UNIVERSIDADE DO MINHO

Programação em lógica estendida e Conhecimento imperfeito.

Mestrado Integrado em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio
(2º semestre, ano letivo 2016/17)

A75135 Bruno Pereira
A73580 Maria Ana de Brito
A27748 Gustavo Andrez
A74634 Rogério Moreira
A76507 Samuel Ferreira

Braga,
2017-04-09

Resumo

Este relatório tem como objetivo, descrever e explicar todas as etapas da resolução do segundo exercício prático da Unidade Curricular de Sistemas de Representação de Conhecimento e Raciocínio, que tem como objetivo a representação de conhecimento imperfeito e programação em lógica estendida.

O projeto proposto baseia-se no desenvolvimento de um sistema de raciocínio e representação de conhecimento perfeito e imperfeito capaz de caracterizar um universo enquadrado na área da prestação de cuidados de saúde pela realização de serviços de atos médicos.

Este relatório fornece toda a informação necessária sobre os resultados obtidos e permite a compreensão dos mesmos. Para além da estratégia usada para resolver o problema, são também caracterizadas as funcionalidades e são também dados alguns exemplos das funcionalidades desenvolvidas.

Índice

Introdução

Motivação e Objetivos

Este projeto proposto na Unidade Curricular de Sistemas de Representação de Conhecimento e Raciocínio surge como uma oportunidade para colocar em prática os conhecimentos aí adquiridos.

Assim, o principal objetivo torna-se em desenvolver, através dos conhecimentos já adquiridos, um sistema de representação de conhecimento e raciocínio com capacidade para caracterizar um universo de discurso. Este universo de discurso contempla conhecimento perfeito, assim como conhecimento imperfeito.

Através deste exercício, mostramos como é que o PROLOG poderá ser útil para ser utilizado na resolução de problemas mais complexos.

Este projeto serve também para aplicar os conhecimentos adquiridos na Unidade Curricular a um problema real.

Estrutura do Relatório

O relatório é constituído por 4 partes. A primeira parte, menciona os principais objetivos e a motivação bem como uma pequena introdução ao projeto desenvolvido. A segunda apresenta-se algum conhecimento preliminar para entender o projeto, na terceira parte explicamos tudo o que foi desenvolvido durante a elaboração do projeto. Por fim, na quarta e última parte são apresentadas algumas conclusões, uma pequena interpretação dos resultados e sugestões de trabalho futuro. São ainda apresentadas no final do presente relatório as referências bibliográficas utilizadas para o desenvolvimento do projeto.

Introdução

Neste segundo exercício prático recorremos novamente à programação em lógica para a representação de conhecimento. Posto isto, e visto que o enunciado descreve um universo semelhante ao do exercício anterior partimos do trabalho anterior e por isso assumimos e utilizamos alguns dos pressupostos e inferências descritos anteriormente, mas que apenas representavam conhecimento perfeito, ou seja, onde o contradomínio era apenas verdadeiro ou falso.

Esta segunda parte incide na representação de conhecimento imperfeito, isto é, cujo o contradomínio toma o valor de verdadeiro, falso e desconhecido. Estendemos por isso o conhecimento já desenvolvido para este novo contexto. O conhecimento imperfeito representado é de três tipos: incerto, impreciso ou interdito, todos os três tipos estão representados no projeto.

O caso de estudo está, mais uma vez, relacionado com o desenvolvimento de um sistema de representação de conhecimento e raciocínio com capacidade para caracterizar um universo de discurso, com o qual de pretende abordar a temática das instituições de saúde e registo de eventos associados à prestação de cuidados. Como é descrito ao longo do relatório, as entidades descritas são algumas das que achamos pertinentes incluir: Atos Médicos, Especialidades, Cuidados, Médicos e Utentes.

Preliminares

Representação de conhecimento incompleto

Apesar de, na lógica tradicional, o conhecimento ser apenas representado em termos do que se pode provar ser verdadeiro e do que se pode provar ser falso, temos também interesse em ter conhecimento sobre o qual nada se pode concluir. Torna-se assim essencial arranjar uma forma de representar este tipo de conhecimento, já que a maioria dos programas escritos em lógica tem que representar este tipo de conhecimento. Consequentemente, torna-se possível agora distinguir três tipos de conclusões para uma questão: verdadeira, falsa ou desconhecida, dando uma maior flexibilidade ao mecanismo de inferência.

Alguns dos pressupostos em que um sistema de representação de conhecimento se baseia são:

- **Pressuposto do Mundo Aberto** – podem existir outras conclusões ou factos verdadeiros para além dos representados;

- **Pressuposto dos Nomes Únicos** – duas constantes diferentes entre si têm que designar entidades diferentes no universo representado;

- **Pressuposto do Domínio Aberto** – podem existir mais objetos no universo representado para além dos designados na base de conhecimento.

O projeto elaborado e aqui descrito tem em conta estes três pressupostos no sentido de dotar o sistema representado de mecanismos de raciocínio.

Extensão à programação em lógica

Um programa em lógica determina respostas em termos de verdade ou de falsidade das questões, não sendo possível, assim, implementar um mecanismo que possibilite abordar questões de informação incompleta.

O objetivo de estender este tipo de programação é o de passar a permitir explicitamente representar conhecimento negativo e imperfeito. A programação em lógica passa a contar com dois tipos de negação: negação por falha e negação forte (forma de identificar conhecimento negativo).

Valores Nulos

Os valores nulos servem para fazer a distinção entre situações em que as respostas a questões deverão ser concretizadas como conhecidas ou desconhecidas.

Existem três géneros de conhecimento desconhecido a serem representados:

- **Tipo Incerto** – valores desconhecidos, não pertencentes a um conjunto de valores admissíveis;

- **Tipo Impreciso, de um conjunto de valores** – valores desconhecidos, pertencentes a um conjunto finito e determinado de valores;

- **Tipo Interdito** – valores desconhecidos, que não são permitidos conhecer.

O sistema é assim capaz de representar valores de respostas do tipo: **verdadeiro**, **falso** ou **desconhecido**.

Demo

O predicado demo permite estender as soluções dos predicados, ou seja, para além dos valores lógicos das soluções de um predicado (verdadeiro ou falso), é possível recorrendo a este predicado uma terceira resposta, desconhecido.

Assim, uma questão é verdadeira, quando se encontra a sua prova. Uma questão é falsa, quando não se prova a sua veracidade e quando não é exceção. Uma questão é desconhecida quando não é verdadeira nem falsa.

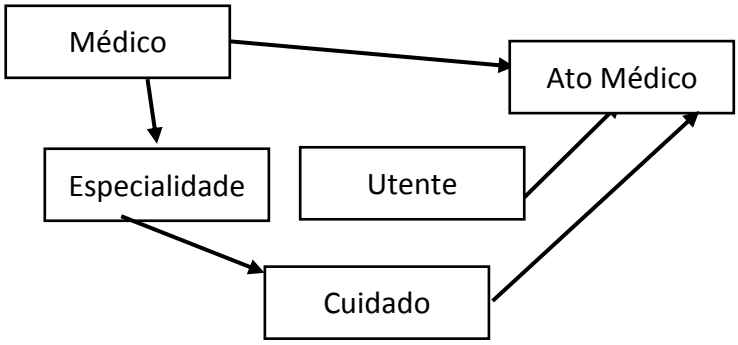
Descrição do trabalho realizado

Nesta secção será apresentada a forma como o grupo decidiu organizar o conhecimento do exercício proposto, partindo da estrutura mais elementar (os factos e os seus relacionamentos), a integridade da base de conhecimento em causa (invariantes definidos e sua justificação) e a representação de conhecimento imperfeito e negativo.

