UNIVERSIDADE DO MINHO

Exercício 2

Mestrado Integrado em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio

(2º Semestre / 2018-2019)

A81047 Catarina Machado

A86264 Gonçalo Faria

A82339 João Vilaça

Braga,

Abril de 2019

Resumo

O presente relatório é constituído por uma descrição e demonstração do trabalho realizado no âmbito do Exercício 2 da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio.

O objetivo deste projeto é utilizar a programação em lógica estendida e a representação de conhecimento imperfeito, recorrendo à temática dos valores nulos, para construir um sistema de representação de conhecimento e raciocínio com capacidade para caracterizar um universo de discurso na área da prestação de cuidados de saúde pela realização de serviços de atos médicos.

Assim sendo, ao longo deste documento será apresentada a base de conhecimento construída, tanto com conhecimento perfeito positivo e negativo, como com conhecimento imperfeito (de todos os valores nulos estudados, isto é, impreciso, incerto e interdito). Serão também apresentados os invariantes estruturais e referenciais que permitem garantir uma correta evolução e involução da base de conhecimento.

Nas secções seguintes serão ainda analisadas as problemáticas da evolução e involução de conhecimento, ilustradas com os procedimentos criados para o efeito e pequenos exemplos de aplicação e, por fim, será apresentado o sistema de inferência desenvolvido, capaz de implementar os mecanismos de raciocínio inerentes a este sistema.

Índice

Resumo ..... *ii*

Índice ..... *iii*

Índice de Figuras ..... *iv*

Índice de Tabelas ..... *vi*

1. Introdução *7*

2. Preliminares *8*

3. Descrição do Trabalho e Análise de Resultados *10*

3.1. Base de Conhecimento **11**

3.1.1. Conhecimento Perfeito 12

3.1.2. Conhecimento Imperfeito 13

3.2. Invariantes **17**

3.2.1. Invariantes universais 17

3.2.2. Invariantes de utente 18

3.2.3. Invariantes de prestador 20

3.2.4. Invariantes de cuidado 21

3.2.5. Invariantes de exame 22

3.3. Evolução de conhecimento **23**

3.3.1. Evolução de conhecimento perfeito positivo e negativo 23

3.3.2. Evolução de conhecimento imperfeito incerto 25

3.3.3. Evolução de conhecimento imperfeito impreciso 29

3.3.4. Evolução de conhecimento imperfeito interdito 31

3.4. Involução de conhecimento **32**

3.4.1. Involução de conhecimento perfeito positivo e negativo 32

3.4.2. Involução de conhecimento imperfeito incerto 34

3.4.3. Involução de conhecimento imperfeito impreciso 38

3.4.4. Involução de conhecimento imperfeito interdito 39

3.5. Sistema de Inferência **40**

4. Conclusões e Sugestões *44*

5. Referências *45*

5.1. Referências Bibliográficas **45**

5.2. Referências Eletrónicas **46**

6. Anexos *47*

6.1. Predicados Auxiliares **47**

Índice de Figuras

Figura 1 - PMF para utente e prestador 12
Figura 2 - Base de Conhecimento Perfeito Positivo 12
Figura 3 - Base de Conhecimento Perfeito Negativo 13
Figura 4 - Base de Conhecimento Imperfeito Incerto 14
Figura 5 - Base de Conhecimento Imperfeito Impreciso 15
Figura 6 - Base de Conhecimento Imperfeito Interdito 16
Figura 7 - Invariante Conhecimento Repetido 17
Figura 8 - Invariante Conhecimento Contraditório 17
Figura 9 - Invariante Exceções Repetidas 18
Figura 10 - Invariante de utente 18
Figura 11 - Invariante de utente 18
Figura 12 - Invariante de utente 19
Figura 13 - Invariante de utente 19
Figura 14 - Invariante de utente 19
Figura 15 - Invariante de utente 19
Figura 16 - Invariante de prestador 20
Figura 17 - Invariante de prestador 20
Figura 18 - Invariante de prestador 20
Figura 19 - Invariante de prestador 21
Figura 20 - Invariante de prestador 21
Figura 21 - Invariante de cuidado 21
Figura 22 - Invariante de cuidado 21
Figura 23 - Invariante de cuidado 22
Figura 24 - Invariante de exame 22
Figura 25 - Invariante de exame 22
Figura 26 - Evolução de conhecimento 23
Figura 27 - Evolução de conhecimento perfeito positivo 23
Figura 28 - Evolução de conhecimento perfeito negativo 24
Figura 29 - Exemplificação evolução de conhecimento perfeito negativo 24
Figura 30 - Evolução de conhecimento imperfeito incerto de utente 25
Figura 31 - Evolução de conhecimento imperfeito incerto de prestador 26
Figura 32 - Evolução de conhecimento imperfeito incerto de cuidado 27
Figura 33 - Evolução de conhecimento imperfeito incerto de exame 28
Figura 34 - Exemplificação evolução de conhecimento imperfeito incerto 28
Figura 35 - Evolução de conhecimento imperfeito impreciso 29
Figura 36 - Exemplificação evolução de conhecimento imperfeito impreciso 29
Figura 37 - Evolução de conhecimento imperfeito impreciso 30
Figura 38 - Exemplificação evolução de conhecimento imperfeito impreciso 30
Figura 39 - Evolução de conhecimento imperfeito interdito de utente 31
Figura 40 - Exemplificação evolução de conhecimento imperfeito interdito de utente 31

Figura 41 – Meta-predicado 'involucao'.....32
Figura 42 - Involução de conhecimento perfeito positivo 32
Figura 43 - Involução de conhecimento perfeito negativo 33
Figura 44 - Exemplificação involução de conhecimento perfeito positivo 33
Figura 45 - Involução de conhecimento imperfeito incerto de utente.....34
Figura 46 - Involução de conhecimento imperfeito incerto de prestador 35
Figura 47 - Involução de conhecimento imperfeito incerto de cuidado.....36
Figura 48 - Involução de conhecimento imperfeito incerto de exame.....37
Figura 49 - Exemplificação involução de conhecimento imperfeito incerto.....37
Figura 50 - Involução de conhecimento imperfeito impreciso.....38
Figura 51 – Exemplificação involução de conhecimento imperfeito impreciso.....38
Figura 52 - Involução de conhecimento imperfeito interdito de utente 39
Figura 53 - Sistema de Inferência simples.....40
Figura 54 - Sistema de Inferência (lista de questões) 40
Figura 55 - Predicado conjunção 42
Figura 56 - Predicado disjunção.....42
Figura 57 - Conjunção (lista de questões) 42
Figura 58 - Disjunção (lista de questões) 42
Figura 59 - Conjunção/Disjunção (2 questões).....43
Figura 60 - Conjunção e/ou Disjunção (lista de questões).....43
Figura 61 - Meta-predicado 'nao' 47
Figura 62 - Meta-predicado 'solucoes'.....47
Figura 63 - Predicado 'comprimento'.....47
Figura 64 – Meta-predicado 'insercao' 48
Figura 65 – Meta-predicado 'remocao'.....48
Figura 66 – Meta-predicado 'teste'.....48
Figura 67 - Predicado 'idadeValida', 'custoValido' e 'generoValido'.....48

Índice de Tabelas

Tabela 1 - Valores de Verdade Conjunção 41
Tabela 2 - Valores de Verdade Disjunção.....41

1. Introdução

Este exercício teve como principal objetivo motivar-nos para a utilização da programação em lógica estendida, no âmbito da representação de conhecimento perfeito e imperfeito, e para construção de mecanismos de raciocínio para a resolução de problemas.

Durante o exercício foi desenvolvido um sistema para caracterizar um universo de discurso na área da prestação de cuidados de saúde pela realização de serviços de atos médicos que incluía conhecimento de utentes, prestadores, cuidados e exames.

Este sistema tinha como requisitos a implementação de várias funcionalidades, apresentadas ao longo deste relatório, que permitem representar conhecimento perfeito positivo e negativo e casos de conhecimento imperfeito, pela utilização de valores nulos dos tipos estudados, manipular invariantes que designem restrições, quer à inserção quer à remoção de conhecimento do sistema, lidar com a problemática da evolução e da involução de conhecimento e ainda desenvolver um sistema de inferência que implementa os mecanismos de raciocínio necessários.

2. Preliminares

Em primeiro lugar, para o desenvolvimento do segundo exercício foi necessário um conhecimento sólido em relação aos conhecimentos postos à prova no exercício anterior. Desta forma, o paradigma da programação em lógica e o motor de inferência PROLOG são agora raciocínios e ferramentas com que estamos habituados a lidar e a trabalhar.

Porém, após um estudo mais aprofundado em relação à programação em lógica e à respetiva representação de informação fomos confrontados com algumas limitações que não poderíamos deixar passar em claro. Para debatermos essas limitações temos que ter em atenção que este paradigma baseia-se nos seguintes pressupostos: no **Pressuposto do Mundo Fechado (PMF)**, que contempla que toda a informação que não existe mencionada na base de dados é considerada falsa; no **Pressuposto dos Nomes Únicos (PNU)** que garante que duas constantes diferentes (que definem valores atómicos ou objetos) designam, necessariamente, duas entidades diferentes do universo de discurso e, por fim, no **Pressuposto do Domínio Fechado (PDF)**, que indica que não existem mais objetos no universo de discurso para além daqueles designados por constantes na base de dados.

Estes pressupostos foram salvaguardados no exercício anterior, mas aplicando-os no contexto do mundo real podem trazer vários problemas. Por exemplo, num problema simples do dia a dia como a intenção de atravessar uma estrada teremos que ter em atenção se está algum carro a aproximar-se. Para tal, recorrendo à programação em lógica, caso na nossa base de conhecimento não exista informação relativa a carros a aproximarem-se o nosso motor de inferência indicar-nos-á que não se aproxima nenhum automóvel e que, consequentemente, podemos atravessar. Porém, isso não prova que não se aproxima nenhum carro, prova apenas que não há nenhuma prova que se aproximam carros, mas podem estar a aproximar-se.

Desta forma, teremos que abolir alguns desses pressupostos visto que nem sempre se pretende assumir que a informação representada é a única que é válida e, muito menos, que as entidades representadas sejam as únicas existentes no mundo exterior. Assim, o **PNU** continua a poder ser mantido visto não interferir negativamente na representação de conhecimento, mas teremos que deixar de nos guiar pelo **PMF** e pelo **PDF**.

Consequentemente, para substituir esses dois pressupostos adotamos agora o **Pressuposto do Mundo Aberto (PMA)**, que indica que podem existir outros factos ou conclusões verdadeiras para além daqueles representados na base de conhecimento e o **Pressuposto do Domínio Aberto (PDA)**, que garante que podem existir mais objetos do

universo de discurso para além daqueles designados pelas constantes da base de conhecimento. Desta forma, foi feita uma **Extensão à Programação em Lógica**.

Para além dos novos pressupostos mencionados, a Programação em Lógica Estendida infere-nos também uma forma de representar a informação incompleta. Assim, existem agora três diferentes tipos de conclusões para uma dada questão: **verdadeira**, quando existe uma prova explícita de que se trata de conhecimento verdadeiro, **falsa**, quando existe uma prova explícita de que se trata de conhecimento falso e **desconhecido**, quando não existe informação que permita inferir uma ou outra das conclusões anteriores.

Tal como mencionado no tipo de conhecimento **falso**, o objetivo de estender a Programação em Lógica é também passar a ser permitido representar informação negativa explicitamente - **negação forte**. Desta forma, a extensão de um programa em lógica passa agora a contar com dois tipos de negação: a **negação por falha**, que é representada pelo termo *não*, que nos indica que não existe uma prova na base de conhecimento que responda à questão e a **negação forte**, que é uma forma de identificar informação negativa ou falsa, representada pela conectiva "-", que nos indica que existe uma prova na base de conhecimento de que a questão é falsa.

Neste momento, temos então uma Extensão à Programação em Lógica bem fundamentada, com o abandono da PMF e da PDF, com a introdução da negação forte e com a representação de conhecimento imperfeito.

