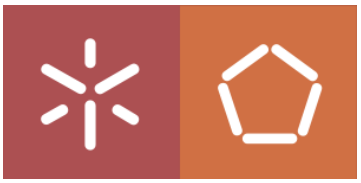

Universidade do Minho

Departamento de Informática

Sistemas de Representação de Conhecimento e Raciocínio

Programação em Lógica Estendida e Conhecimento Imperfeito

André Ferreira (a64296) Maria João Moreira (a89540)
Rúben Rodrigues (a80960) Rui Fernandes (a89138)
Rui Morais (a76650)

Maio 2021

Resumo

O presente relatório tem como objetivo descrever a solução criada pelo grupo para a problemática proposta pelos docentes da Unidade Curricular de *Sistemas de Representação de Conhecimento e Raciocínio*, ao longo do segundo semestre do terceiro ano do Mestrado Integrado em Engenharia Informática da Universidade do Minho.

O presente trabalho tem como principal objetivo aprimorar a utilização da extensão à programação em lógica, bem como a representação de conhecimento imperfeito. Assim, pretende-se desenvolver um sistema de representação de conhecimento e raciocínio com capacidade para caracterizar um universo de discurso na área da vacinação global da população portuguesa no contexto da pandemia COVID-19 que estamos a viver.

Por fim, é apresentada uma reflexão crítica sobre o trabalho desenvolvido, bem como os resultados obtidos.

Conteúdo

Listings

Lista de Tabelas

1 Introdução

O presente relatório documenta a segunda fase do trabalho prático da Unidade Curricular de *Sistemas de Representação de Conhecimento e Raciocínio*. Este projeto tem como finalidade o desenvolvimento de um sistema de representação de conhecimento e raciocínio com capacidade para caracterizar um universo de discurso na área da vacinação global da população portuguesa no contexto da pandemia COVID-19 que estamos a viver.

Inicialmente, começou-se por representar conhecimento positivo e negativo, devido à possibilidade de se ter respostas verdadeiras, falsas e desconhecidas, sendo que o conhecimento positivo já tinha sido abordado na primeira fase do trabalho prático. De seguida, implementou-se a noção de conhecimento imperfeito, devido à presença da resposta como o desconhecido, existindo três tipos diferentes deste tipo de conhecimento incerto, impreciso e interdito.

Para a representação de conhecimento imperfeito foi necessário criar predicados correspondentes aos três tipos de conhecimento imperfeito – incerto, impreciso e interdito.

Neste documento, serão abordados os novos predicados criados, assim como as alterações feitas nos que transitaram da primeira fase do trabalho prático, terminando com uma análise crítica dos resultados do trabalho efetuado.

2 Preliminares

Na primeira fase do trabalho prático realizado anteriormente, foi desenvolvido um sistema de representação de conhecimento e raciocínio capaz de representar conhecimento perfeito, assemelhando-se a um sistema de bases de dados, na medida em que assumia-se que a informação representada seria única e válida e que as entidades representadas seriam as únicas existentes no mundo exterior.

Após um estudo mais aprofundado em relação à programação em lógica e à respetiva representação de informação, fomos confrontados com algumas limitações. Para debater essas limitações, importa ter em atenção que este paradigma se baseia nos seguintes pressupostos:

— **Pressuposto do Mundo Fechado**, que afirma que toda a informação que não é mencionada na base de conhecimento é considerada como falsa;

— **Pressuposto dos Nomes Únicos**, que garante que duas constantes diferentes (que definem valores atómicos ou objetos) designam duas entidades diferentes do universo de discurso;

— **Pressuposto do Domínio Fechado**, que indica que não existem mais objetos no universo de discurso para além daqueles designados por constantes na base de conhecimento.

Estes pressupostos foram salvaguardados na primeira fase do trabalho prático, 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, é necessário ter em atenção se está algum carro a aproximar-se. Para tal, recorrendo à programação em lógica, caso na base de conhecimento não exista informação relativa a carros a aproximarem-se, o motor de inferência indicará que não se aproxima nenhum automóvel e que, consequentemente, é seguro atravessar. Porém, isto não prova que não se aproxima nenhum carro, apenas que não há nenhuma prova de que se aproximam carros.

Desta forma, surge a necessidade de 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 sendo, o Pressuposto dos Nomes Únicos continuará a ser válido uma vez que não interfere negativamente na representação do conhecimento. No entanto, o Pressuposto do Mundo Fechado e o Pressuposto do Domínio Fechado serão substituídos por:

— **Pressuposto do Mundo Aberto**, que indica que podem existir outros factos ou conclusões verdadeiras para além daqueles representados na base de conhecimento;

— **Pressuposto do Domínio Aberto**, que garante que podem existir mais objetos do universo de discurso para além daqueles designados pelas constantes da base de conhecimento.

Para além dos novos pressupostos mencionados, a Programação em Lógica Estendida permite também representar 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.

Por outro lado, 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 na prova*, que é representada pelo termo `nao`, 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 indica que existe uma prova na base de conhecimento de que a questão é falsa.

2.1 Representação de Conhecimento Imperfeito

O conhecimento imperfeito consiste na ausência de informação em relação a uma questão colocada. Em caso de informação incompleta, isto é, desconhecida, existem três casos distintos:

— **Incerto** – representa valores nulos que não pertencem a nenhum conjunto determinado de valores, sendo portanto completamente desconhecidos;

— **Impreciso** – representa valores nulos que pertencem a um conjunto determinado de valores (conhecimento desconhecido mas dentro de um conjunto finito de hipóteses), *i.e* existe uma noção do que é falso mas não se conhece qual a verdadeira resposta;

— **Interdito** – representa valores nulos desconhecidos e que não serão permitidos conhecer.

Em suma, considerando o trabalho desenvovlido na fase anterior, para a elaboração do presente trabalho prático resta agora desenvolver um sistema de inferência que permita implementar os mecanismos de raciocínio mencionados, inerentes a estes sistemas, assim como povoar a base de conhecimento com os diferentes tipo de conhecimento supramencionados.

3 Descrição do Trabalho e Análise de Resultados

Nesta secção irá ser discutida e devidamente fundamentada toda a resolução desta segunda fase do trabalho prático. O foco estará em respeitar as necessidades das várias funcionalidades enumeradas no enunciado do trabalho prático, no sentido de se conseguir produzir uma resolução atenta, mas concisa e aceitável.

3.1 Base de Conhecimento

Tendo em conta que o universo que se pretende representar continua a focar-se na vacinação global da população portuguesa no contexto da pandemia COVID-19, a base de conhecimento que foi desenvolvida na primeira fase do trabalho prático acaba por se manter praticamente igual, existindo apenas mínimas alterações.

Com o intuito de representar informação incompleta, torna-se necessário alterar o domínio de soluções passando a incluir o valor de verdade *Desconhecido*. Este representa um valor que não pode ser definido de imediato e é com ele que surge o conceito de programação em lógica estendida.

Com esta alteração, a definição da base de conhecimento passa a ser a seguinte:

— `utente: #Idutente, No Seguranca_Social, Nome, Data_Nasc, Email, Telefone, Morada, Profissao, [Doencas_Cronicas], #CentroSaude -> {V, F, D}` – Um utente tem associado a si um ID, número de Segurança Social, nome, data de nascimento, *email*, telefone, morada, profissão, eventuais doenças crónicas, assim como o ID do centro de saúde que frequenta;

— `staff: #Idstaff, #Idcentro, Nome, Email -> {V, F, D}` – Um elemento do *staff* do centro de saúde é caracterizado pelo seu ID, pelo ID do centro de saúde em que exerce funções, e pelo seu nome e *email*;

— `centro_saude: #Idcentro, Nome, Morada, Telefone, Email -> {V, F, D}` – Um centro de saúde é identificado pelo seu ID, nome, morada, telefone e *email*;

— `vacinacao_Covid: #Idstaff, #Idutente, Data, Vacina, Toma -> {V, F, D}` – O registo de um ato de vacinação é caracterizado pelo ID do elemento do *staff* responsável pela administração da vacina, o ID do utente, a data em que e qual a vacina que a vacina foi administrada, assim como a respetiva toma.

— `medico: #Idmedico, #Idcentro, Nome, Email, Especialidade -> {V, F, D}` – Um médico possui também um ID, nome, *email* e especialidade. Adicionalmente, existe uma associação entre o médico e o centro de saúde em que este exerce funções;

— `consulta: #Idmedico, #Idutente, #Idcentro, Data -> {V, F, D}` – Uma consulta é caracterizada pelo ID do médico, pelo ID do utente, o centro de saúde é realizada e a respetiva data;

— `tratamento: #IdStaff, #Idutente, #Idcentro, Data, Descricao -> {V, F, D}` – Um tratamento é caracterizado por uma uma breve descrição, o centro de saúde onde é realizado, assim como o elemento do *staff* que responsável, o utente, e a data em que este foi prestado.