Nota: Para os diversos tipos de conhecimento representados é apenas dado um exemplo de desconhecido uma vez que, caso se queira aplicar novos tipos de desconhecido, os princípios a seguir são os mesmos.

Base de Conhecimento

Nesta secção serão explicitadas todas as etapas de resolução dos desafios fornecidos, bem como todas as decisões efetuadas durante o processo de resolução. A estrutura lógica que serviu para representar a nossa base de conhecimento foi a seguinte:



Aos três tipos de conhecimento propostos no enunciado do exercício prático acrescentamos ainda o conhecimento Especialidade e Médico de forma a representar de maneira eficiente estes dois tipos de informação na base de conhecimento.

Portanto, a base de conhecimento deste projeto pode ser caracterizada da seguinte forma:

- **Médico**: (Identificador do Médico, Nome, Idade, Identificador da Especialidade);
- **Especialidade**: (Identificador da Especialidade, Nome);
- **Utente**: (Identificador do Utente, Nome, Idade, Cidade);
- **Cuidado**: (Identificador do Ato Médico, Identificador da Especialidade, Hospital, Cidade);
- **Ato Médico:** (Horas, Minutos, Dia, Mês, Ano, Identificador do Utente, Identificador do Serviço, Identificador do Médico, Custo).

Integridade da Base de Conhecimento

Quando declaramos, por exemplo, o facto utente (u1,'Pedro Pires',21,'Braga') na nossa base de conhecimento, tanto o *functor* como as variáveis possuem nomes suficientemente fáceis para a compreensão do seu significado por parte de qualquer pessoa. Contudo, o *Prolog* desconhece o seu significado, ou seja, que utente representa uma pessoa que usa um cuidado de saúde, esse utente chama-se Pedro Pires, tem 21 anos de idade e mora em Braga, trata-se apenas de uma estrutura na base de conhecimento e não representa qualquer tipo de relacionamento. Temos, então, de arranjar mecanismos que permitam manter a base de conhecimento mais fiel possível à realidade em termos de contexto. Para isso temos nas linguagens de programação lógica os predicados que nos permitem controlar a inserção, remoção e consulta da informação contida na base de conhecimento, como por exemplo, com o uso de invariantes. Os invariantes usados neste exercício prático são de dois tipos:

- +Termo :: Premissa(s) – quando queremos adicionar conhecimento do tipo Termo esse novo conhecimento terá que respeitar a premissa ou premissas expressas no momento da inserção.
- -Termo :: Premissa(s) – quando queremos remover conhecimento do tipo Termo só o poderemos fazer se se cumprir determinada premissa ou premissas.

Em *Prolog* existem também predicados pré-definidos, como o *assert* e o *retract,* que nos permitem adicionar e remover factos na base de conhecimento, respetivamente. Contudo, é necessário definir predicados auxiliares, já que os predicados pré-definidos não garantem a integridade da base de conhecimento, torna-se por isso essencial utilizar predicados auxiliares e invariantes.

É importante também ter em consideração a forma como iremos realizar as operações de inserção e remoção do conhecimento em *Prolog,* em que a informação é inserida, ou removida, na base de conhecimento mediante a sua prova em função dos invariantes, ou seja, caso não cumpra as restrições impostas não é inserida, ou removida. Estas restrições, incidem principalmente sobre a questões relacionados com a integridade de domínio, integridade referencial e estrutural.

Contudo, a representação do conhecimento imperfeito possui algumas características próprias associadas à consistência e integridade da base de conhecimento. Para que se possa diferenciar restrições impostas por invariantes sobre o conhecimento perfeito ou por exceções no conhecimento imperfeito foi necessário implementar alguns predicados:

- **Excecao(Termo)** – predicado que adiciona um caso de exceção;
- **Nulo(Termo)** – quando queremos tomar como nulo o termo;

Para cada caso distinto dos valores desconhecidos foram definidas estratégias distintas.

Conhecimento Perfeito Positivo

Relativamente a conhecimento perfeito positivo, temos conhecimento 5 entidades.

- **Utente**

O utente possui um ID do utente, nome, idade e a sua morada.

```
utente(u1,'Pedro Pires',21,'Braga').
utente(u2,'Marta Barros',29,'Braga').
utente(u3,'Guilherme Martins',36,'Guimaraes').
utente(u8, 'Joana Monteiro', 12, 'Algarve').
utente(u9, 'Manuel Santidade', 23, 'Palmela').
```

- **Médico**

O médico possui um ID do médico, nome, idade e o ID da sua especialidade.

```
medico(m1, 'Fernando Fernandes', 34, e1).
medico(m2, 'Paulo Pinho', 48, e2).
medico(m3, 'Augusto Agostinho', 52, e3).
medico(m4, 'Bernardo Barros', 44, e4).
medico(m5, 'Carlos Costa', 45, e5).
```

- **Especialidade**

A especialidade possui o ID da especialidade e a sua designação.

```
especialidade(e1, 'geral').
especialidade(e2, 'pediatria').
especialidade(e3, 'urologia').
especialidade(e4, 'ginecologia').
especialidade(e5, 'psiquiatria').
especialidade(e6, 'radiologia').
```

- **Cuidado**

O cuidado possui o ID do serviço, o ID da especialidade, a instituição e o cidade da instituição.

```
cuidado(s1,e5,'hospital sao joao','Porto').
cuidado(s2,e4,'hospital sao marcos','Braga').
```

```
cuidado(s3,e2,'hospital sao joao','Porto').
cuidado(s4,e1,'centro de saude do caranda','Braga').
cuidado(s5,e3,'hospital sao joao','Porto').
cuidado(s6,e2,'maternidade alfredo da costa','Lisboa').
```

- **Ato médico**

O ato médico possui as horas, os minutos, o dia, o mês, o ano, o ID do utente, o ID do serviço, o ID do médico e o custo do ato médico.

```
atomedico(12, 20, 10, 3, 2017, u1, s4, m1, 12.20).
atomedico(21, 00, 2, 3, 2017, u2, s3, m2, 16.50).
atomedico(09, 40, 14, 2, 2017, u3, s5, m3, 21.00).
```

Conhecimento Perfeito Negativo

A extensão à programação em lógica estendida permite representar explicitamente informação negativa, isto é, informação que sabemos que é irrevogavelmente falsa. Deste modo, o conhecimento perfeito negativo, à semelhança do conhecimento perfeito positivo, dá-nos a possibilidade de fazer a representação completa de informação falsa.

Deste modo, surgiu a necessidade de definir predicados auxiliares, representados por **-**, com o objetivo de fazer a negação por falha na prova, tal como se pode ver nos seguintes exemplos:

```
-medico(IdMed, Nome, Idade, Esp) :-nao(medico(IdMed, Nome, Idade, Esp)), nao(excecao(medico(IdMed, Nome, Idade, Esp))).

-especialidade(IdEsp, Desc) :- nao(especialidade(IdEsp, Desc)), nao(excecao(especialidade(IdEsp, Desc))).

-utente(IdU, Nome, Idade, Morada) :- nao(utente(IdU, Nome, Idade, Morada)), nao(excecao(utente(IdU, Nome, Idade, Morada))).

-cuidado(IdS, IdE, Hosp, Local) :- nao(cuidado(IdS, IdE, Hosp, Local)), nao(excecao(cuidado(IdS, IdE, Hosp, Local))).

-atomedico(Hora, Minutos,Dia, Mes, Ano,IdU,IdS,IdMed,Custo) :- nao(atomedico(Hora, Minutos,Dia, Mes, Ano,IdU,IdS,IdMed,Custo)), nao(excecao(atomedico(Hora, Minutos,Dia, Mes, Ano,IdU,IdS,IdMed,Custo))).
```

Como se pode analisar, a informação é negativa se não se conseguir provar que é verdade e se não é uma exceção.