No entanto, na representação de conhecimento imperfeito podemos ser mais detalhados e identificar quais são os valores mais comuns que podem surgir numa situação de informação incompleta, que são designados como **valores nulos**. Assim, existem três tipos de valores nulos mais comuns: **incerto**, quando não sabemos algum dos campos, trata-se de conhecimento desconhecido dentro de um conjunto indeterminado de hipóteses, **impreciso**, quando não temos a certeza de qual das informações é a correta, ou seja, é conhecimento desconhecido mas dentro de um conjunto finito de hipóteses e, por fim, **interdito**, quando não é permitido conhecê-lo.

Em suma, para a elaboração do presente exercício resta-nos agora desenvolver um sistema de inferência e de assimulação de conhecimento que nos permita implementar os mecanismos de raciocínio mencionados, inerentes a estes sistemas.

3. Descrição do Trabalho e Análise de Resultados

Para uma melhor exposição do trabalho realizado, dividimos o mesmo em seis diferentes partes. A primeira parte é a **Base de Conhecimento**, onde descrevemos os predicados que constituem a base de conhecimento do nosso sistema de representação de conhecimento e raciocínio, sejam eles relativos a conhecimento perfeito (positivo e negativo) ou relativos a conhecimento imperfeito (impreciso, incerto e interdito).

Em seguida, apresentamos os **Invariantes** estruturais e referenciais que controlam a inserção e remoção de conhecimento de todos os tipos de predicados criados, tendo em consideração todas as nuances existentes.

Posteriormente, abordamos a problemática da **Evolução de Conhecimento** e da **Involução de Conhecimento**, onde apresentamos os procedimentos construídos para o efeito, devidamente ilustrados com simples exemplos práticos de aplicação.

Por fim, foi construído um **Sistema de Inferência** que contempla os mecanismos de raciocínio adequados para o nosso sistema de Programação em Lógica Estendida e de Representação de Conhecimento Imperfeito. Para complementar esse sistema, foram construídos também predicados que permitem fazer conjunções e/ou disjunções de duas ou mais *queries*.

Em anexo apresentamos os **Predicados Auxiliares**, onde se encontram as funções que nos auxiliaram na construção dos predicados elaborados.

Ao longo da presente secção, fomos descrevendo o trabalho e simultaneamente analisando os resultados obtidos.

3.1. Base de Conhecimento

O nosso sistema de representação de conhecimento relativo à área da prestação de cuidados de saúde pela realização de atos médicos é caracterizado por diferentes predicados que procuram ilustrar este sistema da melhor forma. Desta forma, as nossas fontes de conhecimento são o **utente**, o **prestador**, o **cuidado** e o **exame**, que se encontram definidos da seguinte forma.

---

utente: #IdUt, Nome, Idade, Género, Morada -> {V,F,D}

O utente é identificado por um id único e intransmissível, o seu nome, idade, género (‘M’ ou ‘F’) e pela sua morada.

---

prestador: #IdPrest, Nome, Idade, Género, Especialidade, Instituição -> {V,F,D}

O prestador é caracterizado por um id, nome, idade, género (‘M’ ou ‘F’), pela especialidade que exerce e pela instituição onde trabalha.

---

cuidado: Data, #IdUt, #IdPrest, Descrição, Custo -> {V,F,D}

O cuidado possui a data em que se realizou, o id do utente que compareceu, o id do prestador que prestou o cuidado, uma breve descrição e o custo da mesma.

---

exame: #IdUt, #IdPrest, Descrição -> {V,F,D}

O predicado exame é caracterizado pelo id do utente e pelo id do prestador que estiveram presentes no exame e pela respetiva descrição.

É de salientar que para que possamos representar informação incompleta o domínio das soluções teve que ser alterado. Assim, a conclusão para uma dada questão pode agora ser **verdadeira**, **falsa** ou **desconhecida**.

Como forma de construir uma base de conhecimento ampla e generalizada decidimos incluir informação negativa explicitamente, bem como explicitar diretamente o PMF para alguns predicados. Desta forma, mantivemos o Pressuposto do Mundo Fechado para o predicado **utente** e para o predicado **prestador**, assumindo então que um utente e um prestador não definidos na base de conhecimento nem que contenham alguma exceção associada são considerados conhecimento falso. Esta decisão traz-nos bastante vantagens, por exemplo, no caso da representação de conhecimento imperfeito impreciso, visto que se não soubermos exatamente o valor da idade do utente mas soubermos o intervalo de valores em que se situa podemos afirmar com certezas que o utente com a idade não

contida entre esses valores é **falso**, apesar dessa informação não se encontrar explicitamente na nossa base de conhecimento.

```
-utente(Id,N,I,G,M) :-
    nao(utente(Id,N,I,G,M)),
    nao(excecao(utente(Id,N,I,G,M))).

-prestador(Id,N,I,G,E,Inst) :-
    nao(prestador(Id,N,I,G,E,Inst)),
    nao(excecao(prestador(Id,N,I,G,E,Inst))).
```

Figura 1 - PMF para utente e prestador

Já para o caso do predicado **cuidado** e do predicado **exame**, a ausência do Pressuposto do Mundo Fechado garante que apenas os casos explicitamente negativos na base de conhecimento são considerados falsos.

3.1.1. Conhecimento Perfeito

Tendo o panorama de conhecimento bem definido, decidimos começar por povoar o nosso sistema acrescentando factos perfeitos **positivos** relativos a todos os predicados mencionados anteriormente.

```
% Extensão do predicado utente: #IdUt,Nome,Idade,Género,Morada -> {V,F,D}
utente(1,'Carlos',2,'M','Porto').
utente(2,'Joao',20,'M','Braga').
utente(3,'Vilaca',20,'M','Braga').
utente(4,'Catarina',20,'F','Braga').
utente(5,'Eva',0,'F','Braga').
utente(6,'Maria Jose',56,'F','Guimaraes').
utente(7,'Gustavo',28,'M','Lisboa').
utente(8,'Helena',25,'F','Lisboa').

% Extensão do predicado prestador: #IdPrest,Nome,Idade,Género,Especialidade,Instituição -> {V,F,D}
prestador(1,'Josefa',47,'F','Pediatria','Hospital Santa Maria').
prestador(2,'Manuel',29,'M','Ortopedia','Hospital de Braga').
prestador(3,'Gaspar',38,'M','Dermatologia','Hospital Santa Maria').
prestador(4,'Catia',43,'F','Otorrinolaringologia','Hospital Pulido Valente').
prestador(5,'Marta',38,'F','Cirurgia Geral','Hospital Egas Moniz').
prestador(6,'Goncalo',28,'M','Cardiologia','Hospital de Braga').
prestador(7,'Patricia',39,'F','Genetica Medica','Hospital Pulido Valente').

% Extensão do predicado cuidado: Data,#IdUt,#IdPrest,Descrição,Custo -> {V,F,D}
cuidado('01-01-2018',1,4,'Ouvidos',100).
cuidado('23-05-2016',5,1,'Consulta de rotina',9).
cuidado('05-02-2019',3,5,'Teleconsulta de Cirurgia Geral',45).
cuidado('13-12-2018',8,7,'Consulta Pre-Concepcional',7).
cuidado('26-10-2016',2,2,'Consulta de Ortopedia - Joelho',30).
cuidado('10-03-2016',7,3,'Biopsia da Pele e Mucosas',0).
cuidado('04-04-2017',5,7,'Consulta de Aconselhamento Genetico',34).
cuidado('12-12-2017',3,5,'Manometria Anorretal',4).
cuidado('09-09-2017',6,2,'Consulta de Ortopedia - Coluna',20).

% Extensão do predicado exame: #IdUt,#IdPrest,Descrição -> {V,F,D}
exame(1,1,'Prova de Esforco').
exame(2,2,'Biopsia').
exame(3,3,'Endoscopia').
```

Figura 2 - Base de Conhecimento Perfeito Positivo

No caso de conhecimento perfeito **negativo**, adicionamos factos com **negação forte**. Desta forma, conseguimos ter na nossa base de conhecimento negativo através da negação forte mas também através da negação por falha no caso do utente e do prestador, tal como apresentado na *Figura 1 - PMF para utente e prestador*.

```
% Extensão do predicado utente: #IdUt,Nome,Idade,Género,Morada -> {V,F,D}
-utente(9,'Guilherme',58,'M','Porto').
-utente(10,'Ana',34,'F','Guimaraes').

% Extensão do predicado prestador: #IdPrest,Nome,Idade,Género,Especialidade,Instituição -> {V,F,D}
-prestador(8,'Luisa',40,'F','Oftalmologia','Hospital de Braga').
-prestador(9,'Sara',35,'F','Estomatologia','Hospital Egas Moniz').

% Extensão do predicado cuidado: Data,#IdUt,#IdPrest,Descrição,Custo -> {V,F,D}
-cuidado('01-01-2018',4,8,'Avaliação dos Campos Visuais',10).
-cuidado('29-02-2017',6,6,'Eletrocardiografia',50).
-cuidado('02-04-2017',10,9,'Glandulas salivares',23).

% Extensão do predicado exame: #IdUt,#IdPrest,Descrição -> {V,F,D}
-exame(4,4,'Exame de Sangue').
```

Figura 3 - Base de Conhecimento Perfeito Negativo

3.1.2. Conhecimento Imperfeito

Tal como explicado anteriormente, a Extensão à Programação em Lógica trouxe-nos um novo contradomínio de respostas. Desta forma, temos agora a possibilidade de obter respostas verdadeiras, falsas e **desconhecidas**. As respostas verdadeiras e falsas encontram-se no panorama de conhecimento perfeito enquanto que as respostas desconhecidas são caracterizadas como conhecimento imperfeito. No caso das respostas desconhecidas, tal como já abordado, estudamos três diferentes tipos de valores nulos: o **incerto**, o **impreciso** e o **interdito**.

Como tal, para que a nossa base de conhecimento esteja povoada da forma mais abrangente possível, incluímos também conhecimento desses três valores nulos estudados. Em seguida, explicaremos a forma como cada um desse conhecimento foi caracterizado e representado.

Conhecimento Imperfeito Incerto

O conhecimento imperfeito incerto diz respeito a conhecimento desconhecido de um determinado campo do predicado, dentro de um conjunto ilimitado de hipóteses.

No nosso caso de estudo isso pode ser caracterizado como, por exemplo, saber que existe um utente com id igual a 13, chamado Hélder, com 30 anos, do sexo masculino, mas o qual não se sabe a morada. Neste caso, isto é definido como conhecimento **incerto** visto que não sabemos qual é a morada do utente, nem temos uma "pista" de qual possa ser. Para representar este tipo conhecimento, é necessário criar o utente mencionado com "morada_desconhecida" no campo da morada e criar uma exceção que indique que quando se questionar a base de conhecimento relativamente ao utente com "morada_desconhecida" no campo da morada o resultado deverá ser **desconhecido**.