Como forma de construir uma base de conhecimento ampla e generalizada, optamos por incluir informação negativa explicitamente, bem como explicitar diretamente o Pressuposto do Mundo Fechado para alguns predicados. Desta forma, mantivemos este pressuposto para os predicados `utente`, `staff` e `medico`, assumindo então que um utente, um elemento do *staff* do centro de saúde um médico não definidos na base de conhecimento nem que contenham alguma exceção associada são considerados conhecimento falso. Esta decisão traz bastante vantagens, como por exemplo, no caso da representação de conhecimento imperfeito impreciso, visto que se não se soube exatamente o valor da idade do utente mas se souber o intervalo de valores em que esta se situa, é possível afirmar com certezas que o utente com a idade não contida entre esses valores é falso, apesar dessa informação não se encontrar explicitamente na base de conhecimento.

```
% Pressuposto do mundo fechado para o predicado utente
-utente(ID, NUM, NOME, DN, EMAIL, TLF, M, P, DC, CS) :-
    nao(utente(ID, NUM, NOME, DN, EMAIL, TLF, M, P, DC, CS)),
    nao(excecao(utente(ID, NUM, NOME, DN, EMAIL, TLF, M, P, DC, CS))).

% Pressuposto do mundo fechado para o predicado centro_saude
-centro_saude(ID, NOME, M, TLF, EMAIL) :-
    nao(centro_saude(ID, NOME, M, TLF, EMAIL)),
    nao(excecao(centro_saude(ID, NOME, M, TLF, EMAIL))).

% Pressuposto do mundo fechado para o predicado staff
-staff(IDS, IDCENTRO, NOME, EMAIL) :-
    nao(staff(IDS, IDCENTRO, NOME, EMAIL)),
    nao(excecao(staff(IDS, IDCENTRO, NOME, EMAIL))).
```

Listing 1: Pressuposto do Mundo Fechado para os predicados `utente`, `centro_saude` e `staff`

3.2 Representação de Conhecimento Perfeito

A representação de conhecimento positivo já se encontra realizada na primeira fase do trabalho prático. No entanto, para a realização desta segunda fase foi necessário acrescentar a representação do conhecimento negativo.

3.2.1 Conhecimento Perfeito Positivo

Tendo o panorama de conhecimento bem definido, começou-se por povoar o sistema acrescentando factos perfeitos positivos relativos a todos os predicados mencionados anteriormente. O povoamento da base de conhecimento com conhecimento perfeito positivo encontra-se em anexo, na secção 5.2.1

3.2.2 Conhecimento Perfeito Negativo

O conhecimento perfeito negativo foi representado tendo em conta os diferentes tipos.

Negação por Falha na Prova – Quando não existe nenhuma prova aquando da negação do predicado. Esta negação é representada pelo meta-predicado `nao`, que se encontra definido em anexo (secção 5.1).

Negação Forte – Quando se afirma que um determinado predicado é falso. Esta negação é representada pelo teorema `-`.

```
-utente(13, 41582, 'Duarte Pedro', date(1987, 6, 2), 'duartepedro@gmail.
    com', 913654782, 'Coimbra', 'Jornalista', [], 3).
-utente(14, 45217, 'Tiago Loureiro', date(1993, 5, 12), '
    tiagoloureiro@gmail.com', 912396755, 'Lisboa', 'Enfermeiro', [], 4).

-centro_saude(5, 'Centro Hospitalar Povoa de Varzim', 'Povoa de Varzim',
    252690600, 'centrohospitalarpovoadevarzim@gmail.com').

-staff(11, 3, 'Igor Rodrigues', 'igorrodrigues@gmail.com').
-staff(12, 2, 'Ana Mendes', 'anamendes@gmail.com').

-vacinacao_Covid(3, 5, date(2021, 3, 21), 'Pfizer', 1).
-vacinacao_Covid(3, 5, date(2021, 4, 2), 'Pfizer', 2).

-medico(7, 4, 'Cesar Martins', 'cesarmartins@gmail.com', 'Oftalmologia').

-consulta(3, 1, 4, date(2021,2,15)).

-tratamento(7, 5, 1, date(2021,2,17), 'Radiografia').
```

Listing 2: Povoamento da base de conhecimento com conhecimento perfeito negativo

3.3 Representação de Conhecimento Imperfeito

Na problemática da representação de informação incompleta, os valores nulos surgem com o objetivo de distinguir situações em que as respostas às questões são conhecidas e as situações em que as respostas às questões são desconhecidas. É neste sentido que surgem os três tipos de conhecimento imperfeito – incerto, impreciso e interdito – que são essenciais para todo um alargar de possibilidades de respostas às questões com base nas fontes de conhecimento existentes.

3.3.1 Conhecimento Imperfeito Incerto

Este tipo de conhecimento corresponde ao conhecimento para o qual não existe nenhuma prova que comprove que este seja verdadeiro ou falso. Trata-se, portanto, de um valor nulo do tipo desconhecido e não necessariamente de um conjunto de valores.

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

Utente

No nosso caso de estudo, isto pode ser caracterizado pela situação em que se sabe que existe um utente com ID igual a 15, chamado Mateus, mas que não se sabe qual a sua idade, uma vez que a sua data de nascimento é desconhecida. Neste caso, isto é definido como conhecimento incerto visto que não se sabe qual é a sua data de nascimento, nem se tem uma ideia de qual possa ser. Para representar este tipo conhecimento, é necessário criar o utente mencionado com `data_desconhecida` no campo relativo à data de nascimento e criar uma exceção que indique que quando se questionar a base de conhecimento relativamente ao utente com `data_desconhecida`, o resultado deverá ser *desconhecido*.

```
% Nao se conhece a idade do utente #15 uma vez que a sua data de
    nascimento e desconhecida
utente(15, 56257, 'Mateus Silva', data_desconhecida, 'mateussilva@gmail.
    com', 915698401,
        'Coimbra', 'Bombeiro', [], 4).
excecao(utente(ID, NUM, NOME, DN, EMAIL, TLF, M, P, DC, CS)) :-
        utente(ID, NUM, NOME, data_desconhecida, EMAIL, TLF, M, P, DC, CS
    ).
```

Listing 3: Conhecimento Imperfeito Incerto relativo ao utente

Centro de Saúde

Relativamente ao centro de saúde, o conhecimento imperfeito incerto surge quando, por exemplo, não se conhece o número de telefone. De forma análoga, é necessário criar o centro de saúde mencionado com `telefone_desconhecido` no campo relativo ao telefone e criar uma exceção que indique que quando se questionar a base de conhecimento relativamente ao centro de saúde com `telefone_desconhecido`, o resultado deverá ser *desconhecido*.

```
% Nao se conhece o telefone do centro de saude #6
centro_saude(6, 'Hospital da Luz', telefone_desconhecido, 'Guimaraes',
             'hospitaldaluz@gmail.com').
excecao(centro_saude(ID, NOME, M, TLF, EMAIL)) :-
    centro_saude(ID, NOME, M, telefone_desconhecido, EMAIL).
```

Listing 4: Conhecimento Imperfeito Incerto relativo ao centro de saúde

Staff

No que diz respeito ao *staff* de um centro de saúde, o conhecimento imperfeito incerto surge quando, por exemplo, não se conhece o ID do centro de saúde em que um que funcionário exerce funções. Desta forma, é necessário criar o funcionário mencionado com `idc_desconhecido` no campo relativo ao ID do centro em que este exerce funções e criar uma exceção que indique que quando se questionar a base de conhecimento relativamente ao funcionário com `idc_desconhecido`, o resultado deverá ser *desconhecido*.

```
% Nao se sabe em que centro de saude o Antonio exerce funcoes
staff(13, idc_desconhecido, 'Antonio Gomes', 'antoniogomes@gmail.com').
excecao(staff(IDS, IDCENTRO, NOME, EMAIL)) :-
    staff(IDS, idc_desconhecido, NOME, EMAIL).
```

Listing 5: Conhecimento Imperfeito Incerto relativo ao *staff* do centro de saúde

Vacinação

O conhecimento imperfeito incerto surge associado aos registos de vacinação quando, por exemplo, não se conhece com que vacina um determinado utente foi vacinado. Desta forma, é necessário criar o registo de vacinação mencionado com `vacina_desconhecida` no campo relativo à vacina e criar uma exceção que indique que quando se questionar a base de conhecimento relativamente a um registo de vacinação com `vacina_desconhecida`, o resultado deverá ser *desconhecido*.

```
% Nao se sabe com que vacina o utente #7 foi vacinado
vacinacao_Covid(1, 7, date(2020,3,21), vacina_desconhecida, 1).
excecao(vacinacao_Covid(STAFF, UTENTE, DATA, VACINA, TOMA)) :-
    vacinacao_Covid(STAFF, UTENTE, DATA, vacina_desconhecida, TOMA).
```

Listing 6: Conhecimento Imperfeito Incerto relativo aos registos de vacinação

Médico

O conhecimento imperfeito incerto surge associado ao médico quando, por exemplo, não se conhece a sua especialidade. Assim, à semelhança do que foi referido anteriormente, é necessário criar o médico mencionado com `especialidade_desconhecida` no campo relativo à especialidade médica e criar uma exceção que indique que quando se questionar a base de conhecimento relativamente a um médico com `especialidade_desconhecida`, o resultado deverá ser *desconhecido*.

```
% Nao se conhece a especialidade do medico #8
medico(8, 2, 'Henrique Ferreira', 'henriqueferreira@gmail',
       especialidade_desconhecida).
excecao(medico(ID, IDCENTRO, NOME, EMAIL, ESP)) :-
    medico(ID, IDCENTRO, NOME, EMAIL, especialidade_desconhecida).
```

Listing 7: Conhecimento Imperfeito Incerto relativo ao médico

Consulta

O conhecimento imperfeito incerto surge associado a uma consulta quando, por exemplo, não se conhece qual o ID do utente que foi à consulta. Desta forma, é necessário criar o registo da consulta com `utente_desconhecido` no campo relativo ao utente e criar uma exceção que indique que quando se questionar a base de conhecimento relativamente a uma consulta com `utente_desconhecido`, o resultado deverá ser *desconhecido*.

```
% Nao se sabe qual o utente que foi a consulta
consulta(4, utente_desconhecido, 2, date(2021,2,15)).
excecao(consulta(IDM, IDU, IDC, DATA)) :-
    consulta(IDM, utente_desconhecido, IDC, DATA).
```

Listing 8: Conhecimento Imperfeito Incerto relativo à consulta

Tratamento

Seguindo um raciocínio idêntico, o conhecimento imperfeito incerto surge associado a um tratamento quando, por exemplo, não se conhece qual o ID do funcionário responsável pelo mesmo. Desta forma, é necessário criar o registo do tratamento com `ids_desconhecido` no campo relativo ao funcionário e criar uma exceção que indique que quando se questionar a base de conhecimento relativamente a um tratamento com `ids_desconhecido`, o resultado deverá ser *desconhecido*.

```
% Nao se sabe qual o elemento do staff do centro de saude responsavel
    pelo tratamentos
tratamento(ids_desconhecido, 1, 4, date(2020,11,4), 'Biopsia').
excecao(tratamento(IDS, IDU, IDC, DATA, DSCR)) :-
    tratamento(ids_desconhecido, IDU, IDC, DATA, DSCR).
```

Listing 9: Conhecimento Imperfeito Incerto relativo ao tratamento

3.3.2 Conhecimento Imperfeito Impreciso

Tal como o conhecimento desconhecido, este tipo corresponde ao conhecimento para o qual não há provas de que este seja verdadeiro ou falso. No entanto, este insere-se num conjunto de valores conhecidos.