- **Utentes**

A representação negativa de informação relativa aos utentes é feita através da negação de factos, tal como:

```
-utente(u4, 'Rita Sousa', 70, 'Faro').
```

Assim, estamos a afirmar que não existe uma utente chamada Rita Sousa, com ID u4, 70 anos e que mora em Faro.

- **Cuidados Médicos**

A representação de informação falsa relativa aos cuidados médicos é feita através da negação de factos, tal como:

```
-cuidado(s11,e6,'hospital sao joao','Porto').
```

Assim, nega-se que existe um cuidado médico com ID de serviço s11, especialidade e6, no Hospital São João, no Porto.

- **Atos Médicos**

A representação de informação negativa relativamente aos atos médicos é feita através de:

```
-atomedico(14,20,14,2,2017,u2,s1,m5,40.90).
```

Assim, afirma-se que não foi realizado um ato médico às 14h20min, no dia 14 de fevereiro de 2017, com o utente cujo ID é u2, serviço s1, médico m5 e que custou 40.90€.

- **Outros Predicados**

O grupo decidiu fazer representação de conhecimento negativo para os predicados que representam o os utentes, os cuidados e os atos médicos. Assim, a informação falsa para os restantes predicados (especialidade e médico) é feita de forma análoga.

- **Invariantes**

A inserção de conhecimento negativo é controlada, de modo a não criar incoerências na base de conhecimento. Deste modo, é preciso não permitir que certas situações ocorram, tal como por exemplo:

- Não inserir conhecimento negativo repetido, uma vez que é redundante. O invariante que controla este aspeto é o seguinte:

```
+(-Termo) :: (solucoes(Termo, -Termo, S),
                comprimento(S,N),
                  N =< 2).
```

- Não inserir conhecimento negativo que contradiga o conhecimento positivo perfeito que se encontra na base de conhecimento.

  +(-Termo) :: nao(Termo).

- Não inserir conhecimento positivo que contradiga o conhecimento positivo negativo já existente na base de conhecimento.

  +Termo :: nao(-Termo).

- **Evolução de Conhecimento Negativo Perfeito**

A inserção de conhecimento negativo perfeito na base de conhecimento é feita através de um predicado evolução específico, que, por sua vez, chama o predicado evolução com a adição de – antes do facto que se quer inserir.

Deste modo, a inserção de conhecimento negativo relativo ao utente é feita da seguinte maneira:

```
evolucaoUtenteNeg(IdU, Nome, Idade, Morada) :- evolucao(-
utente(IdU, Nome, Idade, Morada)).
```

A inserção de conhecimento negativo relativamente aos cuidados é feita assim:

```
evolucaoCuidadoNeg(IdS, IdEsp, Hosp, Local) :- evolucao(-
cuidado(IdS, IdEsp, Hosp, Local)).
```

A inserção de conhecimento negativo relativo aos atos médicos é realizada através do seguinte predicado:

```
evolucaoAtoMedicoNeg(Hora, Minutos, Dia, Mes, Ano, IdU, IdS,
IdMed, Custo) :- evolucao(-atomedico(Hora, Minutos, Dia,
Mes, Ano, IdU, IdS, IdMed, Custo)).
```

- **Regressão de Conhecimento Negativo Perfeito**

Por outro lado, a eliminação de conhecimento negativo perfeito é feita através de um predicado regressão específico, que, por sua vez, invoca o predicado remove.

A exclusão de conhecimento negativo relativo ao utente é feita da seguinte maneira:

```
regressaoUtenteNeg(IdU, Nome, Idade, Morada) :- remove(-
utente(IdU, Nome, Idade, Morada)).
```

A exclusão de conhecimento negativo relativamente ao cuidado médico é feita assim:

```
regressaoCuidadoNeg(IdS, IdEsp, Hosp, Local) :- remove(-
cuidado(IdS, IdEsp, Hosp, Local)).
```

Por fim, a eliminação de conhecimento negativo relativo aos atos médicos é realizada através do seguinte predicado:

```
regressaoAtoMedicoNeg(Hora, Minutos, Dia, Mes, Ano, IdU,
IdS, IdMed, Custo) :- remove(-atomedico(Hora, Minutos, Dia,
Mes, Ano, IdU, IdS, IdMed, Custo)).
```

Conhecimento Imperfeito Incerto

A representação de informação incompleta do tipo I, ou conhecimento incerto, é caracterizada pela incerteza no conhecimento, ou seja, não se conhece nada acerca do valor, é desconhecido. Nos exemplos que se seguem são mencionadas *flags* representativas de conhecimento incerto. Estas *flags* são genéricas e inalteráveis, ou seja, para representar um tipo de conhecimento incerto é obrigatório utilizar a *flag* correspondente.

- ### Instituição com especialidade incerta

Em determinada instituição, o utente ao preencher os seus dados pessoais, esqueceu-se de escrever qual era a sua morada. Como a instituição deseja ter todo o seu conhecimento representado numa base de conhecimento, perante este problema, decidiu utilizar a representação de conhecimento incerto, de modo a não perder a informação relativa ao utente, apesar de não ser completa. O seguinte exemplo é relativo a esse utente:

```
utente(u7, 'John McCarthy', 56, morada_desconhecida).
```

Assim na representação do conhecimento incerto, para demonstrar que a informação é incerta, é utilizada a *flag* morada_desconhecida no local onde a morada deveria ficar.

Apesar de necessário, este predicado não é suficiente para representar a informação do tipo incerta. Visto que o predicado demo controla se o tipo de resposta é verdadeiro, falso, ou desconhecido, é necessário adicionar uma exceção

que indica que o conhecimento é incerto. A exceção, para o caso do utente referido em cima, será a seguinte:

```
excecao(utente(IdU, Nome, Idade, Morada)) :-
utente(IdU, Nome, Idade, morada_desconhecida).
```

Se for desejável representar qualquer outro parâmetro como incerto, basta alterar o nome da *flag* e o respetivo parâmetro.

- Instituição com especialidade incerta

Um engenheiro de desenvolvimento de *software* deseja criar uma aplicação móvel que contenha todas as instituições de saúde na zona de Braga e as suas características. No entanto, devido à incapacidade de comunicar com uma certa instituição, não conhece a especialidade referente a um determinado serviço. Como tem conhecimento que esse cuidado existe, decidiu representar com a especialidade incerta. Assim, na base de conhecimento, ficou assim representado o conhecimento:

```
    cuidado(s10, esp_desconhecida, 'centro de saude de
infias', 'Braga').
```

Analogamente ao exemplo anterior do utente, temos uma *flag* que indica que a especialidade é incerta, sendo também necessário adicionar a exceção:

```
    excecao(cuidado(IdS, IdE, Hosp, Local)) :-
cuidado(IdS, esp_desconhecida, Hosp, Local).
```

Através deste exemplo, podemos concluir que é possível representar qualquer conhecimento incerto relativamente ao cuidado, sendo necessário alterar o tipo da *flag* e a exceção correspondente.