No caso do prestador, do cuidado e do exame o raciocínio é análogo. Na figura seguinte encontra-se representado um exemplo para cada um dos predicados construídos.

```
% Não se sabe a morada do Helder:
utente(11,'Helder',30,'M',morada_desconhecida).
excecao(utente(Id,N,I,G,M)) :- utente(Id,N,I,G,morada_desconhecida).

% Não se sabe a idade da prestadora Silvia:
prestador(10,'Silvia',idade_desconhecida,'F','Cirurgia Geral','Hospital de Braga').
excecao(prestador(Id,N,I,G,E,Inst)) :- prestador(Id,N,idade_desconhecida,G,E,Inst).

% Não se sabe a descrição da consulta:
cuidado('02-04-2019',3,5,descricao_desconhecida,20).
excecao(cuidado(D,IdUt,IdPrest,Desc,C)) :- cuidado(D,IdUt,IdPrest,descricao_desconhecida,C).

% Não se sabe a descrição do exame:
exame(7,7,descricao_desconhecida).
excecao(exame(IdUt,IdPrest,Desc)) :- exame(IdUt,IdPrest,descricao_desconhecida).
```

Figura 4 - Base de Conhecimento Imperfeito Incerto

É de salientar que para o caso do conhecimento assente na PMF, ou seja, utente e prestador, o conhecimento incerto é tido em consideração, visto a resposta dar **desconhecido** caso a base de conhecimento seja questionada com os campos mencionados, e dar **falso** quando algum dos campos não seja respeitado da forma apresentada na base de conhecimento (por exemplo, perguntar se a morada do Hélder é Braga dá desconhecido, visto a morada ser incerta, mas perguntar se o Hélder tem 31 anos dá falso, visto sabermos de antemão que ele tem 30 anos). Já no caso do conhecimento não assente no PMF, ou seja, no caso do cuidado e do exame, a resposta às questões já seria **desconhecido** caso as exceções não tivessem sido acrescentadas, visto que nestes dois predicados todo o conhecimento que não está presente explicitamente na base de conhecimento é já considerado desconhecido (por exemplo no caso do Hélder daria desconhecido nas duas interrogações).

Conhecimento Imperfeito Impreciso

O conhecimento imperfeito impreciso diz respeito a conhecimento desconhecido de um determinado campo do predicado, dentro de um conjunto limitado de hipóteses.

No presente caso de estudo, temos, por exemplo, a situação de sabermos que existe um prestador com id igual a 11, chamado Rodrigo, com 40 anos de idade, do sexo masculino, que trabalha no Hospital Santa Maria mas que não sabemos se é especialista em Cardiologia ou em Pediatria. De forma a contornar esta situação, através do uso de **exceções**, podemos construir essas duas alternativas de modo a que quando a base de conhecimento for questionada relativamente ao prestador Rodrigo a resposta seja **falso** se a especialidade não for Cardiologia nem Pediatria, e a resposta ser **desconhecido**, caso a base de conhecimento seja questionada com a especialidade igual a Cardiologia ou a Pediatria.

De forma semelhante, ao invés de um valor fixo de alternativas podemos ter também um intervalo fixo de valores. Por exemplo, no caso do predicado utente, temos um utente com id igual a 14, chamada Júpiter, do sexo feminino e de Bragança mas cuja idade é incerta, sabendo somente que se situa entre os 20 e os 30 anos.

Assim, os exemplos criados para o efeito encontram-se definidos na figura seguinte.

```
% Não se sabe se o utente Edgar tem 31 ou 32 anos:
excecao(utente(12,'Edgar',31,'M','Coimbra')).
excecao(utente(12,'Edgar',32,'M','Coimbra')).

% Não se sabe se a utente Soraia é de Leiria ou de Setúbal:
excecao(utente(13,'Soraia',29,'F','Leiria')).
excecao(utente(13,'Soraia',29,'F','Setubal')).

% Não se sabe a idade exata do utente
% apenas se sabe que tem entre 20 e 30 anos:
excecao(utente(14,'Jupiter',I,'F','Braganca')) :- I >= 20, I =< 30.

% Não se sabe se o prestador Rodrigo é especialista de Cardiologia ou Pediatria:
excecao(prestador(11,'Rodrigo',40,'M','Cardiologia','Hospital Santa Maria')).
excecao(prestador(11,'Rodrigo',40,'M','Pediatria','Hospital Santa Maria')).

% Não se sabe se o prestador se chama Flávio ou Fábio:
excecao(prestador(12,'Flávio',29,'M','Dermatologia','Hospital de Braga')).
excecao(prestador(12,'Fábio',29,'M','Dermatologia','Hospital de Braga')).

% Não se sabe se o cuidado foi efetuado no dia 1 ou 2 de março de 2019:
excecao(cuidado('01-03-2019',5,1,'Consulta de rotina',9)).
excecao(cuidado('02-03-2019',5,1,'Consulta de rotina',9)).

% Não se sabe se o exame foi feito pelo prestador 5 ou 6:
excecao(exame(5,5,'Exame a urina')).
excecao(exame(5,6,'Exame a urina')).
```

Figura 5 - Base de Conhecimento Imperfeito Impreciso

É de salientar ainda que apesar de termos criado exemplos para todos os predicados, apenas é sentida uma diferença significativa na resposta por parte do sistema de inferência no caso de predicados assentes no PMF (utente e prestador). Apenas nestes dois casos é que o sistema de inferência dá **falso** como resultado caso a base de conhecimento **não** seja questionada com as alternativas existentes nas exceções construídas, e dá **desconhecido** caso seja questionada com uma das alternativas. No caso do cuidado e do exame, a resposta é sempre **desconhecido** visto que tudo o que não está explicitamente definido como positivo/negativo é que assume um desses valores de verdade, caso contrário, é considerado desconhecido.

Conhecimento Imperfeito Interdito

O conhecimento imperfeito interdito diz respeito a conhecimento desconhecido que não é passível de ser conhecido.

Para a representação deste tipo de conhecimento é necessário ter em consideração diferentes aspetos. Exemplificando para o caso de um utente com morada impossível de conhecer, para além da necessidade de criar um novo utente com as informações do utente, mas com o campo que diz respeito à morada com "morada_impossível", é necessário criar também uma exceção relativa ao campo impossível, para que não seja considerado conhecimento positivo/negativo. Para além disso, é ainda preciso criar uma instância de **nulo interdito** com "morada_impossivel" para que ao construirmos o invariante que impeça a evolução de conhecimento relativo a este utente possamos verificar se se trata de um utente com informação impossível de conhecer e, consequentemente, impossível de ser adicionada.

Para ilustrar o conhecimento imperfeito interdito, criamos um exemplo para o utente e para o prestador, como se pode comprovar através da figura seguinte.

```
% É impossível saber a morada do utente:
utente(15,'Tobias',50,'M',morada_impossivel).
excecao(utente(Id,N,I,G,M)) :- utente(Id,N,I,G,morada_impossivel).
nulointerdito(morada_impossivel).
+utente(Id,N,I,G,M) ::
    (solucoes((Id,N,I,G,M), (utente(15,'Tobias',50,'M',morada_impossivel), nao(nulointerdito(morada_impossivel))), R),
     comprimento(R,0)).


% É impossível saber a idade do prestador:
prestador(13,'Maria Isabel',idade_impossivel,'F','Cardiologia','Hospital Egas Moniz').
excecao(prestador(Id,N,I,G,E,Inst)) :- prestador(Id,N,idade_impossivel,G,E,Inst).
nulointerdito(idade_impossivel).
+prestador(Id,N,I,G,E,Inst) ::
    (solucoes((Id,N,I,G,E,Inst),
        (prestador(13,'Maria Isabel',idade_impossivel,'F','Cardiologia','Hospital Egas Moniz'),
         nao(nulointerdito(idade_impossivel))), R),
     comprimento(R,0)).
```

Figura 6 - Base de Conhecimento Imperfeito Interdito

3.2. Invariantes

Antes de construirmos os procedimentos que permitem a adição e a remoção de conhecimento na base de conhecimento do nosso sistema, construímos invariantes estruturais e referenciais adaptados tanto para os predicados em geral como para todos os predicados contruídos em pormenor.

3.2.1. Invariantes universais

Existem alguns invariantes que podem ser generalizados e, desta forma, todos os predicados que serão inseridos na nossa base de conhecimento terão que obedecer a esses invariantes.

Em primeiro lugar, construímos invariantes que garantem que não existe conhecimento perfeito positivo e conhecimento perfeito negativo repetido na nossa base de conhecimento, ou seja, que não existe conhecimento redundante, tal como se pode ver na figura seguinte.

```
% Invariante que garante que não existe conhecimento
% perfeito positivo repetido
+T :: (solucoes(T, T, R),
        comprimento(R, 1)).

% Invariante que garante que não existe conhecimento
% perfeito negativo repetido
+(-T) :: (solucoes(T, -T, R),
          comprimento(R, 1)).
```

Figura 7 - Invariante Conhecimento Repetido

Em segundo lugar, criamos também invariantes que não permitem que seja adicionado conhecimento perfeito positivo que contradiz conhecimento perfeito negativo presente na base de conhecimento e vice-versa.

Estes dois invariantes, presentes na figura seguinte, são essenciais para que a informação existente seja coerente e não seja contraditória.

```
% Invariante que não permite adicionar conhecimento
% perfeito positivo que contradiz conhecimento perfeito negativo
+T :: nao(-T).

% Invariante que não permite adicionar conhecimento
% perfeito negativo que contradiz conhecimento perfeito positivo
+(-T) :: nao(T).
```

Figura 8 - Invariante Conhecimento Contraditório

Em terceiro lugar, criamos um invariante que não permite que existam exceções repetidas na nossa base de conhecimento.

```
% Invariante que garante que não existem excecoes repetidas
+(excecao(T)) :: (solucoes(T, excecao(T), R),
                  comprimento(R, 1)).
```

Figura 9 - Invariante Exceções Repetidas

3.2.2. Invariantes de utente

No caso específico do utente, aquando da **inserção** de novo conhecimento, a par do exercício anterior, é crucial garantir que o id do utente é único (*Figura 10*), que os utentes que têm ids diferentes não possam ter exatamente a mesma informação nos restantes campos (*Figura 11*), a idade do utente é válida (maior ou igual a 0) (*Figura 12*) e o género do utente é válido ('F' ou 'M') (*Figura 13*), mas neste caso tanto para a adição de conhecimento perfeito positivo como para a adição de conhecimento perfeito negativo.

```
% Invariante que garante que o id de cada utente é único
% para conhecimento perfeito positivo
+utente(Id,N,I,G,M) :: (solucoes(Id, utente(Id,_,_,_,_), R),
                        comprimento(R, 1)).

% Invariante que garante que o id de cada utente é único
% para conhecimento perfeito negativo
+(-utente(Id,N,I,G,M)) :: (solucoes(Id, -utente(Id,_,_,_,_), R),
                           comprimento(R, 1)).
```

Figura 10 - Invariante de utente

```
% Garantir que utentes com ids diferentes têm diferente informação
% para conhecimento perfeito positivo
+utente(Id,N,I,G,M) :: (solucoes((N,I,G,M), utente(_,N,I,G,M), R),
                        comprimento(R, 1)).

% Garantir que utentes com ids diferentes têm diferente informação
% para conhecimento perfeito negativo
+(-utente(Id,N,I,G,M)) :: (solucoes((N,I,G,M), -utente(_,N,I,G,M), R),
                           comprimento(R, 1)).
```

Figura 11 - Invariante de utente

```
% Garantir que a idade do utente é válida (>= 0)
% para conhecimento perfeito positivo
+utente(_,_,I,_,_) :: idadeValida(I).

% Garantir que a idade do utente é válida (>= 0)
% para conhecimento perfeito negativo
+(-utente(_,_,I,_,_)) :: idadeValida(I).
```

Figura 12 - Invariante de utente

```
% Garantir que o género do utente é 'M' ou 'F'
% para conhecimento perfeito positivo
+utente(_,_,_,G,_) :: generoValido(G).

% Garantir que o género do utente é 'M' ou 'F'
% para conhecimento perfeito negativo
+(-utente(_,_,_,G,_)) :: generoValido(G).
```

Figura 13 - Invariante de utente

Para além dos invariantes de inserção mencionados, decidimos construir também um invariante que impede a inserção de conhecimento perfeito positivo relativo a um utente com nome interdito, tal como se pode ver na figura seguinte. Este invariante foi adicionado para posteriormente podermos contruir um predicado de evolução/involução de conhecimento relativo a utentes com nome interdito, que abordaremos em maior detalhe numa das secções seguintes.

```
% Invariante que impede a inserção de conhecimento perfeito positivo relativo
% a um utente com nome interdito (conhecimento imperfeito interdito)
+utente(Id,N,I,G,M) :: (solucoes((Id,Nint,I,G,M), (utente(Id,Nint,I,G,M), nulointerdito(Nint)), R),
                        comprimento(R,0)).
```