Assim, qualquer que seja a resposta ao predicado que utilize um valor fora desse conjunto, o resultado será *desconhecido*, uma vez que apenas se sabe que o valor se encontra contido dentro desse conjunto.

Utente

Neste caso de estudo temos, por exemplo, a situação de sabermos que existe um utente com ID igual a 9, chamado Pedro Reis, mas que não sabemos se mora em Braga ou no Porto. 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 a este utente a resposta seja *desconhecido* se a morada for Braga ou Porto, e a resposta ser *falso*, caso a base de conhecimento seja questionada com a especialidade diferente destas.

```
% Nao se sabe se a Teresa mora em Braga ou no Porto
excecao(utente(16, 59240, 'Teresa Soares', date(1998, 1, 23), '
    teresasoares@gmail.com', 913654700, 'Braga', 'Estudante',
                ['Hipertensao'], 1)).
excecao(utente(16, 59240, 'Teresa Soares', date(1998, 1, 23), '
    teresasoares@gmail.com', 913654700, 'Porto', 'Estudante',
                ['Hipertensao'], 1)).
```

Listing 10: Conhecimento Imperfeito Impreciso relativo ao utente

Centro de Saúde

O conhecimento imperfeito impreciso surge associado ao centro de saúde quando, por exemplo, se sabe que existe um centro de saúde mas não se conhece a sua morada. A título de exemplo, considere-se o Hospital De Santiago, que não se sabe se se situa em Setúbal ou Lisboa. 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 a este centro saúde a resposta seja *desconhecido* se a morada for Setúbal ou Lisboa, e a resposta ser *falso*, caso a base de conhecimento seja questionada com uma morada diferente destas.

```
% Nao se sabe se o centro de saude #7 esta localizado em Setubal ou em
    Lisboa
excecao(centro_saude(7, 'Hospital De Santiago', 'Setubal', 265509200,
        'hospitaldesantiago@gmail.com')).
excecao(centro_saude(7, 'Hospital De Santiago', 'Lisboa', 265509200,
        'hospitaldesantiago@gmail.com')).
```

Listing 11: Conhecimento Imperfeito Impreciso relativo ao centro de saúde

Staff

Neste caso de estudo temos, por exemplo, a situação de sabermos que existe um funcionário com ID igual a 14, chamado Jorge Carvalho, mas que não sabemos se exerce funções no centro de saúde #2 ou #3. 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 a este funcionário a resposta seja *desconhecido* se o ID do centro de saúde for 2 ou 3, e a resposta ser *falso*, caso a base de conhecimento seja questionada com um ID diferente.

```
% Nao se sabe se o Jorge exerce funcoes no centro de saude #2 ou no #3
excecao(staff(14, 2, 'Jorge Carvalho', 'jorgecarvalho@gmail.com')).
excecao(staff(14, 3, 'Jorge Carvalho', 'jorgecarvalho@gmail.com')).
```

Listing 12: Conhecimento Imperfeito Impreciso relativo ao *staff* do centro de saúde

Vacinação

Neste caso de estudo temos, por exemplo, a situação de sabermos que um utente foi vacinado, mas que não sabemos com que vacina – apenas se sabe que ou foi com a vacina da *Pfizer* ou com a da *AstraZeneca*. 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 a este registo de vacinação seja *desconhecido* se a vacina utilizada for da *Pfizer* ou da *AstraZeneca*, e a resposta ser *falso*, caso a base de conhecimento seja questionada com uma vacina diferente.

```
excecao(vacinacao_Covid(3, 11, date(2021, 3, 21), 'Pfizer', 1)).
excecao(vacinacao_Covid(3, 11, date(2021, 3, 21), 'Astrazeneca', 1)).
```

Listing 13: Conhecimento Imperfeito Impreciso relativo aos registos de vacinação

Médico

No presente caso de estudo, temos, por exemplo, a situação de sabermos que existe um médico com ID igual a 9, chamado Pedro Reis, mas que não sabemos se é especialista em Dermatologia ou em Cardiologia. 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 a este médico a resposta seja *desconhecido* se a especialidade for

Dermatologia ou Cardiologia, e a resposta ser *falso*, caso a base de conhecimento seja questionada com a especialidade diferente destas.

```
% Nao se sabe se o medico #9 e dermatologista ou cardiologista
excecao(medico(9, 1, 'Pedro Reis', 'pedroreis@gmail.com',
        'Dermatologia')).
excecao(medico(9, 1, 'Pedro Reis', 'pedroreis@gzmail.com',
        'Cardiologia')).
```

Listing 14: Conhecimento Imperfeito Impreciso relativo ao médico

Consulta

No caso de estudo apresentado, temos, também, a situação de sabermos que o médico #8 deu uma consulta, mas não sabemos se o foi o utente #5 ou #8 que foi à consulta. 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 a esta consulta, a resposta seja *desconhecido* se a o ID do utente for 5 ou 8, e a resposta ser *falso* caso a base de conhecimento seja questionada com um ID diferente.

```
% Nao se sabe se foi o utente #5 ou #8 que foi a consultas
excecao(consulta(4, 5, 2, date(2021,3,1))).
excecao(consulta(4, 8, 2, date(2021,3,1))).
```

Listing 15: Conhecimento Imperfeito Impreciso relativo à consulta

Tratamento

No caso de estudo apresentado, temos, por fim, a situação de sabermos que o o utente #8 realizou uma prova de esforço, mas não sabemos se o funcionário responsável foi o funcinário #3 ou #5. 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 a este tratamento, a resposta seja *desconhecido* se a o ID do funcionário for 3 ou 5, e a resposta ser *falso* caso a base de conhecimento seja questionada com um ID diferente.

```
% Nao se sabe se foi o elemento do staff do centro de saude #3 ou #5
    responsavel pelo tratamento
excecao(tratamento(3, 8, 2, date(2020,12,15), 'Prova de Esforco')).
excecao(tratamento(5, 8, 2, date(2020,12,15), 'Prova de Esforco')).
```

Listing 16: Conhecimento Imperfeito Impreciso relativo ao tratamento

Por fim, importa 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`, `staff` e `medico`. Apenas nestes 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.

3.3.3 Conhecimento Imperfeito Interdito

Este último tipo de conhecimento, para além de caracterizar um tipo de dados desconhecido, caracteriza também, um tipo de dados que não se admite que surja na base de conhecimento. Nesta situação, o valor nulo para além de identificar um valor desconhecido, representará um valor que não é permitido especificar ou conhecer e qualquer tentativa para a concretizar será rejeitada como sendo provocadora de inconsistência na informação presente na base de conhecimento.

Desta forma, este tipo de conhecimento, para além de identificar os valores desconhecidos, não permite que haja evolução desse conhecimento, isto é, não é permitido especificar ou conhecer o seu valor.

Utente

Para mostrar este tipo de conhecimento criou-se um utente chamado Diogo, identificado pelo ID 17, do qual não se conhece o número de Segurança Social – representado por `sc_desconhecido`. Como neste tipo de conhecimento é impossível alguma vez descobrir o número de Segurança Social utente, torna-se necessário criar um invariante que não permita a evolução deste conhecimento (abordado na secção 3.6) e para isso, criou-se um predicado nulo para identificar o `sc_desconhecido`. Desta forma, ao procurar por um utente, se existir o predicado `nulo(sc_desconhecido)`, nunca vai ser possível evoluir as informações relativas ao número de Segurança Social deste utente.

```
% Nao se sabe nem e possivel saber qual o numero de Seguranca Social do
    utente #17
utente(17, sc_desconhecido, 'Diogo Fernandes', date(1995, 9, 5),
       'diogofernandes@gmail.com', 924581465, 'Lisboa', [], 4).
nulo(sc_desconhecido).
excecao(utente(ID, _, NOME, DN, EMAIL, TLF, M, P, DC, CS)) :-
    utente(ID, sc_desconhecido, NOME, DN, EMAIL, TLF, M, P, DC, CS).
```

Listing 17: Conhecimento Imperfeito Interdito relativo ao utente

Centro de Saúde

De forma análoga, neste caso de estudo temos, por exemplo, a situação de existir um centro de saúde do qual não se conhece o telefone – representado por `tlf_desconhecido`. Como neste tipo de conhecimento é impossível alguma vez descobrir o telefone do centro de saúde, torna-se necessário criar um invariante que não permita a evolução deste conhecimento (abordado secção 3.6) e para isso, criou-se um predicado nulo para identificar o `tlf_desconhecido`. Desta forma, ao procurar por um centro de saúde, se existir o predicado `nulo(tlf_desconhecido)`, nunca vai ser possível evoluir as informações relativas ao telefone deste centro de saúde.

```
% Nao se sabe nem e possivel saber o telefone do centro de saude #8
centro_saude(8, 'Hospital da Misericordia de Evora', 'Evora',
             tlf_desconhecido,
             'hospitaldamisericordiadeevora@gmail.com').
nulo(tlf_desconhecido).
excecao(centro_saude(ID, NOME, M, _, EMAIL)) :-
    centro_saude(ID, NOME, M, tlf_desconhecido, EMAIL).
```