- Ato médico com horário incerto

Um utente, afligido por dores inexplicáveis, decidiu ligar ao seu médico para marcar um ato médico. No entanto, devido ao horário muito ocupado do ato médico, este só lhe conseguiu precisar o dia em que ia estar disponível, ou seja, não se sabe as horas nem os minutos do ato médico. Como é necessário registar o ato médico na base de conhecimento, este teve de ser registado com hora e minutos incertos, como é visível no seguinte exemplo:

```
atomedico(hora_desconhecida, minutos_desconhecidos, 10, 7,
2018, u3, s5, m3, 143.90).
```

Tal como nos exemplos anteriores, utilizamos *flags* que representam o conhecimento incerto, bem como a seguinte exceção:

```
excecao(atomedico(Hora, Minutos, Dia, Mes, Ano, IdU, IdS, IdMed, Custo)):- atomedico(hora_desconhecida, minutos_desconhecidos,Dia, Mes, Ano, IdU, IdS, IdMed, Custo).
```

- Evolução do conhecimento incerto

A evolução do conhecimento incerto segue duas etapas. A primeira, será a evolução do conhecimento. A segunda será a inserção da exceção correspondente.

Só irá ser analisada evolução de uma morada incerta, visto que as evoluções de especialidade incerta e horário incerto, só variam nos seus parâmetros e no conteúdo a adicionar.

- Evolução de um utente com morada incerta

O predicado evolucaoMoradaIncerta recebe como parâmetros o ID do utente, o seu nome e a sua morada. Este predicado, insere, utilizando o predicado evolução, o predicado utente com o ID, nome e idade passados como parâmetros, e a flag morada_desconhecida. Desta maneira, obrigamos a que não haja *flags* relativas à mesma informação, mas escritas de maneira diferente. Finalmente, insere-se a exceção correspondente. O seguinte excerto de código representa o predicado evolucaoMoradaIncerta.

```
evolucaoMoradaIncerta(IdU, Nome, Idade) :-
    evolucao(utente(IdU, Nome, Idade, morada_desconhecida)),
assert((excecao(utente(Id, N, I, M)) :- utente(Id, N, I, morada_desconhecida))).
```

Deste modo, é possível criar qualquer predicado de evolução de conhecimento incerto, desde que seja específico ao parâmetro incerto. Assim, a estratégia passa por receber os parâmetros certos, e colocar no parâmetro incerto a *flag* correspondente. Finalmente, adiciona-se a exceção com o parâmetro incerto.

- Evolução do conhecimento incerto

A regressão do conhecimento incerto, tal como a evolução, segue duas etapas. A primeira, será a regressão do conhecimento. A segunda será a remoção da exceção correspondente.

Como o processo de regressão é extremamente semelhante para o conhecimento incerto, só será analisada a regressão de um utente com a morada incerta.

- Regressão de um utente com morada incerta

O predicado regressaoMoradaIncerta recebe como parâmetros o ID do utente, o seu nome e a sua morada. Este predicado, remove, utilizando o predicado removerUtente, o utente com o ID, nome e idade passados como parâmetros, sendo a morada a flag morada_desconhecida. De seguida, remove-se a exceção

correspondente. O seguinte excerto de código representa o predicado regressaoMoradaIncerta .

```
regressaoMoradaIncerta(IdU, Nome, Idade) :-
removerUtente(IdU, Nome, Idade, morada_desconhecida),
remove((excecao(utente(Id, N, I, M)) :- utente(Id, N, I,
morada_desconhecida))).
```

A criação de novos predicados de regressão de conhecimento incerto seguirá os seguintes passos:

- Fornecer ao predicado todos os dados desejados, exceto o parâmetro incerto.
- Na remoção, utilizar a *flag* do parâmetro incerto, em vez de um possível parâmetro.
- Remover a exceção que contém esse parâmetro incerto.

Algo importante de referir, é que, aquando a inserção de exceções, não há nenhum invariante que verifica se está a referir exceções repetidas. A razão de inserções repetidas é para controlar as remoções. Sabendo que existem exceções repetidas, podemos remover a exceção sem ter de verificar se existe essa exceção. Este conhecimento torna-se ainda mais relevante, pois, quando não existirem exceções, sabemos que não há conhecimento do tipo incerto na base de conhecimento.

Conhecimento Imperfeito Impreciso

O conhecimento imperfeito impreciso representa informação incompleta desconhecida de um dado conjunto de valores. Sabemos qual é o conjunto possível de hipóteses, porém desconhecemos a valor exato.

- Utentes

  - Valores Discretos

Um caso de conhecimento impreciso ocorre quando não temos a certeza do valor verdadeiro, porém sabemos que pode tomar um ou outro valor discreto. Neste caso de estudo, temos o seguinte exemplo:

```
excecao(utente(u10, 'Diana Dias', 25, 'Viseu')).
excecao(utente(u10, 'Diana Dias', 26, 'Viseu')).
```

Em que se desconhece a idade da utente Diana Dias, porém sabe-se que pode ter 25 ou 26 anos. Esta situação pode surgir devido à inexistência de conhecimento perfeito positivo relativo a esta utente, uma vez que não se teve a oportunidade de a questionar diretamente sobre a sua idade. A tia afirmou que a

sobrinha tinha 25 anos de idade, contudo o seu primo disse que a Diana tinha 26 anos. Assim, criou-se conhecimento impreciso sobre dois valores discretos.

- ## Atos Médicos

➢ Valores Discretos

Devido à ortografia do utente com ID u8 quando preencheu o formulário do seu ato médico, não se sabe se o custo da consulta foi de 10€ ou de 100€.

```
excecao(atomedico(13,50, 17,10, 2019, u8, s2, m4, 10)).
excecao(atomedico(13,50, 17,10, 2019, u8, s2, m4, 100)).
```

➢ Intervalo contínuo de valores

Outro caso de conhecimento impreciso acontece quando não sabemos o valor exato de algo, no entanto sabemos que se encontra entre dois valores, podendo tomar qualquer valor compreendido entre o intervalo indicado.

Neste caso, o utente com ID u9 não quis revelar o valor exato do custo da sua consulta, porém sabe-se que não pode ter custado menos do que 30€ e menos do que 40€.

```
excecao(atomedico(18, 20, 23, 11, 2018, u9, s1, m5, X)) :- X
>= 30, X =< 40.
```

- ## Evolução de Conhecimento Impreciso

➢ Valores Discretos

De forma a inserir conhecimento impreciso na base de conhecimento, é necessário inserir uma exceção com o termo indicado.