Figura 14 - Invariante de utente

No que diz respeito à **remoção** de conhecimento de utente, sentimos a necessidade de construir um invariante que garante que não é possível remover utentes que estejam associados a cuidados (*Figura 15*).

```
% Garantir que não é possível remover um utente com cuidados
-utente(Id,_,_,_,_) :: (solucoes(Id, cuidado(_,Id,_,_,_), R),
                        comprimento(R, 0)).
```

Figura 15 - Invariante de utente

3.2.3. Invariantes de prestador

No caso do prestador, é também necessário garantir, no que diz respeito à **adição** de conhecimento, que o id do prestador é único (*Figura 16*), que os prestadores que têm ids diferentes não possam ter exatamente a mesma informação nos restantes campos (*Figura 17*), a idade do prestador é válida (maior ou igual a 0) (*Figura 18*) e o seu género é também válido ('F' ou 'M') (*Figura 19*), tanto para o caso da adição de conhecimento perfeito positivo como para o caso da adição de conhecimento perfeito negativo.

```
% Invariante que garante que o id de cada prestador é único
% para conhecimento perfeito positivo
+prestador(Id,N,I,G,E,Inst) :: (solucoes(Id, prestador(Id,_,_,_,_,_), R),
                                comprimento(R, 1)).

% Invariante que garante que o id de cada prestador é único
% para conhecimento perfeito negativo
+(-prestador(Id,N,I,G,E,Inst)) :: (solucoes(Id, -prestador(Id,_,_,_,_,_), R),
                                   comprimento(R, 1)).
```

Figura 16 - Invariante de prestador

```
% Garantir que prestadores com ids diferentes têm diferente informação
% para conhecimento perfeito positivo
+prestador(Id,N,I,G,E,Inst) :: (solucoes((N,I,G,E,Inst), prestador(_,N,I,G,E,Inst), R),
                                comprimento(R, 1)).

% Garantir que prestadores com ids diferentes têm diferente informação
% para conhecimento perfeito negativo
+(-prestador(Id,N,I,G,E,Inst)) :: (solucoes((N,I,G,E,Inst), -prestador(_,N,I,G,E,Inst), R),
                                   comprimento(R, 1)).
```

Figura 17 - Invariante de prestador

```
% Garantir que a idade do prestador é válida (>= 0)
% para conhecimento perfeito positivo
+prestador(_,_,I,_,_,_) :: idadeValida(I).

% Garantir que a idade do prestador é válida (>= 0)
% para conhecimento perfeito negativo
+(-prestador(_,_,I,_,_,_)) :: idadeValida(I).
```

Figura 18 - Invariante de prestador

```
% Garantir que o género do prestador é 'M' ou 'F'
% para conhecimento perfeito positivo
+prestador(_,_,_,G,_,_) :: generoValido(G).

% Garantir que o género do prestador é 'M' ou 'F'
% para conhecimento perfeito negativo
+(-prestador(_,_,_,G,_,_)) :: generoValido(G).
```

Figura 19 - Invariante de prestador

Para **remoção** de conhecimento de prestador, é necessário assegurar que não é possível remover um prestador associado a cuidados (*Figura 20*).

```
% Garantir que não é possível remover um prestador com cuidados
-prestador(Id,_,_,_,_,_) :: (solucoes(Id, cuidado(_,_,Id,_,_), R),
                             comprimento(R, 0)).
```

Figura 20 - Invariante de prestador

3.2.4. Invariantes de cuidado

No cuidado, para a **inserção** de conhecimento recorremos também a alguns invariantes. O primeiro garante que o id do utente associado ao cuidado existe na base de conhecimento (*Figura 21*), o segundo garante que o id do prestador associado ao cuidado também existe no nosso sistema (*Figura 22*) e, por fim, o terceiro e o quarto invariante garantem que o custo do cuidado é válido (maior ou igual a 0) (*Figura 23*), tanto para conhecimento perfeito positivo como negativo.

```
% Garantir que o id do utente associado cuidado existe
+cuidado(_,IdUt,_,_,_) :: (solucoes(IdUt, utente(IdUt,_,_,_,_), R),
                           comprimento(R, 1)).
```

Figura 21 - Invariante de cuidado

```
% Garantir que o id do prestador associado cuidado existe
+cuidado(_,_,IdPrest,_,_) :: (solucoes(IdPrest, prestador(IdPrest,_,_,_,_,_), R),
                              comprimento(R, 1)).
```

Figura 22 - Invariante de cuidado

```
% Garantir que o custo de cada cuidado é válido (>= 0)
% para conhecimento perfeito positivo
+cuidado(_,_,_,_,C) :: custoValido(C).

% Garantir que o custo de cada cuidado é válido (>= 0)
% para conhecimento perfeito negativo
+(-cuidado(_,_,_,_,C)) :: custoValido(C).
```

Figura 23 - Invariante de cuidado

3.2.5. Invariantes de exame

Por fim, para o exame, foi necessário construir dois invariantes que garantem que o id do utente e o id do prestador associados ao exame existem, *Figura 24* e *Figura 25*, respetivamente.

```
% Garantir que o id do utente associado ao exame existe
+exame(IdUt,_,_) :: (solucoes(IdUt, utente(IdUt,_,_,_,_), R),
                     comprimento(R, 1)).
```

Figura 24 - Invariante de exame

```
% Garantir que o id do prestador associado ao exame existe
+exame(_,IdPrest,_) :: (solucoes(IdPrest, prestador(IdPrest,_,_,_,_,_), R),
                        comprimento(R, 1)).
```

Figura 25 - Invariante de exame

3.3. Evolução de conhecimento

Para lidar com a problemática da evolução de conhecimento decidimos criar procedimentos específicos para cada um dos tipos de conhecimento existentes, tendo em consideração cada uma das implicações impostas por cada um deles: conhecimento perfeito positivo e negativo, imperfeito incerto, imperfeito impreciso e imperfeito interdito. Nas próximas secções iremos explicar detalhadamente cada um dos procedimentos criados para o efeito bem como exemplos de aplicação dos mesmos.

3.3.1. Evolução de conhecimento perfeito positivo e negativo

Em primeiro lugar, a par do exercício anterior, construímos o meta-predicado **evolução**, que é responsável por adicionar novo conhecimento à base de conhecimento, verificando se todos os invariantes estruturais e referenciais de adição do conhecimento que se pretende adicionar (**T**) continuam verdadeiros após a inserção. Só em caso de sucesso é que o conhecimento fica efetivamente registado, em caso contrário, o conhecimento inicialmente inserido é transparentemente removido.

Este predicado pode ser utilizado para a evolução de conhecimento relativo a, por exemplo, exceções, nulos interditos, entre muitos outros predicados.

```
% Insere novo conhecimento na base de conhecimento
evolucao(T) :-
    solucoes(I, +T::I, Linv),
    insercao(T),
    teste(Linv).
```

Figura 26 - Evolução de conhecimento

Para a evolução de conhecimento perfeito positivo criamos um predicado que na essência é igual ao predicado presente na *Figura 26*, mas que foi criado especificamente para a evolução de conhecimento perfeito **positivo**.

```
% Insere conhecimento perfeito positivo na base de conhecimento
evolucao(T, positivo) :-
    solucoes(I, +T::I, Linv),
    insercao(T),
    teste(Linv).
```

Figura 27 - Evolução de conhecimento perfeito positivo

Para a evolução de conhecimento perfeito negativo criamos o predicado presente na figura seguinte, que é similar ao da *Figura 27*, mas desta vez é necessário especificar que a intenção é adicionar conhecimento **negativo**. Assim, deverão ser averiguados os invariantes que dizem respeito à inserção de conhecimento negativo forte e, em seguida, deve ser inserido na base de conhecimento o termo **T**, mas como negativo forte (**-T**).

```
% Insere conhecimento perfeito negativo na base de conhecimento
evolucao(T, negativo) :-
    solucoes(I, +(-T)::I, Linv),
    insercao(-T),
    teste(Linv).
```

Figura 28 - Evolução de conhecimento perfeito negativo

Na figura seguinte encontra-se um exemplo da evolução de conhecimento negativo relativo a **exames**.

```
| ?- listing.
-exame(4,4,'Exame de Sangue').
-utente(A,B,C,D,E) :-
        nao(utente(A,B,C,D,E)),
        nao(excecao(utente(A,B,C,D,E))).
-prestador(A,B,C,D,E,F) :-
        nao(prestador(A,B,C,D,E,F)),
        nao(excecao(prestador(A,B,C,D,E,F))).

exame(1, 1, 'Prova de Esforco').

yes
| ?- evolucao(exame(3,3,'Endoscopia'),negativo).
yes
| ?- listing.
-exame(4,4,'Exame de Sangue').
-utente(A,B,C,D,E) :-
        nao(utente(A,B,C,D,E)),
        nao(excecao(utente(A,B,C,D,E))).
-prestador(A,B,C,D,E,F) :-
        nao(prestador(A,B,C,D,E,F)),
        nao(excecao(prestador(A,B,C,D,E,F))).
-exame(3,3,'Endoscopia').

exame(1, 1, 'Prova de Esforco').

yes
```

Figura 29 - Exemplificação evolução de conhecimento perfeito negativo

3.3.2. Evolução de conhecimento imperfeito incerto

O objetivo é que o nosso sistema de representação de conhecimento e raciocínio permita a adição de conhecimento imperfeito do tipo incerto, para que, por exemplo, caso se saiba todos os dados relativos a um prestador mas não se sabe a sua idade essa informação possa ser registada na base de conhecimento.

Para a criação de procedimentos adequados para a evolução do conhecimento incerto foi necessário, numa primeira fase, analisar a forma como esse conhecimento tem que ser escrito para poder ser representado como tal. Como anteriormente abordado em *Conhecimento Imperfeito Incerto*, é então necessário adicionar o predicado pretendido com o campo em falta como "desconhecido" e criar uma exceção associada a esse mesmo campo.

Consequentemente, uma vez que temos quatro diferentes predicados, cada um deles com diferentes possibilidades de campos incertos, foi necessário criar um predicado evolução para cada um deles, tendo como argumentos o predicado efetivo que se pretende adicionar, o tipo de predicado em questão (utente, prestador, cuidado ou exame), o tipo de conhecimento (incerto, neste caso) e, por fim, indicar qual é o campo desconhecido.

Para o caso do utente, os procedimentos criados encontram-se na *Figura 30*, para o prestador na *Figura 31*, para o cuidado na *Figura 32* e, por fim, para o exame na *Figura 33*.