Listing 18: Conhecimento Imperfeito Interdito relativo ao centro de saúde

Staff

Neste caso de estudo temos, por exemplo, a situação de existir um funcionário de um centro de saúde do qual não se conhece o email – representado por `email_desconhecido`. Como neste tipo de conhecimento é impossível alguma vez descobrir o *email* do funcionário, torna-se necessário criar um invariante que não permita a evolução deste conhecimento (abordado secção 3.6) e para isso, criou-se um predicado nulo para identificar o `email_desconhecido`. Assim, ao procurar por um funcionário, se existir o predicado `nulo(email_desconhecido)`, nunca vai ser possível evoluir as informações relativas ao *email* deste funcionário.

```
% Nao se conhece nem e possivel conhecer o email do Rafael
staff(15, 1, 'Rafael Costa', email_desconhecido).
nulo(email_desconhecido).
excecao(staff(IDS, IDCENTRO, NOME, _)) :-
    staff(IDS, IDCENTRO, NOME, email_desconhecido).
```

Listing 19: Conhecimento Imperfeito Interdito relativo ao *staff* do centro de saúde

Vacinação

Pode também a situação de existir registo de vacinação do qual não se conhece qual a vacina administrada – representada por `vac_desconhecida`. Como neste tipo de conhecimento é impossível alguma vez descobrir qual a vacina administrada, torna-se necessário criar um invariante que não permita a evolução deste conhecimento (abordado secção 3.6) e para isso, criou-se um predicado nulo para identificar a `vac_desconhecida`. Assim, ao procurar por um registo de vacinação, se existir o predicado `nulo(vac_desconhecida)`, nunca vai ser possível evoluir as informações relativas ao registo de vacinação.

```
% Nao se conhece nem e possivel conhecer qual a vacina administrada
vacinacao_covid(3, 12, date(2021, 7, 17), vac_desconhecida, 1).
nulo(vac_desconhecida).
excecao(vacinacao_Covid(STAFF, UTENTE, DATA, _, TOMA)) :-
    vacinacao_Covid(STAFF, UTENTE, DATA, vac_desconhecida, TOMA).
```

Listing 20: Conhecimento Imperfeito Interdito relativo aos registos de vacinação

Médico

Considere-se também a situação de existir um médico do qual não se conhece qual o centro de saúde em que exerce funções – representado por `centro_desconhecido`. Como neste tipo de conhecimento é impossível alguma vez descobrir qual o centro de saúde em que este exerce funções, torna-se necessário criar um invariante que não permita a evolução deste conhecimento (abordado secção 3.6) e para isso, criou-se um predicado nulo para identificar o `centro_desconhecido`. Assim, ao procurar por um médico, se existir o predicado `nulo(centro_desconhecido)`, nunca vai ser possível evoluir as informações relativas ao médico.

```
% Nao se sabe nem e possivel saber em que centro de saude o medico #10
    exerce funcoes
medico(10, centro_desconhecido, 'Rui Costa', 'ruicosta@gmail.com',
       'Otorrinolaringologia').
nulo(centro_desconhecido).
excecao(medico(ID, _, NOME, EMAIL, ESP)) :-
    medico(ID, centro_desconhecido, NOME, EMAIL, ESP).
```

Listing 21: Conhecimento Imperfeito Interdito relativo ao médico

Consulta

Neste caso de estudo temos, por exemplo, a situação de não se saber qual o utente que foi a uma consulta – representado por `ut_desconhecido`. Como neste tipo de conhecimento é impossível alguma vez descobrir qual o utente em questão, torna-se necessário criar um invariante que não permita a evolução deste conhecimento (abordado secção 3.6) e para isso, criou-se um predicado nulo para identificar o `ut_desconhecido`. Assim, ao procurar por uma consulta, se existir o predicado `nulo(ut_desconhecido)`, nunca vai ser possível evoluir as informações relativas à consulta em questão.

```
% Nao se sabe nem e possivel saber o utente que foi a consulta
consulta(4, ut_desconhecido, 2, date(2021,1,15)).
nulo(ut_desconhecido).
excecao(consulta(IDM, _, IDC, DATA)) :-
    consulta(IDM, ut_desconhecido, IDC, DATA).
```

Listing 22: Conhecimento Imperfeito Interdito relativo à consulta

Tratamento

Por fim, existe também a situação de não se saber qual o centro de saúde em que um tratamento foi realizado – representado por `cs_desconhecido`. Como neste tipo de conhecimento é impossível alguma vez descobrir qual o centro de saúde em questão, torna-se necessário criar um invariante que não permita a evolução deste conhecimento (abordado secção 3.6) e para isso, criou-se um predicado nulo para identificar o `cs_desconhecido`. Assim, ao procurar por um tratamento, se existir o predicado `nulo(cs_desconhecido)`, nunca vai ser possível evoluir as informações relativas à consulta em questão.

```
% Nao se conhece nem e possivel conhecer qual o centro de saude em que o
    tratamento foi realizado
tratamento(7, 5, cs_desconhecido, date(2020,11,20), 'Exame Pulmonar').
nulo(cs_desconhecido).
excecao(tratamento(IDS, IDU, IDC, _, DSCR)) :-
    tratamento(IDS, IDU, cs_desconhecido, DATA, DSCR).
```

Listing 23: Conhecimento Imperfeito Interdito relativo ao tratamento

3.4 Invariantes

Para que a Base de Conhecimento funcione corretamente, é indispensável garantir certas condições responsáveis pelo controlo da inserção e remoção de conhecimento.

Para esse efeito, foi desenvolvido, já na primeira fase do trabalho prático, uma série de invariantes responsáveis por este controlo que optaremos por não tornar a incluir neste relatório.

Contudo, para além deste controlo da inserção e remoção, acaba por ser também imperativo garantir igualmente o controlo da:

— Inserção de conhecimento perfeito positivo e negativo;

— Inserção de conhecimento imperfeito interdito.

3.4.1 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 de seguida.

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

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

Listing 24: Invariantes relativos ao conhecimento repetido

Por outro lado, foram também criados invariantes que não permitem que seja adicionado conhecimento perfeito positivo que contradiz conhecimento perfeito negativo presente na base de conhecimento e vice-versa. Estes invariantes, apresentados de seguida, são essenciais para que a informação existente seja coerente e não seja contraditória.

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

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

Listing 25: Invariante relativos ao conhecimento contraditório

3.5 Invariantes Estruturais e Referenciais

Os invariantes são completamente indispensáveis para um correto funcionamento de todo o sistema. Apenas com a sua introdução é possível efetuar um controlo da informação que inserida e removida, criando, assim, um intermediário necessário para que os predicados de evolução e retrocesso funcionem como ambicionado.

Para esta gestão de informação se poder processar da maneira mais adequada e lógica face ao sistema, foram criados, na primeira fase do trabalho prático, invariantes associados tanto à inserção como à remoção de conhecimento.

Importa salientar que, em relação ao trabalho desenvolvido na primeira fase, foi acrescentado um invariante que permite a validação da data de nascimento do utente, sendo para que o definir foi necessário definir primeiro o predicado `verificaData`, apresentado em anexo (secção 5.1). Além disso, foram desenvolvidos invariantes relativos ao conhecimento perfeito negativo, assim como predicados no sentido de não permitir a evolução de conhecimento imperfeito interdito.

3.5.1 Invariantes Associados à Inserção de Conhecimento

No que diz respeito à inserção de conhecimento, foram criados invariantes que impedem a inserção de conhecimento repetido. Adicionalmente, foi necessário construír um conjunto de invariantes que não possibilitasse a inserção de informação impossível de se introduzir na base de conhecimento em questão.

Utente

Considerou-se que os IDs não se poderiam de forma alguma repetir uma vez que identificam univocamente cada um dos utentes. Para além disso, foi tido em consideração o facto de estes serem representados por números inteiros. Por outro lado, é necessário garantir que o centro de saúde que o utente frequenta existe de facto. Por fim, foi tido em consideração o facto de não poder existir mais do que um utente com mesmo número de Segurança Social. Todas estas restrições são impostas pelos seguintes invariantes:

```
% O ID do utente deve ser um numero inteiro e deve ser unico
+utente(ID, _, _, _, _, _, _, _, _, _) :: (
    integer(ID),
    solucoes(ID, utente(ID, _, _, _, _, _, _, _, _, _), S),
    comprimento(S, N),
    N == 1
).

% O ID do utente deve ser um numero inteiro e deve ser unico
% Conhecimento perfeito negativo
+(-utente(ID, _, _, _, _, _, _, _, _, _)) :: (
    integer(ID),
    solucoes(ID, utente(ID, _, _, _, _, _, _, _, _, _), S),
    comprimento(S, N),
    N == 1
).
```

```
% O utente tem de estar associado a um centro de saude que exista
+utente(_, _, _, _, _, _, _, _, _, CS) :: (
    solucoes(CS, centro_saude(CS, _, _, _, _), S),
    comprimento(S, N),
    N == 1
).

% O utente tem de estar associado a um centro de saude que exista
% Conhecimento perfeito negativo
+(-utente(_, _, _, _, _, _, _, _, _, CS)) :: (
    solucoes(CS, centro_saude(CS, _, _, _, _), S),
    comprimento(S, N),
    N == 1
).

% So pode existir um utente com um determinado numero de Seguranca Social
+utente(_, NUM, _, _, _, _, _, _, _, _) :: (
    solucoes(NUM, utente(_, NUM, _, _, _, _, _, _, _, _), S),
    comprimento(S, N),
    N == 1
).

% So pode existir um utente com um determinado numero de Seguranca Social
% Conhecimento perfeito negativo
+(-utente(_, NUM, _, _, _, _, _, _, _, _)) :: (
    solucoes(NUM, utente(_, NUM, _, _, _, _, _, _, _, _), S),
    comprimento(S, N),
    N == 1
).

% A data de nascimento do utente deve ser uma data valida
+utente(_, _, _, DN, _, _, _, _, _, _) :: (verificaData(DN)).

% A data de nascimento do utente deve ser uma data valida
% Conhecimento perfeito negativo
+(-utente(_, _, _, DN, _, _, _, _, _, _)) :: (verificaData(DN)).

% Invariante que impede a insercao de conhecimento perfeito positivo
    relativo a um utente com numero de Seguranca Social interdito (
    conhecimento imperfeito interdito)
+utente(ID, NUM, NOME, DN, EMAIL, TLF, M, P, DC, CS) :: (
    solucoes((ID, NUM, NOME, DN, EMAIL, TLF, M, P, DC, CS),
             (utente(ID, NUM, NOME, DN, EMAIL, TLF, M, P, DC, CS),
              nulo(NUM)), S),
    comprimento(S, N),
    N == 0
).
```