Assim, como na maior parte dos casos, insere-se mais do que um valor do conjunto possível de hipóteses, o predicado da evolução recebe como input uma lista de termos, tal como se pode verificar nos seguintes predicados:

```
evolucaoUtenteImpreciso([]).
evolucaoUtenteImpreciso([Termo|T]) :-
                  assert(excecao(Termo)),
                    evolucaoUtenteImpreciso(T).

evolucaoCustoDiscImpreciso([]).
evolucaoCustoDiscImpreciso([Termo|T]) :-
                  assert(excecao(Termo)),
                    evolucaoCustoDiscImpreciso(T).
```

- Intervalo de valores contínuos

À semelhança do caso anterior, para se proceder à inserção de conhecimento impreciso, é necessário inserir a exceção que confirma se um dado valor se encontra dentro do intervalo determinado. Desta forma, para além dos dados do ato médico, é preciso inserir os limites inferior e superior do intervalo, como se pode analisar:

```
evolucaoCustoImpreciso(H, Min, Dia, Mes, Ano, IdU, IdS,
IdMed, LI, LS) :-
assert((excecao(atomedico(H, Min, Dia, Mes, Ano, IdU, IdS,
IdMed, Y)) :- Y >= LI, Y =< LS)).


evolucaoDiaImpreciso(H, Min, Mes, Ano, IdU, IdS, IdMed,
Custo, LI, LS) :-
assert((excecao(atomedico(H, Min, X, Mes, Ano, IdU, IdS,
IdMed, Custo)) :- X >= LI, X =< LS)).
```

- **Regressão de Conhecimento Impreciso**

De forma a eliminar conhecimento impreciso da base de conhecimento é necessário eliminar a exceção do termo indicado.

```
regressaoExcecao([]).
regressaoExcecao([Termo|R]) :- remove(excecao(Termo)),
                               regressaoExcecao(R).
```

Conhecimento Imperfeito Interdito

A representação de informação incompleta do tipo III, ou conhecimento interdito, é caracterizada pelo desconhecimento da informação e a impossibilidade de se conhecer. Ou seja, não se sabe se algo é verdadeiro/falso, nem nunca se poderá saber. Nos exemplos que se seguem são mencionadas *flags* representativas de conhecimento interdito. Estas *flags* são genéricas e inalteráveis, ou seja, para representar um tipo de conhecimento interdito é obrigatório utilizar a *flag* correspondente.

De seguida, apresentar-se-ão três exemplos de como representar conhecimento interdito. No entanto, estes não são os únicos casos em que é possível representar conhecimento interdito. De modo a representar conhecimento interdito sobre parâmetros diferentes, é unicamente necessário mudar esses parâmetros. No exemplo do utente, a morada é interdita, no entanto, se fosse desejável que a idade fosse interdita, as alterações passariam, maioritariamente, por mudar a *flag* de morada_interdita para idade_interdita.

- **Utente com morada interdita**

Uma figura pública mundial sentiu a necessidade de ir a uma instituição de saúde. Ao preencher os seus dados, de acordo com o seu estatuto, exigiu que ninguém pudesse conhecer a sua morada. Assim sendo, a sua morada será interdita a quem a quiser visualizar. Eis o predicado representativo deste exemplo.

```
utente(u12, 'Cristiano Ronaldo', 33, morada_interdita).
```

Neste caso, ao inserir-se o conhecimento, é necessário colocar a *flag* morada_interdita, bem como um predicado nulo, uma exceção e um invariante.

A exceção serve para o predicado demo verificar se o valor é desconhecido ou não, sendo necessário colocar o parâmetro desejado com uma *flag* indicativa de conhecimento interdito.

```
excecao(utente(IdU, Nome, Idade, Morada)) :- utente(IdU, Nome, Idade, morada_interdita).
```

Finalmente, é imperativo colocar um invariante que não permite a inserção de conhecimento positivo perfeito sobre conhecimento interdito, ou seja, é feita a verificação se existe algum predicado com o ID do utente igual ao que foi indicado e, caso exista, verifica-se a existência de um valor nulo com a sua morada O seguinte invariante é utilizado para o caso do utente com morada interdita.

```
+utente(IdU, Nome, Idade, Morada) ::
(solucoes(M, (utente(u12, Nome, Idade, M), nao(nulo(M))), S),
        comprimento(S,N),
        N==0).
```

Antes de colocar o invariante, é necessário adicionar o predicado nulo(CN), caracterizado da seguinte maneira:

```
nulo(morada_interdita).
```

O predicado nulo tem como principal objetivo armazenar todos os valores interditos para poderem ser depois testados no invariante.

Assim, quando o predicado evolução verificar os invariantes, se estiver a tentar inserir conhecimento sobre o conhecimento incerto, não será possível efetuar a inserção.

- **Médico com especialidade interdita**

Um médico, trabalhador numa instituição de saúde, pediu à administração que não fosse conhecida a sua especialidade, tornando-a interdita a quem quer que quisesse consultá-la. Eis o predicado representativo deste exemplo:

```
medico(m6, 'Rodrigo Silva', 53, esp_interdita).
```

Todos os dados referentes ao médico estão presentes na base de conhecimento, exceto a sua especialidade, que é representada pela flag esp_interdita.

Tal como no exemplo do utente com a morada interdita, será necessário adicionar a exceção, o predicado nulo e o invariante que não permite inserção de conhecimento positivo perfeito sobre conhecimento interdito.
Exceção:

```
excecao(medico(IdMed, Nome, Idade, Esp)) :-
medico(IdMed, Nome, Idade, esp_interdita).
```

Predicado nulo:
```
nulo(esp_interdita).
```

Invariante que verifica se existe algum predicado com o ID do médico igual ao que foi indicado e, caso exista, verifica-se a existência de um valor nulo com a sua especialidade:

```
+medico(IdU, Nome, Idade, Especialidade) ::
(solucoes(E, (medico(m6, Nome, Idade, E), nao(nulo(E))), S),
comprimento(S,N),
N==0).
```

- **Ato médico com custo interdito**

Um utente, após ter realizado um ato médico, fez um pedido à instituição de saúde
para que o custo do ato médico nunca pudesse ser revelado. A instituição acedeu ao pedido e assim, o ato médico apresenta todos os dados do ato médico, exceto o seu custo, que aparece como interdito. De seguida, apresenta-se um exemplo deste caso.

```
atomedico(16, 45, 9, 10, 2017, u1, s5, m3, custo_interdito).
```

De modo a tornar o custo interdito, foi necessário utilizar a flag custo_interdito, juntamente com a exceção correspondente:

```
    excecao(atomedico(Hora,        Minutos,Dia,        Mes,
Ano,IdU,IdS,IdMed,Custo)) :-
atomedico(Hora,Minutos,Dia,Mes,Ano,IdU,IdS,            IdMed,
custo_interdito),
```

o predicado nulo:

```
    nulo(custo_interdito)
```

e o invariante que não permite a inserção de conhecimento positivo perfeito sobre conhecimento interdito:

```
     +atomedico(Hora, Minutos, Dia, Mes, Ano, IdU, IdS, IdMed,
Custo) ::
(solucoes(C, (atomedico(16, 45, 9, 10, 2017, u1, s5, m3, C),
nao(nulo(C))), S),
comprimento(S,N),
N==0).
```

- **Evolução de conhecimento interdito**

A evolução do conhecimento interdito segue quatro inserções.

1. Evolução do termo (por exemplo: utente);
2. Inserção do invariante que não permite inserção de conhecimento positivo perfeito sobre conhecimento interdito;
3. Inserção da exceção;
4. Inserção do valor nulo.

Assim, a evolução é feita para cada caso específico, ou seja, dependendo do parâmetro que for interdito. De seguida, será ilustrado um exemplo de uma evolução de um caso interdito que, apesar de ser específico, será muito semelhante a todos os outros que se queiram implementar, pois basta mudar o valor das *flags* e dos parâmetros não interditos.