Utente

```
% Insere conhecimento imperfeito incerto na base de conhecimento
% no caso do utente com nome desconhecido
evolucao(utente(Id,Nome_desconhecido,I,G,M), utente, incerto, nome) :-
    evolucao(utente(Id,Nome_desconhecido,I,G,M), positivo),
    insercao((excecao(utente(IdUt,Nome,Idade,Genero,Morada)) :-
                utente(IdUt,Nome_desconhecido,Idade,Genero,Morada))).

% Insere conhecimento imperfeito incerto na base de conhecimento
% no caso do utente com idade desconhecida
evolucao(utente(Id,N,Idade_desconhecida,G,M), utente, incerto, idade) :-
    evolucao(utente(Id,N,Idade_desconhecida,G,M), positivo),
    insercao((excecao(utente(IdUt,Nome,Idade,Genero,Morada)) :-
                utente(IdUt,Nome,Idade_desconhecida,Genero,Morada))).

% Insere conhecimento imperfeito incerto na base de conhecimento
% no caso do utente com genero desconhecido
evolucao(utente(Id,N,I,Genero_desconhecido,M), utente, incerto, genero) :-
    evolucao(utente(Id,N,I,Genero_desconhecido,M), positivo),
    insercao((excecao(utente(IdUt,Nome,Idade,Genero,Morada)) :-
                utente(IdUt,Nome,Idade,Genero_desconhecido,Morada))).

% Insere conhecimento imperfeito incerto na base de conhecimento
% no caso do utente com morada desconhecida
evolucao(utente(Id,N,I,G,Morada_desconhecida), utente, incerto, morada) :-
    evolucao(utente(Id,N,I,G,Morada_desconhecida), positivo),
    insercao((excecao(utente(IdUt,Nome,Idade,Genero,Morada)) :-
                utente(IdUt,Nome,Idade,Genero,Morada_desconhecida))).
```

Figura 30 - Evolução de conhecimento imperfeito incerto de utente

Prestador

```
% Insere conhecimento imperfeito incerto na base de conhecimento
% no caso do prestador com nome desconhecido
evolucao(prestador(Id,Nome_desconhecido,I,G,E,Inst), prestador, incerto, nome) :-
    evolucao(prestador(Id,Nome_desconhecido,I,G,E,Inst), positivo),
    insercao((excecao(prestador(IdPrest,Nome,Idade,Genero,Especialidade,Instituicao)) :-
                prestador(IdPrest,Nome_desconhecido,Idade,Genero,Especialidade,Instituicao))).

% Insere conhecimento imperfeito incerto na base de conhecimento
% no caso do prestador com idade desconhecida
evolucao(prestador(Id,N,Idade_desconhecida,G,E,Inst), prestador, incerto, idade) :-
    evolucao(prestador(Id,N,Idade_desconhecida,G,E,Inst), positivo),
    insercao((excecao(prestador(IdPrest,Nome,Idade,Genero,Especialidade,Instituicao)) :-
                prestador(IdPrest,Nome,Idade_desconhecida,Genero,Especialidade,Instituicao))).

% Insere conhecimento imperfeito incerto na base de conhecimento
% no caso do prestador com genero desconhecido
evolucao(prestador(Id,N,I,Genero_desconhecido,E,Inst), prestador, incerto, genero) :-
    evolucao(prestador(Id,N,I,Genero_desconhecido,E,Inst), positivo),
    insercao((excecao(prestador(IdPrest,Nome,Idade,Genero,Especialidade,Instituicao)) :-
                prestador(IdPrest,Nome,Idade,Genero_desconhecido,Especialidade,Instituicao))).

% Insere conhecimento imperfeito incerto na base de conhecimento
% no caso do prestador com especialidade desconhecida
evolucao(prestador(Id,N,I,G,Especialidade_desconhecida,Inst), prestador, incerto, especialidade) :-
    evolucao(prestador(Id,N,I,G,Especialidade_desconhecida,Inst), positivo),
    insercao((excecao(prestador(IdPrest,Nome,Idade,Genero,Especialidade,Instituicao)) :-
                prestador(IdPrest,Nome,Idade,Genero,Especialidade_desconhecida,Instituicao))).

% Insere conhecimento imperfeito incerto na base de conhecimento
% no caso do prestador com instituicao desconhecida
evolucao(prestador(Id,N,I,G,E,Instituicao_desconhecida), prestador, incerto, instituicao) :-
    evolucao(prestador(Id,N,I,G,E,Instituicao_desconhecida), positivo),
    insercao((excecao(prestador(IdPrest,Nome,Idade,Genero,Especialidade,Instituicao)) :-
                prestador(IdPrest,Nome,Idade,Genero,Especialidade,Instituicao_desconhecida))).
```

Figura 31 - Evolução de conhecimento imperfeito incerto de prestador

Cuidado

```
% Insere conhecimento imperfeito incerto na base de conhecimento
% no caso do cuidado com data desconhecida
evolucao(cuidado(Data_desconhecida,IdUt,IdPrest,Desc,C), cuidado, incerto, data) :-
    evolucao(cuidado(Data_desconhecida,IdUt,IdPrest,Desc,C), positivo),
    insercao((excecao(cuidado(Data,IdUtente,IdPrestador,Descricao,Custo)) :-
                cuidado(Data_desconhecida,IdUtente,IdPrestador,Descricao,Custo))).

% Insere conhecimento imperfeito incerto na base de conhecimento
% no caso do cuidado com id de utente desconhecido
evolucao(cuidado(D,IdUtente_desconhecido,IdPrest,Desc,C), cuidado, incerto, idutente) :-
    evolucao(cuidado(D,IdUtente_desconhecido,IdPrest,Desc,C), positivo),
    insercao((excecao(cuidado(Data,IdUtente,IdPrestador,Descricao,Custo)) :-
                cuidado(Data,IdUtente_desconhecido,IdPrestador,Descricao,Custo))).

% Insere conhecimento imperfeito incerto na base de conhecimento
% no caso do cuidado com id de prestador desconhecido
evolucao(cuidado(D,IdUt,IdPrestador_desconhecido,Desc,C), cuidado, incerto, idprestador) :-
    evolucao(cuidado(D,IdUt,IdPrestador_desconhecido,Desc,C), positivo),
    insercao((excecao(cuidado(Data,IdUtente,IdPrestador,Descricao,Custo)) :-
                cuidado(Data,IdUtente,IdPrestador_desconhecido,Descricao,Custo))).

% Insere conhecimento imperfeito incerto na base de conhecimento
% no caso do cuidado com descrição desconhecida
evolucao(cuidado(D,IdUt,IdPrest,Descricao_desconhecida,C), cuidado, incerto, descricao) :-
    evolucao(cuidado(D,IdUt,IdPrest,Descricao_desconhecida,C), positivo),
    insercao((excecao(cuidado(Data,IdUtente,IdPrestador,Descricao,Custo)) :-
                cuidado(Data,IdUtente,IdPrestador,Descricao_desconhecida,Custo))).

% Insere conhecimento imperfeito incerto na base de conhecimento
% no caso do cuidado com custo desconhecido
evolucao(cuidado(D,IdUt,IdPrest,Desc,Custo_desconhecido), cuidado, incerto, custo) :-
    evolucao(cuidado(D,IdUt,IdPrest,Desc,Custo_desconhecido), positivo),
    insercao((excecao(cuidado(Data,IdUtente,IdPrestador,Descricao,Custo)) :-
                cuidado(Data,IdUtente,IdPrestador,Descricao,Custo_desconhecido))).
```

Figura 32 - Evolução de conhecimento imperfeito incerto de cuidado

Exame

```
% Insere conhecimento imperfeito incerto na base de conhecimento
% no caso do cuidado com id de utente desconhecido
evolucao(exame(IdUtente_desconhecido,IdPrest,Desc), exame, incerto, idutente) :-
    evolucao(exame(IdUtente_desconhecido,IdPrest,Desc), positivo),
    insercao((excecao(exame(IdUtente,IdPrestador,Descricao)) :-
                exame(IdUtente_desconhecido,IdPrestador,Descricao))).

% Insere conhecimento imperfeito incerto na base de conhecimento
% no caso do exame com id de prestador desconhecido
evolucao(exame(IdUt,IdPrestador_desconhecido,Desc), exame, incerto, idprestador) :-
    evolucao(exame(IdUt,IdPrestador_desconhecido,Desc), positivo),
    insercao((excecao(exame(IdUtente,IdPrestador,Descricao)) :-
                exame(IdUtente,IdPrestador_desconhecido,Descricao))).

% Insere conhecimento imperfeito incerto na base de conhecimento
% no caso do exame com descrição desconhecida
evolucao(exame(IdUt,IdPrest,Descricao_desconhecida), exame, incerto, descricao) :-
    evolucao(exame(IdUt,IdPrest,Descricao_desconhecida), positivo),
    insercao((excecao(exame(IdUtente,IdPrestador,Descricao)) :-
                exame(IdUtente,IdPrestador,Descricao_desconhecida))).
```

Figura 33 - Evolução de conhecimento imperfeito incerto de exame

Na figura seguinte encontra-se um exemplo de inserção de conhecimento de utente incerto, no caso da morada ser desconhecida.

```
| ?- listing.
-utente(A,B,C,D,E) :-
        nao(utente(A,B,C,D,E)),
        nao(excecao(utente(A,B,C,D,E))).
-prestador(A,B,C,D,E,F) :-
        nao(prestador(A,B,C,D,E,F)),
        nao(excecao(prestador(A,B,C,D,E,F))).

yes
| ?- evolucao(utente(11,'Helder',30,'M',morada_desconhecida),utente,incerto,morada).
yes
| ?- listing.
-utente(A,B,C,D,E) :-
        nao(utente(A,B,C,D,E)),
        nao(excecao(utente(A,B,C,D,E))).
-prestador(A,B,C,D,E,F) :-
        nao(prestador(A,B,C,D,E,F)),
        nao(excecao(prestador(A,B,C,D,E,F))).

excecao(utente(A,B,C,D,_)) :-
        utente(A, B, C, D, morada_desconhecida).

utente(11, 'Helder', 30, 'M', morada_desconhecida).

yes
```

Figura 34 - Exemplificação evolução de conhecimento imperfeito incerto

3.3.3. Evolução de conhecimento imperfeito impreciso

O nosso sistema de representação de conhecimento e raciocínio permite também a inserção de conhecimento imperfeito impreciso, no caso de, por exemplo, não se ter a certeza se um utente tem 10 ou 11 anos mas saber-se todas as restantes informações. Para a construção de procedimentos que satisfaçam este requisito foi necessário inicialmente analisar como é que este conhecimento tem que ser escrito para ser representado como impreciso na nossa base de conhecimento.

Tal como abordado em *Conhecimento Imperfeito Impreciso*, para adicionar conhecimento impreciso é somente necessário adicionar exceções para cada uma dessas imprecisões. O procedimento criado para o efeito, presente na *Figura 35*, pode ser utilizado para qualquer um dos tipos de predicados existentes no nosso sistema.

Para utilizar este procedimento basta invocá-lo com o predicado em questão e a palavra **impreciso** como argumento. O que este procedimento faz é averiguar se os invariantes relativos a exceções permanecem verdadeiros depois do conhecimento ser inserido. Em caso afirmativo, o conhecimento permanece na base de conhecimento.

```
% Insere conhecimento imperfeito impreciso na base de conhecimento
% seja de utente, prestador, cuidado ou exame
evolucao(T, impreciso) :-
    solucoes(I, +(excecao(T))::I, Lint),
    insercao(excecao(T)),
    teste(Lint).
```

Figura 35 - Evolução de conhecimento imperfeito impreciso

Na figura seguinte encontra-se um exemplo de aplicação deste procedimento, para o caso de não se saber se a Soraia é de Leiria ou de Setúbal.

```
| ?- listing.
-utente(A,B,C,D,E) :-
        nao(utente(A,B,C,D,E)),
        nao(excecao(utente(A,B,C,D,E))).
-prestador(A,B,C,D,E,F) :-
        nao(prestador(A,B,C,D,E,F)),
        nao(excecao(prestador(A,B,C,D,E,F))).

yes
| ?- evolucao(utente(13,'Soraia',29,'F','Leiria'),impreciso).
yes
| ?- evolucao(utente(13,'Soraia',29,'F','Setubal'),impreciso).
yes
| ?- listing.
-utente(A,B,C,D,E) :-
        nao(utente(A,B,C,D,E)),
        nao(excecao(utente(A,B,C,D,E))).
-prestador(A,B,C,D,E,F) :-
        nao(prestador(A,B,C,D,E,F)),
        nao(excecao(prestador(A,B,C,D,E,F))).

excecao(utente(13,'Soraia',29,'F','Leiria')).
excecao(utente(13,'Soraia',29,'F','Setubal')).

yes
```

Figura 36 - Exemplificação evolução de conhecimento imperfeito impreciso

No caso de conhecimento impreciso, existe também a possibilidade de adicionar conhecimento com um campo desconhecido entre um intervalo finito de valores, por exemplo, no caso da idade do prestador se encontrar situada entre 30 e 40 anos.