Listing 26: Invariantes de inserção relativos ao predicado `utente`

Staff

Seguindo um raciocínio idêntico, os IDs dos elementos do *staff* de um centro de saúde, além de serem representados por números inteiros, não se podem repetir. Além disso é necessário garantir que o centro de saúde em que o elemento do *staff* presta serviços existe de facto, restrições essas que são impostas pelos seguintes invariantes.

```
% O ID de cada elemento do staff do centro de saude deve ser um numero
    inteiro e deve ser unico
+staff(ID, _, _, _) :: (
    integer(ID),
    solucoes(ID, staff(ID, _, _, _), S),
    comprimento(S, N),
    N == 1
).

% O ID de cada elemento do staff do centro de saude deve ser um numero
    inteiro e deve ser unico
% Conhecimento perfeito negativo
+(-staff(ID, _, _, _)) :: (
    integer(ID),
    solucoes(ID, staff(ID, _, _, _), S),
    comprimento(S, N),
    N == 1
).

% Cada elemento do staff do centro de saude tem de estar associado a um
    centro de saude que exista
+staff(_, CS, _, _) :: (
    solucoes(CS, centro_saude(CS, _, _, _, _), S),
    comprimento(S, N),
    N == 1
).

% Cada elemento do staff do centro de saude tem de estar associado a um
    centro de saude que exista
% Conhecimento perfeito negativo
+(-staff(_, CS, _, _)) :: (
    solucoes(CS, centro_saude(CS, _, _, _, _), S),
    comprimento(S, N),
    N == 1
).
```

```
% Invariante que impede a insercao de conhecimento perfeito positivo
    relativo a um funcionario com email interdito (conhecimento imperfeito
     interdito)
+staff(IDS, IDCENTRO, NOME, EMAIL) :: (
    solucoes((IDS, IDCENTRO, NOME, EMAIL),
              (staff(IDS, IDCENTRO, NOME, EMAIL), nulo(EMAIL)), S),
    comprimento(S, N),
    N == 0
).
```

Listing 27: Invariantes de inserção relativos ao predicado `staff`

Centro de Saúde

Uma vez que o ID, representado por um número inteiro, identifica univocamente um centro de saúde, não é possível a existência de mais do que um centro de saúde com o mesmo ID, restrição essa que é imposta pelo seguinte invariante:

```
% O ID do centro de saude deve ser um numero inteiro e deve ser unico
+centro_saude(ID, _, _, _, _) :: (
    integer(ID),
    solucoes(ID, centro_saude(ID, _, _, _, _), S),
    comprimento(S, N),
    N == 1
).

% O ID do centro de saude deve ser um numero inteiro e deve ser unico
% Conhecimento perfeito negativo
+(-centro_saude(ID, _, _, _, _)) :: (
    integer(ID),
    solucoes(ID, centro_saude(ID, _, _, _, _), S),
    comprimento(S, N),
    N == 1
).

% Invariante que impede a insercao de conhecimento perfeito positivo
    relativo a um centro de saude com telefone interdito (conhecimento
    imperfeito interdito)
+centro_saude(ID, NOME, M, TLF, EMAIL) :: (
    solucoes((ID, NOME, M, TLF, EMAIL),
             (centro_saude(ID, NOME, M, TLF, EMAIL),
              nulo(TLF)), S),
    comprimento(S, N),
    N == 0
).
```

Listing 28: Invariante de inserção relativo ao predicado `centro_saude`

Vacinação

De forma a que um registo de vacinação seja válido, este tem de estar associado a um elemento do *staff* e a um utente que, de facto, existam na base de conhecimento. Além disso, a toma da vacina só pode assumir os valores 1 e 2, sendo estritamente necessário que a segunda toma ocorra após a primeira. Note-se também que não é permitida a inserção de registos repetidos. Todas estas restrições são impostas pelos seguintes invariantes:

```
% Nao permite a insercao de registos duplicados
+vacinacao_Covid(STAFF, UTENTE, DATA, VACINA, TOMA) :: (
    solucoes((STAFF, UTENTE, DATA, VACINA, TOMA),
             vacinacao_Covid(STAFF, UTENTE, DATA, VACINA, TOMA), S),
    comprimento(S, N),
    N == 1
).

% Nao permite a insercao de registos duplicados
% Conhecimento perfeito negativo
+(-vacinacao_Covid(STAFF, UTENTE, DATA, VACINA, TOMA)) :: (
    solucoes((STAFF, UTENTE, DATA, VACINA, TOMA),
             vacinacao_Covid(STAFF, UTENTE, DATA, VACINA, TOMA), S),
    comprimento(S, N),
    N == 1
).

% O registo de vacinacao tem de estar associado a um elemento do staff
   que exista
+vacinacao_covid(IDS, _, _, _) :: (
    solucoes(IDS, staff(IDS, _, _, _), S),
    comprimento(S, N),
    N == 1
).

% O registo de vacinacao tem de estar associado a um elemento do staff
   que exista
% Conhecimento perfeito negativo
+(-vacinacao_covid(IDS, _, _, _)) :: (
    solucoes(IDS, staff(IDS, _, _, _), S),
    comprimento(S, N),
    N == 1
).

% O registo de vacinacao tem de estar associado a um utente que existe
+vacinacao_covid(_, IDU, _, _, _) :: (
    solucoes(IDU, utente(IDU, _, _, _, _, _, _, _, _), S),
    comprimento(S, N),
    N == 1
).
```

```
% O registo de vacinacao tem de estar associado a um utente que existe
% Conhecimento perfeito negativo
+(-vacinacao_covid(_, IDU, _, _, _)) :: (
    solucoes(IDU, utente(IDU, _, _, _, _, _, _, _, _, _), S),
    comprimento(S, N),
    N == 1
).

% A toma da vacina so pode assumir os valores 1 e 2
+vacinacao_covid(_, _, _, _, T) :: (T > 0, T =< 2).

% A toma da vacina so pode assumir os valores 1 e 2
% Conhecimento perfeito negativo
+(-vacinacao_covid(_, _, _, _, T)) :: (T > 0, T =< 2).

% A segunda toma da vacina so pode ser adminstrada apos a primeira
+vacinacao_covid(_, IDU, _, _, 2) :: (
    solucoes(IDU, vacinacao_covid(_, IDU, _, _, 1), S),
    comprimento(S, N),
    N == 1
).

% A segunda toma da vacina so pode ser adminstrada apos a primeira
% Conhecimento perfeito negativo
+(-vacinacao_covid(_, IDU, _, _, 2)) :: (
    solucoes(IDU, vacinacao_covid(_, IDU, _, _, 1), S),
    comprimento(S, N),
    N == 1
).

% Invariante que impede a insercao de conhecimento perfeito positivo
    relativo a um registo de vacinacao com vacina interdita (conhecimento
    imperfeito interdito)
+vacinacao_Covid(STAFF, UTENTE, DATA, VACINA, TOMA) :: (
    solucoes((STAFF, UTENTE, DATA, VACINA, TOMA),
             (vacinacao_Covid(STAFF, UTENTE, DATA, VACINA, TOMA),
              nulo(VACINA)), S),
    comprimento(S, N),
    N == 0
).
```

Listing 29: Invariantes de inserção relativos ao predicado `vacinacao_covid`

Médico

O ID, representado por um número inteiro, identifica univocamente um médico e, por isso, não é possível a existência de mais do que um médico com o mesmo ID. Por outro lado, é imperativo que o médica exerça funções num centro de saúde que, de facto, exista.

```
% O ID do medico deve ser um numero inteiro e este deve ser unico
+medico(ID, _, _, _, _) :: (
    integer(ID),
    solucoes(ID, medico(ID, _, _, _, _), S),
    comprimento(S, N),
    N == 1
).

% O ID do medico deve ser um numero inteiro e este deve ser unico
% Conhecimento perfeito negativo
+(-medico(ID, _, _, _, _)) :: (
    integer(ID),
    solucoes(ID, medico(ID, _, _, _, _), S),
    comprimento(S, N),
    N == 1
).

% O medico tem de estar associado a um centro de saude que existe
+medico(_, CS, _, _, _) :: (
    solucoes(CS, centro_saude(CS, _, _, _, _), S),
    comprimento(S, N),
    N == 1
).

% O medico tem de estar associado a um centro de saude que existe
% Conhecimento perfeito negativo
+(-medico(_, CS, _, _, _)) :: (
    solucoes(CS, centro_saude(CS, _, _, _, _), S),
    comprimento(S, N),
    N == 1
).

% Invariante que impede a insercao de conhecimento perfeito positivo
    relativo a um medico com centro de saude interdito (conhecimento
    imperfeito interdito)
+medico(ID, IDCENTRO, NOME, EMAIL, ESP) :: (
    solucoes((ID, IDCENTRO, NOME, EMAIL, ESP),
             (medico(ID, IDCENTRO, NOME, EMAIL, ESP),
              nulo(IDCENTRO)), S),
    comprimento(S, N),
    N == 0
).
```