- Evolução de um utente com a morada interdita

O predicado evolucaoMoradaInterdita recebe como parâmetros o ID do utente, o seu nome e a morada. Primeiramente, é efetuado uma inserção através do predicado evolução, que insere o termo utente com os parâmetros recebidos e morada com a *flag* morada_interdita. De seguida inserem-se o invariante que não permite inserção de conhecimento positivo perfeito sobre conhecimento interdito, a exceção e o predicado nulo correspondente. O seguinte segmento de código representa o predicado evolucaoMoradaInterdita.

```
evolucaoMoradaInterdita(IdU, Nome, Idade) :-
evolucao(utente(IdU,              Nome,              Idade,
morada_interdita)),assert(+utente(Id, N, I, Morada) ::
(solucoes(M, (utente(IdU, N, I, M), nao(nulo(M))), S),
     comprimento(S,C),
     C==0)),
assert((excecao(utente(Id, N, I, M)) :- utente(Id, N, I,
morada_interdita))),
assert(nulo(morada_interdita)).
```

- Regressão de um utente com a morada interdita

O predicado regressaoMoradaInterdita recebe como parâmetros o ID do utente, o seu nome e a morada. Primeiramente, é efetuado uma remoção através do predicado removerUtente, que remove o termo utente com os parâmetros recebidos e morada com a *flag* morada_interdita. De seguida removem-se o invariante que não permite inserção de conhecimento positivo perfeito sobre conhecimento interdito, a exceção e o predicado nulo(CNNN) correspondente. O seguinte segmento de código representa o predicado regressaoMoradaInterdita.

```
regressaoMoradaInterdita(IdU, Nome, Idade) :-
        removerUtente(IdU, Nome, Idade, morada_interdita),
           remove(+utente(Id, N, I, Morada) ::
               (solucoes(M,    (utente(IdU,    N,    I,    M),
nao(nulo(M))), S),
                        comprimento(S,C),
                        C==0)),
             remove((excecao(utente(Id, N, I, M)) :-
                    utente(Id, N, I, morada_interdita))),
                remove(nulo(morada_interdita)).
```

Inserção de Conhecimento Positivo sobre Conhecimento Negativo e Imperfeito

- **Conhecimento Perfeito Negativo**

Graças ao invariante acima mencionado, não é permitida a inserção de conhecimento perfeito positivo quando há conhecimento perfeito negativo na base de conhecimento sobre o mesmo predicado. Assim, garante-se que não ocorre nenhuma contradição entre conhecimentos.

```
+Termo :: nao(-Termo).
```

- **Conhecimento Imperfeito Incerto**

Sempre que se pretende inserir conhecimento positivo quando há conhecimento incerto sobre o mesmo predicado na base de conhecimento, é necessário eliminar esse desconhecimento e inserir o conhecimento perfeito.

Assim, para os utentes elimina-se, caso existam, as exceções de conhecimento incerto relativas ao termo indicado e o facto respetivo que contém a flag de desconhecimento, recorrendo predicado que é chamado no registoUtente:

```
verificaMoradaIncerta(utente(IdU, Nome, Idade, Morada))
:- removeMoradaIncerta(utente(IdU, Nome, Idade,
Morada)).
```

```
removeMoradaIncerta(utente(IdU, Nome, Idade, Morada))
:- demo(utente(IdU, Nome, Idade, morada_desconhecida),
verdadeiro),

removerUtente(IdU, Nome, Idade, morada_desconhecida),

retract((excecao(utente(Id, N, I, M)) :-

utente(Id, N, I, morada_desconhecida))).
```

```
removeMoradaIncerta(utente(IdU, Nome, Idade,
Morada)).
```

No caso dos cuidados médicos, quando se pretende inserir conhecimento positivo perfeito quando há conhecimento incerto sobre o mesmo predicado, procede-se à eliminação da sua exceção e predicado, tal como se pode verificar:

```
removeEspecialidadeIncerta(cuidado(Id, Descricao,
Instituicao, Cidade)) :- demo(cuidado(Id,
esp_desconhecida, Instituicao, Cidade), verdadeiro),
removerCuidado(Id, esp_desconhecida, Instituicao,
Cidade),
retract((excecao(cuidado(IdS, D, I, C)) :-
cuidado(IdS, esp_desconhecida, I, C))).
```

```
removeEspecialidadeIncerta(cuidado(Id, Descricao,
Instituicao, Cidade)).
```

Por fim, relativamente aos atos médicos, remove-se a exceção a si associada, bem como o facto com a flag de desconhecimento, como se pode analisar em:

```
removeHorarioIncerto(atomedico(Hora, Minutos, Dia, Mes,
Ano, IdU, IdS, IdMed, Custo)) :-
demo(atomedico(hora_desconhecida,
minutos_desconhecidos, Dia, Mes, Ano, IdU, IdS, IdMed,
Custo), verdadeiro),
removerAtoMedico(hora_desconhecida,
minutos_desconhecidos, Dia, Mes, Ano, IdU, IdS, IdMed,
Custo),
remocao((excecao(atomedico(H, Min, D, M, A, IdUt,
IdSrv, IdM, C)) :-
atomedico(hora_desconhecida, minutos_desconhecidos, D,
M, A, IdUt, IdSrv, IdM, C))).
```

```
removeHorarioIncerto(atomedico(Hora, Minutos, Dia,
Mes, Ano, IdU, IdS, IdMed, Custo)).
```

- **Conhecimento Imperfeito Impreciso**

Neste caso, sempre que queremos introduzir conhecimento perfeito positivo quando na base de conhecimento existe conhecimento impreciso sobre o mesmo predicado, é necessário retirar as exceções a si associadas.

No caso dos utentes, elimina-se exceções de imprecisão sobre a idade, recorrendo a um predicado que é usado em registoUtente, tal como se pode verificar em:

```
apagaExcecoesIdade(utente(IdU, Nome, Idade, Morada)) :-
                demo(excecao(utente(IdU, Nome, _,
Morada)), verdadeiro),
                solucoes(utente(Id, N, I, M),
excecao(utente(IdU,N,_,M)),S),
                regressaoExcecao(S).

apagaExcecoesIdade(utente(IdU, Nome, Idade, Morada)).
```

No caso dos atos médicos, a eliminação das exceções sobre os custos das consultas é feita através de um predicado, como se pode analisar em:

```
apagaExcecoesCusto(atomedico(Hora, Minutos, Dia, Mes,
Ano, IdU, IdS, IdMed, Custo)) :-
        demo(atomedico(Hora, Minutos, Dia, Mes, Ano,
IdU, IdS, IdMed, _), verdadeiro),
          solucoes(atomedico(Hora, Minutos, Dia, Mes,
Ano, IdU, IdS, IdMed, C), excecao(atomedico(Hora,
Minutos, Dia, Mes, Ano, IdU, IdS, IdMed, _)),S),
            regressaoExcecao(S).

apagaExcecoesCusto(atomedico(Hora, Minutos, Dia, Mes, Ano,
IdU, IdS, IdMed, Custo)).
```

Em relação às exceções que tratam intervalos de valores contínuos, não foi possível fazer o predicado que elimina a sua exceção, pois é impossível fazer match com o predicado da exceção, uma vez que este tem a seguinte forma:

```
excecao(atomedico(18, 20, 23, 11, 2018, u9, s1, m5, X)) :- X
>= 30, X =< 40.
```

Logo, não foi possível definir um predicado que conjugasse o X, que seria uma incógnita, com os limites inferior e superior, que também teriam de ser variáveis, de forma a serem genéricos. Quando o utilizador insere conhecimento positivo perfeito não indica os valores extremos do intervalo que já existe na base de conhecimento, logo foi impossível criar um predicado que conseguisse apagar corretamente a exceção deste tipo de desconhecimento.

- **Conhecimento Imperfeito Interdito**

O conhecimento interdito representa a informação desconhecida e que não é possível conhecer, logo não é permitida qualquer inserção de conhecimento positivo perfeito sobre um predicado que tenha a si associada informação interdita.