Para este efeito, foram construídos outros predicados de evolução, tal como se pode ver na figura seguinte. Para a utilização destes procedimentos, é necessário indicar o termo **T** que se pretende adicionar, o tipo de predicado em questão, a palavra **impreciso**, o campo desconhecido e o limite inferior e superior, respetivamente, do intervalo de valores. No nosso caso, decidimos então criar procedimentos relativos a imprecisão das idades de utentes e de prestadores.

```
% Insere conhecimento imperfeito impreciso na base de conhecimento
% no caso de utente com idade contida dentro de um intervalo finito de valores
evolucao(utente(Id,N,Idade_imprecisa,G,M), utente, impreciso, idade, LimiteInferior, LimiteSuperior) :-
    insercao((excecao(utente(Id,N,Idade_imprecisa,G,M)) :-
                Idade_imprecisa >= LimiteInferior, Idade_imprecisa =< LimiteSuperior)).

% Insere conhecimento imperfeito impreciso na base de conhecimento
% no caso de prestador com idade contida dentro de um intervalo finito de valores
evolucao(prestador(Id,N,Idade_imprecisa,G,E,Inst), prestador, impreciso, idade, LimiteInferior, LimiteSuperior) :-
    insercao((excecao(prestador(Id,N,Idade_imprecisa,G,E,Inst)) :-
                Idade_imprecisa >= LimiteInferior, Idade_imprecisa =< LimiteSuperior)).
```

Figura 37 - Evolução de conhecimento imperfeito impreciso

Na figura abaixo encontra-se apresentado um exemplo de aplicação do procedimento, no que diz respeito a um prestador com id igual a 7, com idade compreendida entre 20 e 30.

```
| ?- listing.
-utente(A,B,C,D,E) :-
        nao(utente(A,B,C,D,E)),
        nao(excecao(utente(A,B,C,D,E))).
-prestador(A,B,C,D,E,F) :-
        nao(prestador(A,B,C,D,E,F)),
        nao(excecao(prestador(A,B,C,D,E,F))).

yes
| ?- evolucao(prestador(7,'Patricia',I,'F','Genetica Medica','Hospital Pulido Valente'),prestador,impreciso,idade,20,30).
yes
| ?- listing.
-utente(A,B,C,D,E) :-
        nao(utente(A,B,C,D,E)),
        nao(excecao(utente(A,B,C,D,E))).
-prestador(A,B,C,D,E,F) :-
        nao(prestador(A,B,C,D,E,F)),
        nao(excecao(prestador(A,B,C,D,E,F))).

excecao(prestador(7,'Patricia',A,'F','Genetica Medica','Hospital Pulido Valente')) :-
        A>=20,
        A=<30.

yes
```

Figura 38 - Exemplificação evolução de conhecimento imperfeito impreciso

3.3.4. Evolução de conhecimento imperfeito interdito

Uma vez que a evolução de conhecimento imperfeito interdito é mais sensível, decidimos criar apenas um exemplo de inserção do mesmo. Para tal, foi essencial analisar a forma como esse conhecimento tem que ser escrito, apresentada em *Conhecimento Imperfeito Interdito*. Assim, criamos um procedimento que permite a inserção de conhecimento interdito no caso de um utente com nome interdito. Para isso, foi essencial a existência prévia do invariante presente na *Figura 14* no nosso sistema, de modo a que seja impossível adicionar a informação interdita posteriormente na base de conhecimento.

Consequentemente, o procedimento criado para o efeito encontra-se na figura seguinte.

```
% Insere conhecimento imperfeito interdito na base de conhecimento
% no caso de utente com nome interdito
evolucao(utente(Id,Nome_impossivel,I,G,M), utente, interdito, nome) :-
    evolucao(utente(Id,Nome_impossivel,I,G,M), positivo),
    insercao((excecao(utente(IdUt,Nome,Idade,Genero,Morada)) :-
                utente(IdUt,Nome_impossivel,Idade,Genero,Morada))),
    insercao((nulointerdito(Nome_impossivel))).
```

Figura 39 - Evolução de conhecimento imperfeito interdito de utente

Tal como se pode verificar através da figura abaixo, os requisitos para a construção deste predicado são corretamente cumpridos.

```
| ?- evolucao(utente(15,nome_impossivel,50,'M','Braga'),utente,interdito,nome).
yes
| ?- listing.
-utente(A,B,C,D,E) :-
        nao(utente(A,B,C,D,E)),
        nao(excecao(utente(A,B,C,D,E))).
-prestador(A,B,C,D,E,F) :-
        nao(prestador(A,B,C,D,E,F)),
        nao(excecao(prestador(A,B,C,D,E,F))).

excecao(utente(A,_,B,C,D)) :-
        utente(A, nome_impossivel, B, C, D).

nulointerdito(nome_impossivel).

utente(15, nome_impossivel, 50, 'M', 'Braga').

yes
| ?- evolucao(utente(15,'Sonia',50,'M','Braga'),positivo).
no
| ?- listing.
-utente(A,B,C,D,E) :-
        nao(utente(A,B,C,D,E)),
        nao(excecao(utente(A,B,C,D,E))).
-prestador(A,B,C,D,E,F) :-
        nao(prestador(A,B,C,D,E,F)),
        nao(excecao(prestador(A,B,C,D,E,F))).

excecao(utente(A,_,B,C,D)) :-
        utente(A, nome_impossivel, B, C, D).

nulointerdito(nome_impossivel).

utente(15, nome_impossivel, 50, 'M', 'Braga').

yes
```

Figura 40 - Exemplificação evolução de conhecimento imperfeito interdito de utente

3.4. Involução de conhecimento

Para lidar com a problemática da involução de conhecimento o raciocínio adotado foi análogo ao da problemática da evolução de conhecimento. Assim, decidimos criar procedimentos específicos para cada um dos tipos de conhecimento existentes, tendo em consideração cada uma das implicações impostas por cada um deles: conhecimento perfeito positivo e negativo, imperfeito incerto, imperfeito impreciso e imperfeito interdito.

3.4.1. Involução de conhecimento perfeito positivo e negativo

Para a involução foi necessário também criar inicialmente o meta-predicado **involução**, já utilizado no primeiro exercício, que é responsável por remover conhecimento da base de conhecimento, verificando se todos os invariantes estruturais e referenciais de remoção do conhecimento que se pretende remover (**T**) continuam verdadeiros. Só em caso de sucesso é que o conhecimento é permanentemente removido, em caso contrário, o conhecimento é novamente inserido.

```
% Retira conhecimento da base de conhecimento
involucao(T) :- solucoes(I, -T::I, Linv),
                remocao(T),
                teste(Linv).
```

Figura 41 – Meta-predicado 'involucao'

Em seguida, apresentamos o predicado utilizado para a involução de conhecimento perfeito **positivo**, na *Figura 42*, e o predicado utilizado para a involução de conhecimento perfeito **negativo**, *Figura 43*.

```
% Retira conhecimento perfeito positivo na base de conhecimento
involucao(T, positivo) :-
    solucoes(I, -T::I, Linv),
    remocao(T),
    teste(Linv).
```

Figura 42 - Involução de conhecimento perfeito positivo

```
% Retira conhecimento perfeito negativo na base de conhecimento
involucao(T, negativo) :-
    solucoes(I, -(-T)::I, Linv),
    remocao(-T),
    teste(Linv).
```

Figura 43 - Involução de conhecimento perfeito negativo

Na figura seguinte encontra-se um exemplo de utilização do predicado involução para o caso da remoção de conhecimento positivo.

```
| ?- listing.
-utente(A,B,C,D,E) :-
        nao(utente(A,B,C,D,E)),
        nao(excecao(utente(A,B,C,D,E))).
-prestador(A,B,C,D,E,F) :-
        nao(prestador(A,B,C,D,E,F)),
        nao(excecao(prestador(A,B,C,D,E,F))).

utente(1, 'Carlos', 2, 'M', 'Porto').

yes
| ?- involucao(utente(1,'Carlos',2,'M','Porto'),positivo).
yes
| ?- listing.
-utente(A,B,C,D,E) :-
        nao(utente(A,B,C,D,E)),
        nao(excecao(utente(A,B,C,D,E))).
-prestador(A,B,C,D,E,F) :-
        nao(prestador(A,B,C,D,E,F)),
        nao(excecao(prestador(A,B,C,D,E,F))).

yes
```

Figura 44 - Exemplificação involução de conhecimento perfeito positivo

Tal como mencionado anteriormente, o raciocínio utilizado para a involução de conhecimento foi o mesmo raciocínio utilizado para a evolução. Desta forma, como os procedimentos criados são meramente espelhos dos procedimentos de inserção, mas neste caso utilizando os predicados auxiliares de remoção, vamos somente apresentar os procedimentos criados para a involução de conhecimento imperfeito não entrando em mais detalhe. Vamos apresentar também alguns exemplos de aplicação desses procedimentos de forma a comprovar a sua eficácia.

3.4.2. Involução de conhecimento imperfeito incerto

Para o caso da involução de conhecimento imperfeito foram então criados diferentes procedimentos para cada um dos tipos de predicados existentes no nosso sistema.

Utente

```
% Retira conhecimento imperfeito incerto na base de conhecimento
% no caso do utente com nome desconhecido
involucao(utente(Id,Nome_desconhecido,I,G,M), utente, incerto, nome) :-
    involucao(utente(Id,Nome_desconhecido,I,G,M), positivo),
    remocao((excecao(utente(IdUt,Nome,Idade,Genero,Morada)) :-
                utente(IdUt,Nome_desconhecido,Idade,Genero,Morada))).

% Retira conhecimento imperfeito incerto na base de conhecimento
% no caso do utente com morada desconhecida
involucao(utente(Id,N,I,G,Morada_desconhecida), utente, incerto, morada) :-
    involucao(utente(Id,N,I,G,Morada_desconhecida), positivo),
    remocao((excecao(utente(IdUt,Nome,Idade,Genero,Morada)) :-
                utente(IdUt,Nome,Idade,Genero,Morada_desconhecida))).

% Retira conhecimento imperfeito incerto na base de conhecimento
% no caso do utente com idade desconhecida
involucao(utente(Id,N,Idade_desconhecida,G,M), utente, incerto, idade) :-
    involucao(utente(Id,N,Idade_desconhecida,G,M), positivo),
    remocao((excecao(utente(IdUt,Nome,Idade,Genero,Morada)) :-
                utente(IdUt,Nome,Idade_desconhecida,Genero,Morada))).

% Retira conhecimento imperfeito incerto na base de conhecimento
% no caso do utente com genero desconhecido
involucao(utente(Id,N,I,Genero_desconhecido,M), utente, incerto, genero) :-
    involucao(utente(Id,N,I,Genero_desconhecido,M), positivo),
    remocao((excecao(utente(IdUt,Nome,Idade,Genero,Morada)) :-
                utente(IdUt,Nome,Idade,Genero_desconhecido,Morada))).
```