Listing 30: Invariantes de inserção relativos ao predicado `medico`

Consulta

De forma a que o registo de uma consulta seja válido, este deve estar associado a um utente, um médico e a um centro de saúde que, de facto, existam na base de conhecimento. Por fim, não é admitida a inserção de conhecimento repetido.

```
% Nao permite a insercao de registos duplicados
+consulta(IDM, IDU, IDC, DATA) :: (
    solucoes((IDM, IDU, IDC, DATA), consulta(IDM, IDU, IDC, DATA), S),
    comprimento(S, N),
    N == 1
).

% Nao permite a insercao de registos duplicados
% Conhecimento perfeito negativo
+(-consulta(IDM, IDU, IDC, DATA)) :: (
    solucoes((IDM, IDU, IDC, DATA), consulta(IDM, IDU, IDC, DATA), S),
    comprimento(S, N),
    N == 1
).

% O registo de uma consulta tem de estar associado a um medico, utente e
    centro de saude que existam
+consulta(IDM, IDU, CS, _) :: (
    solucoes((IDM, IDU, CS), (medico(IDM, CS, _, _, _),
                              utente(IDU, _, _, _, _, _, _, _, _, CS),
                              centro_saude(CS, _, _, _, _)), S),
    comprimento(S, N),
    N == 1
).

% O registo de uma consulta tem de estar associado a um medico, utente e
    centro de saude que existam
% Conhecimento perfeito negativo
+(-consulta(IDM, IDU, CS, _)) :: (
    solucoes((IDM, IDU, CS), (medico(IDM, CS, _, _, _),
                              utente(IDU, _, _, _, _, _, _, _, _, CS),
                              centro_saude(CS, _, _, _, _)), S),
    comprimento(S, N),
    N == 1
).
```

```
% Invariante que impede a insercao de conhecimento perfeito positivo
    relativo a uma consulta com utente interdito (conhecimento imperfeito
    interdito)
+consulta(IDM, IDU, IDC, DATA) :: (
    solucoes((IDM, IDU, IDC, DATA),
             (consulta(IDM, IDU, IDC, DATA), nulo(IDU)), S),
    comprimento(S, N),
    N == 0
).
```

Listing 31: Invariantes de inserção relativos ao predicado `consulta`

Tratamento

De forma análoga, o registo de um tratamento deve estar associado a um utente, um elemento do *staff* e a um centro de saúde que existam efetivamente na base de conhecimento. Neste caso, também não é permitida a inserção de conhecimento repetido.

```
% Nao permite a insercao de registos duplicados
+tratamento(IDS, IDU, IDC, DATA, DESCR) :: (
    solucoes((IDS, IDU, IDC, DATA, DESCR),
             tratamento(IDS, IDU, IDC, DATA, DESCR), S),
    comprimento(S, N),
    N == 1
).

% Nao permite a insercao de registos duplicados
% Conhecimento perfeito negativo
+(-tratamento(IDS, IDU, IDC, DATA, DESCR)) :: (
    solucoes((IDS, IDU, IDC, DATA, DESCR),
             tratamento(IDS, IDU, IDC, DATA, DESCR), S),
    comprimento(S, N),
    N == 1
).

% O registo de um tratamento tem de estar associado a um elemento do
    staff, utente e centro de saude que existam
+tratamento(IDS, IDU, CS, _, _) :: (
    solucoes((IDS, IDU, CS), (staff(IDS, CS, _, _),
                              utente(IDU, _, _, _, _, _, _, _, _, CS),
                              centro_saude(CS, _, _, _, _)) , S),
    comprimento(S, N),
    N == 1
).
```

```
% O registo de um tratamento tem de estar associado a um elemento do
    staff, utente e centro de saude que existam
% Conhecimento perfeito negativo
+(-tratamento(IDS, IDU, CS, _, _)) :: (
    solucoes((IDS, IDU, CS), (staff(IDS, CS, _, _),
                              utente(IDU, _, _, _, _, _, _, _, _, CS),
                              centro_saude(CS, _, _, _, _)) , S),
    comprimento(S, N),
    N == 1
).

% Invariante que impede a insercao de conhecimento perfeito positivo
    relativo a um tratamento com centro de saude interdito (conhecimento
    imperfeito interdito)
+tratamento(IDS, IDU, IDC, DATA, DSCR) :: (
    solucoes((IDS, IDU, IDC, DATA, DSCR),
             (tratamento(IDS, IDU, IDC, DATA, DSCR), nulo(IDC)), S),
    comprimento(S, N),
    N == 0
).
```

Listing 32: Invariantes de inserção relativos ao predicado `tratamento`

3.5.2 Invariantes Associados à Remoção de Conhecimento

Para a remoção de conhecimento, foi oportuno desenvolver invariantes que, perante toda a lógica do funcionamento do sistema, não permitissem a remoção de conhecimento relativo a utentes, elementos do *staff*, centros de saúde e médicos quando estes se encontrassem associados a um determinado registo de vacinação, consulta ou tratamento. Por outro lado, a remoção destes mesmos registos não é permitida.

Utente

Apenas é possível a remoção de um utente se não existirem registos de vacinação, consulta ou tratamento, restrição essa imposta pelos seguintes invariantes.

```
% Nao permite a remocao de um utente se existirem registos de vacinacao a
    si associados
-utente(ID, _, _, _, _, _, _, _, _, _) :: (
    solucoes(ID, vacinacao_Covid(_, ID, _, _, _), S),
    comprimento(S, N),
    N == 0
).

% Nao permite a remocao de um utente se existirem registos de consultas a
    si associaos
-utente(ID, _, _, _, _, _, _, _, _, _) :: (
    solucoes(ID, consulta(_, ID, _, _), S),
    comprimento(S, N),
    N == 0
).
```

```
% Nao permite a remocao de um utente se existirem
-utente(ID, _, _, _, _, _, _, _, _, _) :: (
    solucoes(ID, tratamento(_, ID, _, _, _), S),
    comprimento(S, N),
    N == 0
).
```

Listing 33: Invariantes de remoção relativos ao predicado `utente`

Staff

Foram construídos os seguintes invariantes no sentido de não permitir a remoção de um elemento do *staff* caso existam registos de vacinação ou de tratamento a si associados.

```
% Nao permite a remocao de um elemento do staff se existirem registos de
    vacinacao a si associaos
-staff(ID, _, _, _) :: (
    solucoes(ID, vacinacao_Covid(ID, _, _, _, _), S),
    comprimento(S, N),
    N == 0
).

% Nao permite a remocao de um elemento do staff se existirem registos de
    tratamentos a si associaos
-staff(ID, _, _, _) :: (
    solucoes(ID, tratamento(ID, _, _, _, _), S),
    comprimento(S, N),
    N == 0
).
```

Listing 34: Invariantes de remoção relativos ao predicado `staff`

Centro de Saúde

Para ser possível a remoção de um centro de saúde, é necessário que não existam utentes, médicos ou elementos do *staff* a si associados, restrição essa que é imposta pelos seguintes invariantes.

```
% Nao permite a remocao de um centro de saude se existirem utentes a si
    associaos
-centro_saude(CS, _, _, _, _) :: (
    solucoes(CS, utente(_, _, _, _, _, _, _, _, _, CS), S),
    comprimento(S, N),
    N == 0
).
```

```
% Nao permite a remocao de um centro de saude se existirem elementos do
    staff a si associaos
-centro_saude(CS, _, _, _, _) :: (
    solucoes(CS, staff(_, CS, _, _), S),
    comprimento(S, N),
    N == 0
).

% Nao permite a remocao de um centro de saude se existirem medicos a si
    associaos
-centro_saude(CS, _, _, _, _) :: (
    solucoes(CS, medico(_, CS, _, _, _), S),
    comprimento(S, N),
    N == 0
).
```

Listing 35: Invariantes de remoção relativos ao predicado `centro_saude`

Vacinação

O seguinte invariante garante que não é possível remover registos de vacinação.

```
% Nao permite a remocao de registos de atos de vacinacao
-vacinacao_Covid(_, _, _, _, _) :: fail.
```

Listing 36: Invariante de remoção relativos ao predicado `vacinacao_covid`

Médico

Para ser possível a remoção de um médico, é necessário que não existam registos de consultas a si associados, restrição essa que é imposta pelo seguinte invariante.

```
% Nao permite a remocao de um medico se existirem consultas a si
    associadas
-medico(ID, _, _, _, _) :: (solucoes(ID, consulta(ID, _, _, _), S),
                            comprimento(S, N),
                            N == 0).
```

Listing 37: Invariante de remoção relativo ao predicado `medico`

Consulta

O seguinte invariante garante que não é possível remover registos de consultas.

```
% Nao permite a remocao de registos de consultas
-consulta(_, _, _, _) :: fail.
```

Listing 38: Invariante de remoção relativo ao predicado `consulta`

Tratamento

O seguinte invariante garante que não é possível remover registos de tratamentos.

```
% Nao permite a remocao de registos de tratamentos
-tratamento(_, _, _, _, _) :: fail.
```

Listing 39: Invariante de remoção relativo ao predicado `tratamento`

3.6 Invariantes Associados ao Conhecimento Imperfeito Interdito

Utente

```
% Invariante que impede a insercao de conhecimento perfeito positivo
    relativo a um utente com numero de Seguranca Social interdito (
    conhecimento imperfeito interdito)
+utente(ID, NUM, NOME, DN, EMAIL, TLF, M, P, DC, CS) :: (
    solucoes((ID, NUM, NOME, DN, EMAIL, TLF, M, P, DC, CS),
             (utente(ID, NUM, NOME, DN, EMAIL, TLF, M, P, DC, CS),
              nulo(NUM)), S),
    comprimento(S, N),
    N == 0
).
```

3.7 Problemática da Evolução do Conhecimento

O tratamento da problemática da evolução do conhecimento prende-se com o facto de manter a base de conhecimento coesa e inviolável em termos de existência de conhecimento repetido, tendo em conta cada inserção ou remoção que possa acontecer.