Deste modo, é preciso fazer a verificação se existe algum predicado como o que foi indicado que acarrete algum tipo de desconhecimento e se algum dos seus valores é um valor nulo. Caso exista, não é possível inserir, caso contrário, a inserção é efetuada.

No caso dos utentes, é feita a verificação se existe algum predicado com o ID do utente igual ao que foi indicado e, caso exista, verifica-se a existência de um valor nulo com a sua morada, tal como se pode observar em:

```
+utente(IdU, Nome, Idade, Morada) ::
    (solucoes(M, (utente(u12, Nome, Idade, M),
nao(nulo(M))), S),
          comprimento(S,N),
          N==0).
```

No caso dos médicos, é feita a verificação se existe algum predicado com o ID do médico igual ao que foi indicado e, caso exista, verifica-se a existência de um valor nulo com a sua especialidade, tal como se pode verificar em:

```
+medico(IdU, Nome, Idade, Especialidade) ::
    (solucoes(E, (medico(m6, Nome, Idade, E), nao(nulo(E))),
S),
          comprimento(S,N),
          N==0).
```

No caso dos atos médicos, é feita a verificação se existe algum predicado com todos os dados do ato médico indicado (excetuando o custo) e, caso exista, verifica-se se o seu custo é interdito, como se pode analisar em:

```
+atomedico(Hora, Minutos, Dia, Mes, Ano, IdU, IdS, IdMed,
Custo) ::
    (solucoes(C, (atomedico(16, 45, 9, 10, 2017, u1, s5, m3,
C), nao(nulo(C))), S),
          comprimento(S,N),
          N==0).
```

Sistema de inferência

Para a implementação de um sistema de inferência capaz de suportar várias perguntas, é necessário tomar determinadas decisões. O sistema irá suportar disjunções e conjunções de respostas, o que não traria nenhum problema se só fossem considerados dois tipos de resposta (verdadeiro e falso). No entanto, o desconhecido também pode ser uma resposta, logo é necessário definir valores para a conjunção e disjunção do desconhecido com esses valores.

De seguida, apresentam-se as tabelas relativas à disjunção e conjunção de valores.

Conjunção(∧)	Verdadeiro	Falso	Desconhecido
Verdadeiro	V	F	D
Falso	F	F	F
Desconhecido	D	F	D

Disjunção(∨)	Verdadeiro	Falso	Desconhecido
Verdadeiro	V	V	V
Falso	V	F	D
Desconhecido	V	D	D

Relativamente ao desconhecido, temos que qualquer que seja a operação, disjunção ou conjunção, desconhecido com desconhecido resulta em desconhecido, visto que os dois podem corresponder a desconhecidos diferentes. Relativamente ao desconhecido os outros valores, a lógica seguida foi a seguinte:

- **V e D:** como na conjunção o verdadeiro é o elemento mais fraco, o resultado é desconhecido.
- **F e D:** como na conjunção o falso é o elemento mais forte, o resultado é falso.
- **V ou D:** como na disjunção o verdadeiro é o elemento mais forte, o resultado é verdadeiro.
- **F ou D:** como na disjunção o falso é o elemento mais fraco, o resultado é desconhecido.

Com este conhecimento, podemos avançar para o predicado demoListas,predicado este que recebe uma lista de questões e uma lista onde colocará as respostas. Cada questão, pode ser a conjunção ou a disjunção de várias questões. O predicado utiliza a recursividade para avançar de questão para questão e o predicado demoThreeValued para calcular a resposta à pergunta efetuada. O seguinte excerto de código é referente ao predicado demoListas.

```
demoListas([], []).
demoListas([H|T], R) :-
        demoThreeValued(H, A),
        demoListas(T, B),
        R = [A|B].
```

O predicado demoThreeValued é um predicado que representa as duas tabelas de verdade apresentadas previamente. Recebe dois parâmetros, ou uma conjunção de questões, ou uma disjunção de questões, e a resposta correspondente. A tática utilizada passou por listar todos os casos da tabela de verdade, utilizando o predicado demo que verifica se cada uma das questões é verdadeira, falsa ou desconhecida. De seguida, apresenta-se o predicado demoThreeValued.

```
demoThreeValued((Q1 e Q2), verdadeiro) :- demo(Q1, verdadeiro), demo(Q2,verdadeiro).
demoThreeValued((Q1 e Q2), falso) :- demo(Q1, falso), demo(Q2,verdadeiro).
demoThreeValued((Q1 e Q2), falso) :- demo(Q1, verdadeiro), demo(Q2,falso).
demoThreeValued((Q1 e Q2), falso) :- demo(Q1, desconhecido), demo(Q2,falso).
demoThreeValued((Q1 e Q2), falso) :- demo(Q1, falso), demo(Q2,desconhecido).
demoThreeValued((Q1 e Q2), falso) :- demo(Q1, falso), demo(Q2,falso).
demoThreeValued((Q1 e Q2), desconhecido) :- demo(Q1, desconhecido), demo(Q2,desconhecido).
demoThreeValued((Q1 e Q2), desconhecido) :- demo(Q1, desconhecido), demo(Q2,verdadeiro).
demoThreeValued((Q1 e Q2), desconhecido) :- demo(Q1, verdadeiro), demo(Q2,desconhecido).
demoThreeValued((Q1 ou Q2), verdadeiro) :- demo(Q1, verdadeiro), demo(Q2, verdadeiro).
demoThreeValued((Q1 ou Q2), verdadeiro) :- demo(Q1, verdadeiro), demo(Q2, desconhecido).
demoThreeValued((Q1 ou Q2), verdadeiro) :- demo(Q1, desconhecido), demo(Q2, verdadeiro).
demoThreeValued((Q1 ou Q2), verdadeiro) :- demo(Q1, verdadeiro), demo(Q2, falso).
demoThreeValued((Q1 ou Q2), verdadeiro) :- demo(Q1, falso), demo(Q2, verdadeiro).
demoThreeValued((Q1 ou Q2), falso) :- demo(Q1, falso), demo(Q2, falso).
demoThreeValued((Q1 ou Q2), desconhecido) :- demo(Q1, desconhecido), demo(Q2, desconhecido).
demoThreeValued((Q1 ou Q2), desconhecido) :- demo(Q1, falso), demo(Q2, desconhecido).
demoThreeValued((Q1 ou Q2), desconhecido) :- demo(Q1, desconhecido), demo(Q2, falso).
```

Exemplos Práticos e Análise de Resultados

- **Conhecimento Negativo**

```
| ?-
| ?- listing(-).
-utente(u4,'Rita Sousa',70,'Faro').
-cuidado(s11,e6,'hospital sao joao','Porto').
-atomedico(14,20,14,2,2017,u2,s1,m5,40.9).
-medico(A,B,C,D) :-
        nao(medico(A,B,C,D)),
        nao(excecao(medico(A,B,C,D))).
-especialidade(A,B) :-
        nao(especialidade(A,B)),
        nao(excecao(especialidade(A,B))).
-utente(A,B,C,D) :-
        nao(utente(A,B,C,D)),
        nao(excecao(utente(A,B,C,D))).
-cuidado(A,B,C,D) :-
        nao(cuidado(A,B,C,D)),
        nao(excecao(cuidado(A,B,C,D))).
-atomedico(A,B,C,D,E,F,G,H,I) :-
        nao(atomedico(A,B,C,D,E,F,G,H,I)),
        nao(excecao(atomedico(A,B,C,D,E,F,G,H,I))).

yes
| ?- demo(utente(u4,'Rita Sousa',70,'Faro'),R).
R = falso ?
yes
```

Aqui listou-se o conhecimento negativo na base de conhecimento e quando se interroga sobre um utente com informação negativa, tem-se como resposta “falso”.