Figura 45 - Involução de conhecimento imperfeito incerto de utente

Prestador

```
% Retira conhecimento imperfeito incerto na base de conhecimento
% no caso do prestador com nome desconhecido
involucao(prestador(Id,Nome_desconhecido,I,G,E,Inst), prestador, incerto, nome) :-
    involucao(prestador(Id,Nome_desconhecido,I,G,E,Inst), positivo),
    remocao((excecao(prestador(IdPrest,Nome,Idade,Genero,Especialidade,Instituicao)) :-
                prestador(IdPrest,Nome_desconhecido,Idade,Genero,Especialidade,Instituicao))).

% Retira conhecimento imperfeito incerto na base de conhecimento
% no caso do prestador com idade desconhecida
involucao(prestador(Id,N,Idade_desconhecida,G,E,Inst), prestador, incerto, idade) :-
    involucao(prestador(Id,N,Idade_desconhecida,G,E,Inst), positivo),
    remocao((excecao(prestador(IdPrest,Nome,Idade,Genero,Especialidade,Instituicao)) :-
                prestador(IdPrest,Nome,Idade_desconhecida,Genero,Especialidade,Instituicao))).

% Retira conhecimento imperfeito incerto na base de conhecimento
% no caso do prestador com genero desconhecido
involucao(prestador(Id,N,I,Genero_desconhecido,E,Inst), prestador, incerto, genero) :-
    involucao(prestador(Id,N,I,Genero_desconhecido,E,Inst), positivo),
    remocao((excecao(prestador(IdPrest,Nome,Idade,Genero,Especialidade,Instituicao)) :-
                prestador(IdPrest,Nome,Idade,Genero_desconhecido,Especialidade,Instituicao))).

% Retira conhecimento imperfeito incerto na base de conhecimento
% no caso do prestador com especialidade desconhecida
involucao(prestador(Id,N,I,G,Especialidade_desconhecida,Inst), prestador, incerto, especialidade) :-
    involucao(prestador(Id,N,I,G,Especialidade_desconhecida,Inst), positivo),
    remocao((excecao(prestador(IdPrest,Nome,Idade,Genero,Especialidade,Instituicao)) :-
                prestador(IdPrest,Nome,Idade,Genero,Especialidade_desconhecida,Instituicao))).

% Retira conhecimento imperfeito incerto na base de conhecimento
% no caso do prestador com instituicao desconhecida
involucao(prestador(Id,N,I,G,E,Instituicao_desconhecida), prestador, incerto, instituicao) :-
    involucao(prestador(Id,N,I,G,E,Instituicao_desconhecida), positivo),
    remocao((excecao(prestador(IdPrest,Nome,Idade,Genero,Especialidade,Instituicao)) :-
                prestador(IdPrest,Nome,Idade,Genero,Especialidade,Instituicao_desconhecida))).
```

Figura 46 - Involução de conhecimento imperfeito incerto de prestador

Cuidado

```
% Retira conhecimento imperfeito incerto na base de conhecimento
% no caso do cuidado com data desconhecida
involucao(cuidado(Data_desconhecida,IdUt,IdPrest,Desc,C), cuidado, incerto, data) :-
    involucao(cuidado(Data_desconhecida,IdUt,IdPrest,Desc,C), positivo),
    remocao((excecao(cuidado(Data,IdUtente,IdPrestador,Descricao,Custo)) :-
                cuidado(Data_desconhecida,IdUtente,IdPrestador,Descricao,Custo))).

% Retira conhecimento imperfeito incerto na base de conhecimento
% no caso do cuidado com id de utente desconhecido
involucao(cuidado(D,IdUtente_desconhecido,IdPrest,Desc,C), cuidado, incerto, idutente) :-
    involucao(cuidado(D,IdUtente_desconhecido,IdPrest,Desc,C), positivo),
    remocao((excecao(cuidado(Data,IdUtente,IdPrestador,Descricao,Custo)) :-
                cuidado(Data,IdUtente_desconhecido,IdPrestador,Descricao,Custo))).

% Retira conhecimento imperfeito incerto na base de conhecimento
% no caso do cuidado com id de prestador desconhecido
involucao(cuidado(D,IdUt,IdPrestador_desconhecido,Desc,C), cuidado, incerto, idprestador) :-
    involucao(cuidado(D,IdUt,IdPrestador_desconhecido,Desc,C), positivo),
    remocao((excecao(cuidado(Data,IdUtente,IdPrestador,Descricao,Custo)) :-
                cuidado(Data,IdUtente,IdPrestador_desconhecido,Descricao,Custo))).

% Retira conhecimento imperfeito incerto na base de conhecimento
% no caso do cuidado com descrição desconhecida
involucao(cuidado(D,IdUt,IdPrest,Descricao_desconhecida,C), cuidado, incerto, descricao) :-
    involucao(cuidado(D,IdUt,IdPrest,Descricao_desconhecida,C), positivo),
    remocao((excecao(cuidado(Data,IdUtente,IdPrestador,Descricao,Custo)) :-
                cuidado(Data,IdUtente,IdPrestador,Descricao_desconhecida,Custo))).

% Retira conhecimento imperfeito incerto na base de conhecimento
% no caso do cuidado com custo desconhecido
involucao(cuidado(D,IdUt,IdPrest,Desc,Custo_desconhecido), cuidado, incerto, custo) :-
    involucao(cuidado(D,IdUt,IdPrest,Desc,Custo_desconhecido), positivo),
    remocao((excecao(cuidado(Data,IdUtente,IdPrestador,Descricao,Custo)) :-
                cuidado(Data,IdUtente,IdPrestador,Descricao,Custo_desconhecido))).
```

Figura 47 - Involução de conhecimento imperfeito incerto de cuidado

Exame

```
% Retira conhecimento imperfeito incerto na base de conhecimento
% no caso do cuidado com id de utente desconhecido
involucao(exame(IdUtente_desconhecido,IdPrest,Desc), exame, incerto, idutente) :-
    involucao(exame(IdUtente_desconhecido,IdPrest,Desc), positivo),
    remocao((excecao(exame(IdUtente,IdPrestador,Descricao)) :-
                exame(IdUtente_desconhecido,IdPrestador,Descricao))).

% Retira conhecimento imperfeito incerto na base de conhecimento
% no caso do exame com id de prestador desconhecido
involucao(exame(IdUt,IdPrestador_desconhecido,Desc), exame, incerto, idprestador) :-
    involucao(exame(IdUt,IdPrestador_desconhecido,Desc), positivo),
    remocao((excecao(exame(IdUtente,IdPrestador,Descricao)) :-
                exame(IdUtente,IdPrestador_desconhecido,Descricao))).

% Retira conhecimento imperfeito incerto na base de conhecimento
% no caso do exame com descrição desconhecida
involucao(exame(IdUt,IdPrest,Descricao_desconhecida), exame, incerto, descricao) :-
    involucao(exame(IdUt,IdPrest,Descricao_desconhecida), positivo),
    remocao((excecao(exame(IdUtente,IdPrestador,Descricao)) :-
                exame(IdUtente,IdPrestador,Descricao_desconhecida))).
```

Figura 48 - Involução de conhecimento imperfeito incerto de exame

Na figura seguinte encontra-se um exemplo de involução de conhecimento de prestador incerto, no caso da idade ser desconhecida.

```
| ?- listing.
-utente(A,B,C,D,E) :-
        nao(utente(A,B,C,D,E)),
        nao(excecao(utente(A,B,C,D,E))).
-prestador(A,B,C,D,E,F) :-
        nao(prestador(A,B,C,D,E,F)),
        nao(excecao(prestador(A,B,C,D,E,F))).

excecao(prestador(A,B,_,C,D,E)) :-
        prestador(A, B, idade_desconhecida, C, D, E).

prestador(10, 'Silvia', idade_desconhecida, 'F', 'Cirurgia Geral', 'Hospital de Braga').

yes
| ?- involucao(prestador(10,'Silvia',idade_desconhecida,'F','Cirurgia Geral','Hospital de Braga'),prestador,incerto,idade).
yes
| ?- listing.
-utente(A,B,C,D,E) :-
        nao(utente(A,B,C,D,E)),
        nao(excecao(utente(A,B,C,D,E))).
-prestador(A,B,C,D,E,F) :-
        nao(prestador(A,B,C,D,E,F)),
        nao(excecao(prestador(A,B,C,D,E,F))).

yes
```

Figura 49 - Exemplificação involução de conhecimento imperfeito incerto

3.4.3. Involução de conhecimento imperfeito impreciso

Para o caso da involução de conhecimento imperfeito impreciso os procedimentos criados encontram-se na figura seguinte.

```
% Retira conhecimento imperfeito impreciso na base de conhecimento
% seja de utente, prestador, cuidado ou exame
involucao(T, impreciso) :-
    solucoes(I, -(excecao(T))::I, Lint),
    remocao(excecao(T)),
    teste(Lint).

% Retira conhecimento imperfeito impreciso na base de conhecimento
% no caso de utente com idade contida dentro de um intervalo finito de valores
involucao(utente(Id,N,Idade_imprecisa,G,M), utente, impreciso, idade, LimiteInferior, LimiteSuperior) :-
    remocao((excecao(utente(Id,N,Idade_imprecisa,G,M)) :-
            Idade_imprecisa >= LimiteInferior, Idade_imprecisa =< LimiteSuperior)).

% Retira conhecimento imperfeito impreciso na base de conhecimento
% no caso de prestador com idade contida dentro de um intervalo finito de valores
involucao(prestador(Id,N,Idade_imprecisa,G,E,Inst), prestador, impreciso, idade, LimiteInferior, LimiteSuperior) :-
    remocao((excecao(prestador(Id,N,Idade_imprecisa,G,E,Inst)) :-
            Idade_imprecisa >= LimiteInferior, Idade_imprecisa =< LimiteSuperior)).
```

Figura 50 - Involução de conhecimento imperfeito impreciso

Na figura seguinte encontra-se um exemplo de aplicação do predicado de involução no caso de um utente com idade imprecisa em relação à idade, que se encontrava dentro de um intervalo de valores.

```
| ?- listing.
-utente(A,B,C,D,E) :-
        nao(utente(A,B,C,D,E)),
        nao(excecao(utente(A,B,C,D,E))).
-prestador(A,B,C,D,E,F) :-
        nao(prestador(A,B,C,D,E,F)),
        nao(excecao(prestador(A,B,C,D,E,F))).

excecao(utente(14,'Jupiter',A,'F','Braganca')) :-
        A>=20,
        A=<30.

yes
| ?- involucao(utente(14,'Jupiter',A,'F','Braganca'),utente,impreciso,idade,20,30).
yes
| ?- listing.
-utente(A,B,C,D,E) :-
        nao(utente(A,B,C,D,E)),
        nao(excecao(utente(A,B,C,D,E))).
-prestador(A,B,C,D,E,F) :-
        nao(prestador(A,B,C,D,E,F)),
        nao(excecao(prestador(A,B,C,D,E,F))).

yes
```

Figura 51 – Exemplificação involução de conhecimento imperfeito impreciso

3.4.4. Involução de conhecimento imperfeito interdito

Por fim, no caso da involução de conhecimento imperfeito interdito, os motivos por detrás desta abordagem são os mesmos dos já explicados na Secção 3.3.4.

O procedimento gerado encontra-se na figura seguinte.

```
% Retira conhecimento imperfeito interdito na base de conhecimento
% no caso de utente com nome interdito
 involucao(utente(Id,Nome_impossivel,I,G,M), utente, interdito, nome) :-
    involucao(utente(Id,Nome_impossivel,I,G,M), positivo),
    remocao((excecao(utente(IdUt,Nome,Idade,Genero,Morada)) :-
                utente(IdUt,Nome_impossivel,Idade,Genero,Morada))),
    remocao((nulointerdito(Nome_impossivel))).
```

Figura 52 - Involução de conhecimento imperfeito interdito de utente

3.5. Sistema de Inferência

Tendo em consideração que o nosso sistema de representação de conhecimento e raciocínio contém agora um contradomínio mais alargado de respostas (verdadeiro, falso ou desconhecido) foi essencial construir um sistema de inferência poderoso, capaz de dar implementar os mecanismos de raciocínio adequados.

Para começar, construímos o predicado **si**, que dá **verdadeiro** como resposta se existir uma prova explícita na base de conhecimento em como a Questão é verdadeira, **falso**, se existir na base de conhecimento a negação forte da Questão, e dá **desconhecido**, caso não existam provas sobre Q na base de conhecimento.

```
% Extensao do meta-predicado si: Questao,Resposta
%                                Resposta = {verdadeiro,falso,desconhecido}
% capaz de responder a uma única questão
si(Q,verdadeiro) :- Q.
si(Q,falso) :- -Q.
si(Q,desconhecido) :- nao(Q),
                      nao(-Q).
```

Figura 53 - Sistema de Inferência simples

Para enriquecer o nosso sistema de inferência, criamos um predicado chamado **siLista**, que permite responder a várias questões em simultâneo. Desta forma, basta utilizar este predicado uma única vez e obterá a resposta a cada uma delas.

```
% Extensao do meta-predicado siLista: [Questao],[Resposta] -> {V,F,D}
% capaz de responder a várias questões em simultâneo
siLista([],[]).
siLista([Q|Qs],[R|Rs]) :- si(Q,R),
                          siLista(Qs,Rs).
```

Figura 54 - Sistema de Inferência (lista de questões)

Em seguida, debruçamo-nos também sobre a possibilidade do utilizador querer saber a conjunção e/ou disjunção de uma lista de várias questões. Desta forma, tivemos que pensar inicialmente qual o valor de verdade correspondente a, por exemplo, uma conjunção entre verdadeiro e desconhecido. Depois de todas as decisões tomadas, chegamos às conclusões presentes na *Tabela 1* para o caso da conjunção, e na *Tabela 2* para o caso da disjunção.