Para que fosse possível a inserção e remoção de conhecimento na base de conhecimento, foram desenvolvidos os predicados evolução e involução. Estes predicados obrigam a que a inserção ou remoção cumpra certas regras definidas pelos invariantes.

Assim, é necessário implementar uma série de medidas que não permita:

— Remover informação que seja dependente de outra;

— Inserção de informação repetida.

De modo a garantir esta segurança, não pode ser apagada informação dependente de outra, ou seja, não pode ser removida a informação sobre um utente se este possuir registos de vacinação a si associados. Além disso, não pode ser adicionada informação repetida, uma vez que não traz adição de conhecimento. Assim, no momento de alteração de informação, é necessário testar se esta corrompe a base de conhecimento. Estes testes são efetuados através do uso de invariantes, previamente explicados

3.8 Evolução de Conhecimento

3.8.1 Conhecimento Perfeito Positivo

Numa primeira fase, e de maneira a todo o conhecimento positivo ser evoluído, ou seja, ser adicionado à base de conhecimento, foi criado o seguinte predicado `evolucao`. Para ser possível adicionar conhecimento, este predicado verifica primeiro se o elemento a ser adicionado respeita todos os invariantes e, caso seja verdade então este é adicionado à base de conhecimento.

```
% Extensao do meta-predicado evolucao: T -> {V, F}
% Conhecimento perfeito positivo
evolucao(T) :- solucoes(I, +T::I, L),
               insercao(T),
               teste(L).
```

Listing 40: Evolução de Conhecimento Perfeito Positivo

3.8.2 Conhecimento Perfeito Negativo

De maneira a ser exequível a adição de conhecimento perfeito negativo na base de conhecimento, o predicado criado é em tudo semelhante ao interior, mas neste caso é necessário especificar que o conhecimento a adicionar é negativo. Neste caso, os invariantes a respeitar são os que dizem respeito à inserção de conhecimento negativo.

```
% Extensao do meta-predicado evolucao: T -> {V, F}
% Conhecimento perfeito negativo
evolucao(-T) :- solucoes(I, +(-T)::I, L),
                insercao(-T),
                teste(L).
```

Listing 41: Evolução de Conhecimento Perfeito Negativo

3.9 Involução de Conhecimento

3.9.1 Conhecimento Perfeito Positivo

Foi criado o predicado `involucao` análogo ao das aulas práticas de forma a permitir a remoção de conhecimento da base de conhecimento, verificando todos os invariantes necessários. Assim, só em caso de sucesso é que o conhecimento é, de facto, removido.

```
% Extensao do meta-predicado involucao: T -> {V, F}
% Conhecimento perfeito positivo
involucao(T) :- solucoes(I, -T::I, L),
                teste(L),
                remocao(T).
```

Listing 42: Involução de Conhecimento Perfeito Positivo

3.9.2 Conhecimento Perfeito Negativo

De forma análoga, para permitir também a involução do conhecimento negativo, foi criado outro predicado `involucao` mas com a particularidade de ser necessário especificar que se trata de conhecimento negativo.

```
% Extensao do meta-predicado involucao: T -> {V, F}
% Conhecimento perfeito negativo
involucao(-T) :- solucoes(I, -(-T)::I, L),
                 teste(L),
                 remocao(-T).
```

Listing 43: Involução de Conhecimento Perfeito Negativo

3.10 Sistema de Inferência

Um sistema de inferência capaz de implementar os mecanismos de raciocínio inerentes aos sistemas que anteriormente foram representados é um passo indispensável ser efetuado uma vez que é com este sistema que podemos "criar"uma espécie de interpretador de questões.

Assim, a implementação do sistema de inferência começou pela criação do meta-predicado `demo: Questao, Resposta -> {V, F}`, que, para uma determinada questão, e coloca na Resposta, um dos valores do conjunto de soluções – verdadeiro, falso e desconhecido.

A determinação da resposta sobre a questão colocada através do meta-predicado `demo` é verdadeira caso exista conhecimento positivo sobre a mesma, falsa caso exista conhecimento negativo, e desconhecida caso não exista nenhum dos conhecimentos.

Em termos práticos, considerando uma questão abstrata $q(X)$, em que X pode tomar a forma $X_1, X_2, ..., X_n$, definem-se as seguintes três respostas possíveis para essa questão:

— Verdadeiro se $\exists X\ q(X)$

— Falso se $\exists X\ \neg q(X)$

— Desconhecido caso contrário

Neste sentido, foi desenvolvido interpretador de questões. Este interpretador de questões trata-se de um predicado que se baseia nos diferentes tipos de conhecimento imperfeito.

O predicado `demo` apresenta uma resposta, mediante uma questão, determinando o seu valor de verdade.

```
% Extensao do meta-predicado demo: Questao, Resposta -> {V, F, D}
demo(Questao, verdadeiro) :- Questao.
demo(Questao, falso) :- -Questao.
demo(Questao, desconhecido) :- nao(Questao), nao(-Questao).
```

Listing 44: Extensão do meta-predicado `demo`

Para complementar esse sistema, foram construídos também predicados que permitem fazer conjunções e/ou disjunções de duas ou mais *queries*.

Disjunção

P	Q	$P \vee Q$
Verdadeiro	-	Verdadeiro
-	Verdadeiro	Verdadeiro
Falso	Falso	Falso
Falso	Desconhecido	Desconhecido
Desconhecido	Falso	Desconhecido
Desconhecido	Desconhecido	Desconhecido

Tabela 1: Tabela de verdade da disjunção

Conjunção

P	Q	$P \wedge Q$
Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Desconhecido	Desconhecido
Desconhecido	Verdadeiro	Desconhecido
Desconhecido	Desconhecido	Desconhecido
Falso	-	Falso
-	Falso	Falso

Tabela 2: Tabela de verdade da conjunção

Assim, com o auxílio dos predicados `disjuncao` e `conjuncao`, desenvolveu-se o predicado `demoComp`, que extende a funcionalidade do meta-predicado `demo`, permitindo interpretas uma lista de questões.

```
% Extensao do predicado disjuncao: X, Y, R -> {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).
```

Listing 45: Extensão do predicado `disjuncao`

```
% Extensao do predicado conjuncao: X, Y, R -> {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).
```

Listing 46: Extensão do predicado `conjuncao`

```
% Extensao do predicado demoComp: Lista, R -> {V, F, D}
demoComp([Q], R) :- demo(Q,R).
demoComp([Q1, ou, Q2|T], R) :- demo(Q1, R1),
                               demoComp([Q2|T], R2),
                               disjuncao(R1, R2, R).
demoComp([Q1, e, Q2|T], R) :- demo(Q1, R1),
                              demoComp([Q2|T], R2),
                              conjuncao(R1,R2,R).
```

Listing 47: Extensão do predicado `demoComp`

4 Conclusão e Trabalho Futuro

No final deste trabalho prático, foram implementadas todas as funcionalidades requisitadas, permitindo a representação de todo o tipo de conhecimento estudado, tanto perfeito (positivo, negativo) como imperfeito (incerto, impreciso e interdito), com vários exemplos e demonstrações de cada um deles.

Contrariamente ao que foi feito na primeira fase do trabalho prático, nesta o principal foco foi, não apostar tanto na diversidade de predicados capazes de explorar o sistema em questão, mas garantir que se representava o conhecimento de forma completa, exemplificando explicitamente, para os quatro predicados base do enunciado do problema, todos os tipos de conhecimento estudados.

Como trabalho futuro, poderia abordar-se a problemática da evolução e involução de conhecimento imperfeito. Além disso, a base de conhecimento poderia ser mais estendida, ainda que o trabalho proposto seja suficiente para demonstrar todos os tipos de conhecimento imperfeito bem como os invariantes desenvolvidos para a gestão da base de conhecimento.

5 Anexos

5.1 Predicados Auxiliares

Para o desenvolvimento dos procedimentos requeridos pelo sistema de representação de conhecimento e raciocínio, foi necessário recorrer a vários predicados auxiliares que foram extremamente úteis no decorrer de todo o processo. Em seguida, apresentam-se esses predicados desenvolvidos.

Meta-predicado `nao`

O meta-predicado `nao` 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 – *i.e.* na ausência de prova – dá `yes`.

```
% Extensao do meta-predicado nao: Questao -> {V, F}
% Negacao fraca
nao(Q) :- Q, !, fail.
nao(_).
```

Listing 48: Extensão do meta-predicado `nao`

Predicado `comprimento`

O predicado comprimento coloca em `R` o comprimento da lista passada como argumento.

```
% Extensao do predicado comprimento: S, N -> {V, F}
comprimento(S, N) :- length(S, N).
```

Listing 49: Extensão do predicado `comprimento`

Meta-predicado `solucoes`

Utiliza-se este predicado quando se pretende obter a listagem de todas as soluções possíveis `Z`, para uma dada questão `Y`, cujo formato da lista é especificado por `X`. Faz-se uso do predicado disponibilizado pelo `PROLOG`, `findall`, uma vez que este não falha na eventualidade de não existir resposta a esta questão, ao contrário do que aconteceria com o predicado `bagof`.

```
% Extensao do meta-predicado solucoes: X, Y, Z -> {V, F}
solucoes(X, Y, Z) :- findall(X, Y, Z).
```

Listing 50: Extensão do meta-predicado `solucoes`

Meta-predicado `teste`

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

```
% Extensao do meta-predicado teste: L -> {V, F}
teste([]).
teste([H|T]) :- H, teste(T).
```