- **Conhecimento Imperfeito Incerto**

```
| ?-
| ?- listing(utente).
utente(u1, 'Pedro Pires', 21, 'Braga').
utente(u2, 'Marta Barros', 29, 'Braga').
utente(u3, 'Guilherme Martins', 36, 'Guimaraes').
utente(u8, 'Joana Monteiro', 12, 'Algarve').
utente(u9, 'Manuel Santidade', 23, 'Palmela').
utente(u7, 'John McCarthy', 56, morada_desconhecida).
utente(u12, 'Cristiano Ronaldo', 33, morada_interdita).

yes
| ?- demo(utente(u7,'John McCarthy',56,'Tui'),R).
R = desconhecido ?
yes
```

Neste caso, interrogou-se John McCarthy sobre a sua morada, que é desconhecida, e obtém-se como resposta “desconhecido”, como era esperado.

- **Conhecimento Imperfeito Impreciso**

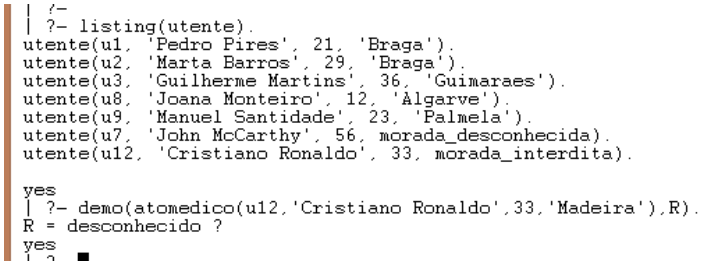
```
| ?- listing(excecao).
excecao(utente(A,B,C,_)) :-
        utente(A, B, C, morada_desconhecida).
excecao(cuidado(A,_,B,C)) :-
        cuidado(A, esp_desconhecida, B, C).
excecao(atomedico(_,_,A,B,C,D,E,F,G)) :-
        atomedico(hora_desconhecida, minutos_desconhecidos, A, B, C, D, E, F, G
).
excecao(utente(u10,'Diana Dias',25,'Viseu')).
excecao(utente(u10,'Diana Dias',26,'Viseu')).
excecao(atomedico(18,20,23,11,2018,u9,s1,m5,A)) :-
        A>=30,
        A=<40.
excecao(atomedico(13,50,17,10,2019,u8,s2,m4,10)).
excecao(atomedico(13,50,17,10,2019,u8,s2,m4,100)).
excecao(utente(A,B,C,_)) :-
        utente(A, B, C, morada_interdita).
excecao(medico(A,B,C,_)) :-
        medico(A, B, C, esp_interdita).
excecao(atomedico(A,B,C,D,E,F,G,H,_)) :-
        atomedico(A, B, C, D, E, F, G, H, custo_interdito).

yes
| ?-
| ?-
| ?-
| ?-
| ?- demo(atomedico(13,50,17,10,2019,u8,s2,m4,100),R).
R = desconhecido ?
yes
| ?- demo(atomedico(13,50,17,10,2019,u8,s2,m4,123),R).
R = falso ?
yes
```

Como se pode verificar, interrogou-se sobre um ato médico que pode ter custo 10 ou 100. Com o custo 100, obteve-se a resposta “desconhecido”. Com o custo 123, obteve-se a resposta “falso”.

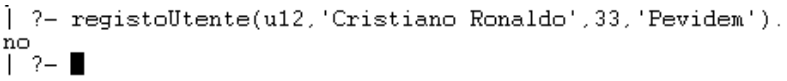
```
| ?- listing(excecao).
excecao(utente(A,B,C,_)) :-
        utente(A, B, C, morada_desconhecida).
excecao(cuidado(A,_,B,C)) :-
        cuidado(A, esp_desconhecida, B, C).
excecao(atomedico(_,_,A,B,C,D,E,F,G)) :-
        atomedico(hora_desconhecida, minutos_desconhecidos, A, B, C, D, E, F, G
).
excecao(utente(u10,'Diana Dias',25,'Viseu')).
excecao(utente(u10,'Diana Dias',26,'Viseu')).
excecao(atomedico(18,20,23,11,2018,u9,s1,m5,A)) :-
        A>=30,
        A=<40.
excecao(atomedico(13,50,17,10,2019,u8,s2,m4,10)).
excecao(atomedico(13,50,17,10,2019,u8,s2,m4,100)).
excecao(utente(A,B,C,_)) :-
        utente(A, B, C, morada_interdita).
excecao(medico(A,B,C,_)) :-
        medico(A, B, C, esp_interdita).
excecao(atomedico(A,B,C,D,E,F,G,H,_)) :-
        atomedico(A, B, C, D, E, F, G, H, custo_interdito).

yes
| ?- demo(atomedico(18,20,23,11,2018,u9,s1,m5,34),R).
R = desconhecido ?
yes
| ?- demo(atomedico(18,20,23,11,2018,u9,s1,m5,342),R).
R = falso ?
yes
| ?-
```

Como se pode verificar, interrogou-se sobre um ato médico que pode ter um custo entre 30 e 40€. Com o custo 34, obteve-se a resposta “desconhecido”. Com o custo 342, obteve-se a resposta “falso”.

- **Conhecimento Imperfeito Interdito**



Neste exemplo, interrogou-se sobre a morada interdita de um utente e obteve-se a resposta “desconhecido”.



Neste exemplo, tentou-se inserir conhecimento positivo sobre conhecimento interdito, logo a inserção não foi efetuada.

Conclusões e Sugestões Futuras

Ao longo da realização deste trabalho prático, foram surgindo algum tipo de dificuldades relacionadas, principalmente, com a representação do conhecimento imperfeito em *Prolog.* Contudo, recorrendo ao material disponibilizado pela equipa docente durante as aulas esses problemas foram solucionados.

Como trabalho futuro seria interessante analisar e estudar formas de incluir novo tipo de conhecimento para que todo o universo de cuidados de saúde fosse incluído na base de conhecimento. Para além disso, seria também interessante incluir novos tipos de desconhecimento para cada predicado.

Contudo, acreditamos que as funcionalidades extra acrescentadas neste trabalho, relativamente ao que foi pedido encontra-se justificado pela sua utilidade, como por exemplo a implementação dos predicados médico e especialidade, aumentando não só a integridade estrutural da base de conhecimento, assim como alargando o leque de possibilidades de conhecimento a ser inserido permitindo responder a questões mais complexas.

Concluindo, acreditamos que os resultados obtidos neste projeto/exercício prático, cumprem os objetivos e requisitos pedidos, conseguindo-se verificar que o nosso sistema desenvolvido é bastante útil e prático.

Bibliografia

Referências Bibliográficas

[Analide, 2011] ANALIDE, César, NOVAIS, Paulo, NEVES, José,
“Sugestões para a Elaboração de Relatórios”, Relatório Técnico, Departamento de Informática, Universidade do Minho, Portugal, 2011.

“SICStus Prolog User’s Manual”,
SICS Swedish ICT AB PO Box 1263 SE-164 29 Kista, Sweden, dezembro 2016

ANALIDE, César, NEVES, José,
“Representação de Informação Incompleta”,
Departamento de Informática, Universidade do Minho, Portugal

Referências WWW

Three Valued Logic.[online]Available at: http://wiki.c2.com/?ThreeValuedLogic [Accessed on 05/04/2017]