1, R2, R).$

$demoDisj([], falso).$
$demoDisj([X], R) : -demo(X, R).$
$demoDisj([Q1\|Q2], R) : -demo(Q1, R1), demoDisj(Q2, R2), disjuncao(R1, R2, R).$

Como se verifica, estes dois predicados são auxiliados por predicados como o *conjuncao* e *disjuncao*, respetivamente, que, por sua vez usam operadores de lógica ternária, definidos por nós, sendo o **$$** referente à conjunção ternária e o // referente à disjunção ternária. Também foi definido o operador **equals** que é equivalente ao operador **is**.

```
conjuncao(X,Y,R) :- R equals X $$ Y.
disjuncao(X,Y,R) :- R equals X // Y.
```

### 3.5.1 Representação do Conhecimento Imperfeito

Para a negação de um predicado seguimos a lógica referida anteriormente:

$\neg p(x) \leftarrow \exists x : p(x) \vee excecao(p(x))$

E definimos predicados que se baseiam na mesma para a representação da negação dos predicados utente, prestador, cuidado, instituição e data. Este último predicado é novo e foi desenvolvido com o intuito de se poder acrescentar este novo tipo de conhecimento em relação à data de um cuidado de saúde. Assim, um cuidado possui um identificador de uma certa data que é formada, portanto:

- **data** : IdData, Ano, Mes, Dia -> {V,F}

Posteriormente vai ser explicado como foi acrescentado conhecimento imperfeito em relação a este novo parâmetro.

#### Conhecimento Incerto

Para a representação deste tipo de conhecimento tomamos partido da definição acima e do predicado **excecao**. Para a declaração de conhecimento incerto, primeiramente

declaramos p(x) e, de seguida, é necessário declarar a exceção de p(x). Como por exemplo, para registar um utente cuja morada é desconhecida:

utente(16,'Joaquina',57,morada_desconhecida).
excecao(utente(Idu,N,Idd,_)) :- utente(Idu,N,Idd,morada_desconhecida).

### Conhecimento Impreciso

Para a representação deste tipo de conhecimento foi desenvolvido um predicado que se achou vir a ser útil para distinguir entre que intervalo de valores informação sobre a idade desconhecida de um utente é desconhecida ou falsa, sendo este o **aproximadamente**:

aproximadamente(X,Y) :- W is 0.85 * Y, Z is 1.15 * Y, X >= W, X =< Z.

Tirando, novamente, partido do predicado **excecao**, começamos por definir um utente com qualquer parâmetro desconhecido. Por exemplo, um utente do qual não se conhece a idade mas sabe-se que é, aproximadamente, 50 pode ser definido da seguinte maneira:

utente(19,'Carmo',idade_desconhecida,'Rua dos Pinheiros').

excecao(utente(19,'Carmo',Idd,'Rua dos Pinheiros')) :- aproximadamente(Idd,50).

Para a representação do desconhecimento da idade de um utente mas sabendo que não é 20 anos:

utente(18,'Jonas',idade_desconhecida,'Rua de Cristal').
-utente(18,'Jonas',20,'Rua de Cristal').

### Conhecimento Interdito

Como sabemos, conhecimento interdito é o que não se sabe qualquer tipo de informação e está destinado a nunca se saber, por isso é necessário garantir que este nunca evolui. Para proibir a expansão desse conhecimento é utilizado o predicado **nulo** em conjunto com a sua utilização em invariantes. Assim, a representação de um cuidado de saúde que possui uma prioridade interdita pode ser representado:

nulo(prioridade_interdita).

cuidado(20,10,9,prioridade_interdita,'AVC',20).

+**cuidado**(Data,Idu,Idp,Prio,Desc,Custo) :: (solucoes((Data,Idu,Idp,Prio,Desc,Custo), (cuidado(20,10,9,prioridade_interdita,'AVC',20), nao(nulo(prioridade_interdita))), R), comprimento(R,N), N==0).

### 3.5.2 Evolução de Conhecimento Imperfeito

Acriação deste tópico foi devida ao facto do nosso sistema ser capaz de evolui conhecimento imperfeito para perfeito. Isto acontece porque é substituido um facto que possui valor deesconhecido por outro com um valor concreto, mantendo os outros parâmetros inalterados e sendo sempre validados pelos invariantes. Assim, foram criados predicados de maneira a tudo isto ser possível, sendo estes:

- **preencher_utente_idade(Id, Idade)**
- **preencher_utente_morada(Id, Morada)**
- **preencher_prestador_inst(Idp, Inst)**
- **preencher_Data(IdD,Dia)**

Como os nomes indicam, estes predicados substituem os valores desconhecidos dos parâmetros idade de um utente, morada de um utente, instituição de um prestador e dia de uma data, respetivamente, por valores conhecidos.

```
?- utenteID(18,R).
R = utente(18, "Jonas", idade_desconhecida, "Rua de Cristal").

?- preencher_utente_idade(18,15).
true .

?- utenteID(18,R).
R = utente(18, "Jonas", 15, "Rua de Cristal").

?- demo(utente(18, "Jonas", 15, "Rua de Cristal"),R).
R = verdadeiro .
```

```
?- utenteID(16,R).
R = utente(16, "Joaquina", 57, morada_desconhecida).

?- preencher_utente_morada(16,"Rua das Caminhadas").
true .

?- utenteID(16,R).
R = utente(16, "Joaquina", 57, "Rua das Caminhadas").

?- demo(utente(16, "Joaquina", 57, "Rua das Caminhadas"),R).
R = verdadeiro .
```

```
?- prestadorID(21,R).
R = prestador(21, "Amaral", "Optometria", inst_desconhecida).

?- preencher_prestador_inst(21, "Hospital Privado de Braga").
true .

?- prestadorID(21,R).
R = prestador(21, "Amaral", "Optometria", "Hospital Privado de Braga").

?- demo(prestador(21, "Amaral", "Optometria", "Hospital Privado de Braga"),R).
R = verdadeiro .
```

```
?- dataID(18,R).
R = data(18, 2018, 4, dia_desconhecido).

?- preencher_Data(18,5).
true .

?- dataID(18,R).
R = data(18, 2018, 4, 5).

?- demo(data(18, 2018, 4, 5),R).
R = verdadeiro .
```

## 3.6 Conclusão

Após a finalização do segundo trabalho prático, podemos afirmar que agora o nosso sistema é capaz de representar conhecimento perfeito e conhecimento imperfeito. Podemos concluir então, que a nossa base de conhecimento deixou de representar apenas lógica clássica e agora representa o valor lógico *desconhecido* e outros tipos de negação. A realização do mesmo permitiu-nos consolidar e melhorar os nossos conhecimentos sobre a matéria dada na unidade curricular de SRCR.