Listing 51: Extensão do meta-predicado `teste`

Meta-predicado `insercao` e `remocao`

O meta-predicado `insercao` coloca `T` na base de conhecimento no caso de sucesso, retornando `yes`, e retira `T` no caso de haver retrocesso, retornando `no`. O meta- predicado `remocao`, por sua vez, faz o oposto, ou seja, remove `T` da base de conhecimento no caso de sucesso, retornando `yes` e adiciona `T` no caso de haver retrocesso, retornando `no`.

```
% Extensao do meta-predicado insercao: T -> {V, F}
insercao(T) :- assert(T).
insercao(T) :- retract(T), !, fail.
```

Listing 52: Extensão do meta-predicado `insercao`

```
% Extensao do meta-predicado remocao: T -> {V, F}
remocao(T) :- retract(T).
remocao(T) :- assert(T), !, fail.
```

Listing 53: Extensão do meta-predicado `remocao`

Predicado `verificaData`

O predicado `verificaData` permite determinar se uma determinada data é válida.

```
% Extensao do predicado verificaData: date(D, M, A) -> {V, F}
verificaData(date(D, 1, A)) :- D > 0, D =< 31, A > 0.
verificaData(date(D, 2, A)) :- D > 0, A mod 4 =:= 0, D =< 29, A > 0.
verificaData(date(D, 2, A)) :- D > 0, A mod 4 =\= 0, D =< 28, A > 0.
verificaData(date(D, 3, A)) :- D > 0, D =< 31, A > 0.
verificaData(date(D, 4, A)) :- D > 0, D =< 30, A > 0.
verificaData(date(D, 5, A)) :- D > 0, D =< 31, A > 0.
verificaData(date(D, 6, A)) :- D > 0, D =< 30, A > 0.
verificaData(date(D, 7, A)) :- D > 0, D =< 31, A > 0.
verificaData(date(D, 8, A)) :- D > 0, D =< 31, A > 0.
verificaData(date(D, 9, A)) :- D > 0, D =< 30, A > 0.
verificaData(date(D, 10, A)) :- D > 0, D =< 31, A > 0.
verificaData(date(D, 11, A)) :- D > 0, D =< 30, A > 0.
verificaData(date(D, 12, A)) :- D > 0, D =< 31, A > 0.
```

Listing 54: Extensão do predicado `verificaData`

5.2 Base de Conhecimento

5.2.1 Conhecimento Perfeito Positivo

```
% Extensao do predicado utente: #Idutente, No Seguranca_Social, Nome,
                                 Data_Nasc, Email, Telefone, Morada,
                                 Profissao, [Doencas_Cronicas],
                                 #CentroSaude -> {V, F, D}

utente(1, 14492, 'Duarte Carvalho', date(1998,4,21),
       'duartecarvalho@gmail.com', 91761416, 'Lisboa', 'Estudante',
      ['Asma'], 4).
utente(2, 95110, 'Mariana Pereira', date(1988,11,12),
       'marianapereira@gmail.com', 915992520, 'Porto', 'Atleta', [], 4).
utente(3, 79297, 'Teresa Marques', date(1987,8,30),
       'teresamarques@gmail.com', 913844675, 'Viseu', 'Engenheira',
       ['Hipertensao'], 4).
utente(4, 71723, 'Sofia Soares', date(1979,6,8), 'sofiasoares@gmail.com',
       919555582, 'Porto', 'Advogada', ['Diabetes'], 2).
utente(5, 40203, 'Rui Rocha', date(1987,12,24), 'ruirocha@gmail.com',
       919219565, 'Braga', 'Atleta', [], 1).
utente(6, 77645, 'Joao Martins', date(1998,11,12), 'joaomartins@gmail.com
       ', 917630282, 'Coimbra', 'Estudante', ['Asma'], 2).
utente(7, 67275, 'Diogo Rodrigues', date(1995,1,25),
       'diogorodrigues@gmail.com', 916543686, 'Lisboa', 'Engenheiro',
       ['Diabetes','Cancro'], 2).
utente(8, 76991, 'Francisca Lopes', date(1986,9,15),
       'franciscalopes@gmail.com', 913672965, 'Braga', 'Enfermeira',
       ['Asma'], 2).
utente(9, 82539, 'Tiago Lima', date(1978,12,12), 'tiagolima@gmail.com',
       91683448, 'Lisboa', 'Medico', ['Hipertensao'], 5).
utente(10, 16086, 'Daniela Macedo', date(1977,10,25),
       'danielamacedo@gmail.com', 911216397, 'Coimbra', 'Medico', [], 3).
utente(11, 56418, 'Daniel Santos', date(1955, 2, 23),
       'danielsantos@gmail.com', 915632478, 'Faro', 'Advogado', [], 4).
```

```
% Extensao do predicado centro_saude: #Idcentro, Nome, Morada, Telefone,
                                        Email -> {V, F, D}

centro_saude(1, 'Hospital de Braga', 'Braga', 253027000,
             'hospitaldebraga@gmail.com').
centro_saude(2, 'Hospital Da Senhora Da Oliveira', 'Guimaraes',
             253540330, 'hospitaldasenhoradaoliveira@gmail.com').
centro_saude(3, 'Centro Hospitalar e Universitario de Coimbra',
             'Coimbra', 239400400,
             'centrohospitalareuniversitariodecoimbra@gmail.com').
centro_saude(4, 'Hospital de Santa Maria', 'Lisboa', 217805000,
             'hospitaldesantamaria@gmail.com').


% Extensao do predicado staff: #Idstaff, #Idcentro, Nome,
                                  Email -> {V, F, D}

staff(1, 4, 'Teresa Rodrigues', 'teresarodrigues@gmail.com').
staff(2, 3, 'Diogo Martins', 'diogomartins@gmail.com').
staff(3, 2, 'Daniela Marques', 'danielamarques@gmail.com').
staff(4, 2, 'Joao Lopes', 'joaolopes@gmail.com').
staff(5, 3, 'Rui Lima', 'ruilima@gmail.com').
staff(6, 4, 'Mariana Soares', 'marianasoares@gmail.com').
staff(7, 1, 'Duarte Pereira', 'duartepereira@gmail.com').
staff(8, 4, 'Francisca Lima', 'franciscalima@gmail.com').
staff(9, 3, 'Sofia Macedo', 'sofiamacedo@gmail.com').
staff(10, 1, 'Tiago Carvalho', 'tiagocarvalho@gmail.com').


% Extensao do predicado vacinacao_Covid: #Idstaff, #Idutente, Data,
                                          Vacina, Toma -> {V, F, D}

vacinacao_Covid(7, 1, date(2021,4,23), 'Pfizer', 1).
vacinacao_Covid(5, 1, date(2021,8,24), 'Pfizer', 2).
vacinacao_Covid(7, 2, date(2020,2,1), 'Astrazeneca', 1).
vacinacao_Covid(10, 3, date(2020,10,13), 'Pfizer', 1).
vacinacao_Covid(9, 4, date(2020,11,11), 'Pfizer', 1).
vacinacao_Covid(6, 4, date(2020,12,19), 'Pfizer', 2).
vacinacao_Covid(1, 5, date(2020,3,21), 'Astrazeneca', 1).
vacinacao_Covid(5, 5, date(2020,7,30), 'Astrazeneca', 2).
vacinacao_Covid(8, 6, date(2020,5,3), 'Pfizer', 1).
vacinacao_Covid(7, 6, date(2020,10,20), 'Pfizer', 2).
vacinacao_Covid(7, 8, date(2021,7,17), 'Pfizer', 1).
vacinacao_Covid(9, 9, date(2020,2,21), 'Astrazeneca', 1).
vacinacao_Covid(3, 10, date(2020,9,29), 'Pfizer', 1).
vacinacao_Covid(1, 10, date(2020,12,13), 'Pfizer', 2).
```

```
% Extensao do predicado medico: #Idmedico, #Idcentro, Nome, Email,
                                Especialidade -> {V, F, D}

medico(1, 3, 'Goncalo Santos', 'goncalosantos@gmail.com', 'Cardiologia').
medico(2, 3, 'Roberto Moreira', 'robertomoreira@gmail.com',
       'Anestesiologia').
medico(3, 4, 'Rui Santos', 'ruisantos@gmail.com', 'Clinica Geral').
medico(4, 2, 'Ines Nunes', 'inesnunes@gmail.com', 'Dermatologia').
medico(5, 1, 'Hugo Alves', 'hugoalves@gmail.com', 'Gastrenterologia').
medico(6, 1, 'Ricardo Sousa', 'ricardosousa@gmail.com',
       'Medicina Dentaria').

% Extensao do predicado consulta: #Idmedico, #Idutente, #Idcentro,
                                  Data -> {V, F, D}

consulta(3, 1, 4, date(2020,10,15)).
consulta(6, 5, 1, date(2021,3,2)).
consulta(4, 8, 2, date(2020,12,20)).
consulta(3, 2, 4, date(2020,8,14)).

% Extensao do predicado tratamento: #IdStaff, #Idutente, #Idcentro, Data,
                                    Descricao -> {V, F, D}

tratamento(4, 4, 2, date(2021,3,14), 'Radiografia Perna').
tratamento(6, 1, 4, date(2021,2,1), 'Eletrocardiograma').
tratamento(7, 5, 1, date(2020,5,14), 'Exame Pulmonar').
tratamento(3, 8, 2, date(2021,3,1), 'Analises Clinicas').
```

Listing 55: Povoamento inicial da Base de Conhecimento com Conhecimento Perfeito Positivo

Lista de Siglas e Acrónimos

ID Identificador

PMF Pressuposto do Mundo Fechado

Referências

Referências Bibliográficas

[1] ANALIDE, Cesar, Neves, José
"Representação de Informação Incompleta"
Texto Pedagógico, Grupo de Inteligência Artificial, Centro de Ciências e Tecnologias da Computação, Portugal, 1996

[2] ANALIDE, Cesar, NOVAIS, Paulo, Neves, José
"Sugestões para a Redacção de Relatórios Técnicos"
Relatório Técnico, Departamento de Informática, Universidade do Minho, Portugal, 2011