Valor de verdade 1	Valor de verdade 2	Conjunção
Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Desconhecido	Desconhecido
Desconhecido	Verdadeiro	Desconhecido
Desconhecido	Desconhecido	Desconhecido
Falso	_	Falso
_	Falso	Falso

Tabela 1 - Valores de Verdade Conjunção

Valor de verdade 1	Valor de verdade 2	Disjunção
Verdadeiro	_	Verdadeiro
_	Verdadeiro	Verdadeiro
Falso	Falso	Falso
Falso	Desconhecido	Desconhecido
Desconhecido	Falso	Desconhecido
Desconhecido	Desconhecido	Desconhecido

Tabela 2 - Valores de Verdade Disjunção

Os predicados que traduzem essas decisões encontram-se na *Figura 55* e *Figura 56*, respetivamente.

```
% Extensao do predicado conjuncao: X,Y -> {V,F,D}
conjuncao(verdadeiro,verdadeiro,verdadeiro).
conjuncao(verdadeiro,desconhecido,desconhecido).
conjuncao(desconhecido,verdadeiro,desconhecido).
conjuncao(desconhecido,desconhecido,desconhecido).
conjuncao(falso,_,falso).
conjuncao(_,falso,falso).
```

Figura 55 - Predicado conjunção

```
% Extensao do predicado disjuncao: X,Y -> {V,F,D}
disjuncao(verdadeiro,_,verdadeiro).
disjuncao(_,verdadeiro,verdadeiro).
disjuncao(falso,falso,falso).
disjuncao(falso,desconhecido,desconhecido).
disjuncao(desconhecido,falso,desconhecido).
disjuncao(desconhecido,desconhecido,desconhecido).
```

Figura 56 - Predicado disjunção

Posteriormente, utilizando os predicados anteriormente apresentados, criamos dois procedimentos: o primeiro, na *Figura 57*, que permite fazer a **conjunção de uma lista** de questões, obtendo como resposta um único resultado final, e um segundo procedimento, na *Figura 58*, que faz a **disjunção**.

```
% Extensao do predicado queryConjuncao: [Questao],Resposta -> {V,F,D}
% capaz de fazer a conjunção de uma lista de questões
% produzindo um resultado final
queryConjuncao([],R).
queryConjuncao([Q],R) :- si(Q,R).
queryConjuncao([Q|Qs], R) :- si(Q, R1), queryConjuncao(Qs,R2), conjuncao(R1,R2,R).
```

Figura 57 - Conjunção (lista de questões)

```
% Extensao do predicado queryDisjuncao: [Questao],Resposta -> {V,F,D}
% capaz de fazer a disjunção de uma lista de questões
% produzindo um resultado final
queryDisjuncao([],R).
queryDisjuncao([Q],R) :- si(Q,R).
queryDisjuncao([Q|Qs], R) :- si(Q, R1), queryDisjuncao(Qs,R2), disjuncao(R1,R2,R).
```

Figura 58 - Disjunção (lista de questões)

Se o objetivo for saber a disjunção ou conjunção entre somente duas questões pode usar-se o predicado criado para o efeito, presenta na figura seguinte.

```
% Extensao do predicado query: (Operador, Questao1, Questao2),Resposta -> {V,F,D}
% capaz de fazer a conjunção ou a disjunção de duas questões
% produzindo um resultado final
query(('OR',Q1,Q2),R) :- query(Q1,R1), query(Q2,R2), disjuncao(R1,R2,R).
query(('AND',Q1,Q2),R) :- query(Q1,R1), query(Q2,R2), conjuncao(R1,R2,R).
query(Q,R) :- si(Q,R).
```

Figura 59 - Conjunção/Disjunção (2 questões)

Por fim, decidimos alargar o domínio do procedimento acima para uma lista de várias questões, ao invés de apenas 2. O objetivo deste procedimento é dar resposta a uma lista de questões, intercaladas por conjunções e/ou disjunções, obtendo um único resultado final.

```
% Extensao do meta-predicado queries: [Questao], Resposta -> {V,F,D}
% capaz de fazer a conjunção e/ou disjunção de uma lista de questões
% produzindo um resultado final
queries([],verdadeiro).
queries([Q],R) :- si(Q,R).
queries([Q1,'AND'|Qs],R) :- si(Q1,R1),
                            queries(Qs,R2),
                            conjuncao(R1,R2,R).
queries([Q1,'OR'|Qs],R) :- si(Q1,R1),
                           queries(Qs,R2),
                           disjuncao(R1,R2,R).
```

Figura 60 - Conjunção e/ou Disjunção (lista de questões)

4. Conclusões e Sugestões

No final deste exercício, a equipa conclui que foi desenvolvido com sucesso um sistema de caraterização de um universo na área da prestação de cuidados de saúde pela realização de serviços de atos médicos, capaz de demonstrar as funcionalidades subjacentes à programação em lógica estendida e à representação de conhecimento imperfeito, recorrendo à temática dos valores nulos. Este sistema inclui diversos tipos de conhecimento como utentes, prestadores, cuidados e exames.

Foram ainda implementadas todas as funcionalidades solicitadas que permitem uma representação de todo o tipo de conhecimento estudado, perfeito (positivo e negativo) e imperfeito (incerto, impreciso e interdito), com vários exemplos e demonstrações de cada um deles. Foram pensados ao pormenor os invariantes necessários para uma correta manipulação da base de conhecimento, tanto para o caso da adição como para o caso da remoção de conhecimento.

A problemática da evolução e da involução de conhecimento foi analisada ao detalhe para que todos os casos fossem abrangidos como forma de conseguirmos construir um sistema que insere e remove qualquer tipo de conhecimento. Os procedimentos criados foram sendo acompanhados com pequenos exemplos de aplicação para que a sua eficiência fosse de certa forma comprovada. Foi ainda construído um sistema de inferência refinado, possibilitando a resposta a conjunções e disjunções de várias questões, tendo sempre em atenção que estamos perante uma extensão à programação em lógica.

Em futuras evoluções deste trabalho poderá, por exemplo, ser estendida a fonte de conhecimento, através da inclusão de, por exemplo, auxiliares de saúde, equipamentos médicos, entre muitos outros. Seria ainda possível que futuras versões permitissem pesquisas mais refinadas na base de conhecimento, acrescentando novos predicados.

A boa preparação da equipa e o estudo prévio, quer do motor de inferência, quer do caso de estudo, quer da problemática da programação em lógica estendida possibilitaram que todo o processo decorresse dentro da normalidade, sem que tenham surgido problemas de dimensão considerável.

5. Referências

5.1. Referências Bibliográficas

[Analide, 1996] ANALIDE, Cesar, Neves, José,

"Representação de Informação Incompleta",

Texto Pedagógico, Grupo de Inteligência Artificial, Centro de Ciências e Tecnologias da Computação, Portugal, 1996.

[Analide, 2011] ANALIDE, Cesar, NOVAIS, Paulo, Neves, José,

"Sugestões para a Redacção de Relatórios Técnicos",

Relatório Técnico, Departamento de Informática, Universidade do Minho, Portugal, 2011.

[Bratko, 2000] BRATKO, Ivan,

"PROLOG: Programming for Artificial Intelligence",

United States, Pearson Education (us), 2000.

5.2. Referências Eletrónicas

CATEGORIA: Exames médicos. Disponível em: https://pt.wikipedia.org/wiki/Categoria:Exames_m%C3%A9dicos. Acesso em: 11 mar. 2019.

LISTA de Especialidades Médicas. Disponível em: https://pt.wikipedia.org/wiki/Lista_de_especialidades_m%C3%A9dicas. Acesso em: 11 mar. 2019.

ORDEM dos Enfermeiros: Especialidades. Disponível em: https://www.ordemenfermeiros.pt/faqs/especialidades/. Acesso em: 11 mar. 2019.

PROLOG. Disponível em: https://pt.wikibooks.org/wiki/Prolog. Acesso em: 5 abr. 2019.

SICStus Prolog. Disponível em: https://sicstus.sics.se/sicstus/docs/latest4/html/sicstus.html/. Acesso em: 5 abr. 2019.

6. Anexos

6.1. Predicados Auxiliares

Para o desenvolvimento dos procedimentos requeridos pelo nosso sistema de representação de conhecimento e raciocínio, recorremos a vários predicados auxiliares que foram extremamente úteis no decorrer de todo o processo.

Em seguida, apresentamos esses predicados desenvolvidos, que fomos referindo ao longo do desenvolvimento do presente relatório.

O meta-predicado **não**, que devolve o valor de verdade contrário ao termo **Q** passado como parâmetro através da negação fraca, isto é, caso exista uma prova afirmativa ou negativa explícita de Q na base de conhecimento dá *no* como resposta e, caso contrário, ou seja, na ausência de prova, dá *yes*.

```
% Extensao do meta-predicado não: Q -> {V,F}
% Devolve o valor de verdade contrário a Q (negação fraca)
nao(Q) :- Q, !, fail.
nao(Q).
```

Figura 61 - Meta-predicado 'nao'

O meta-predicado **soluções** que coloca em R a lista com as soluções do tipo X para todo o XS que encontrar.

```
% Extensao do predicado soluções: X,XS,R -> {V,F}
% Encontra todas as soluções
solucoes(X, XS, R) :- XS, assert(tmp(X)), fail.
solucoes(X, XS, R) :- solucoesAux([], R).

solucoesAux(L, R) :- retract(tmp(X)), !, solucoesAux([X|L], R).
solucoesAux(R, R).
```

Figura 62 - Meta-predicado 'solucoes'

O predicado **comprimento**, que coloca em R o comprimento da lista passada como argumento.

```
% Extensao do predicado comprimento: X,R -> {V,F}
% Devolve o comprimento de uma lista
comprimento([],0).
comprimento([_|XS],R) :- comprimento( XS,S ), R is 1 + S.
```

Figura 63 - Predicado 'comprimento'

O meta-predicado **inserção** que coloca Q na base de conhecimento no caso de sucesso, retornando *yes*, e retira Q no caso de haver retrocesso, retornando *no*. O meta-predicado **remoção**, que faz o oposto do meta-predicado **inserção**, ou seja, elimina Q da base de conhecimento no caso de sucesso e adiciona Q no caso de haver retrocesso, retornando *no*.

```
% Inserção de conhecimento
insercao(Q) :- assert(Q).
insercao(Q) :- retract(Q), !, fail.
```

Figura 64 – Meta-predicado 'insercao'

```
% Remoção de conhecimento
remocao(Q) :- retract(Q).
remocao(Q) :- assert(Q), !, fail.
```

Figura 65 – Meta-predicado 'remocao'

O meta-predicado **teste**, que testa se todos os predicados passados como parâmetro são verdadeiros.

```
% Testa se todos os predicados são verdadeiros
teste([]).
teste([I|L]) :- I, teste(L).
```

Figura 66 – Meta-predicado 'teste'

E, por fim, o predicado **idade válida** (idade >= 0), **custo válido** (custo >= 0) e **género válido** (masculino, ‘M’, ou feminino, ‘F’).

```
% Idade válida (>= 0)
idadeValida(I) :- I >= 0.

% Custo válido (>= 0)
custoValido(C) :- C >= 0.

% Género é masculino ('M') ou feminino ('F')
generoValido('M').
generoValido('F').
```

Figura 67 - Predicado 'idadeValida', 'custoValido' e 'generoValido'