

UNIVERSIDADE DO MINHO
DEPARTAMENTO DE INFORMÁTICA
MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA - MIEINF



TRABALHO PRÁTICO - FASE 1

Sistemas de Representação de Conhecimento e Raciocínio

GRUPO 13



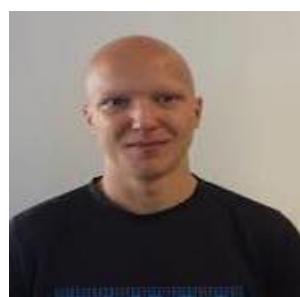
André Viera A90166



Inês Bastos A89522



Joana Sousa A83614



Tiago Gomes A78141

Maio de 2021

Conteúdo

1	Introdução	2
1.1	Contextualização	2
1.1.1	Breve resumo	2
2	Objetivos	3
2.1	Preliminares	4
2.2	Requisitos	5
3	Descrição do trabalho	6
3.1	Fundamentação teórica	6
3.1.1	Conceitos	6
3.2	Extensão à Programação em Lógica	8
3.3	Implementação	9
3.3.1	Base de Conhecimento	9
3.3.2	Programa	11
4	Resolução das Funcionalidades Propostas	23
4.1	Representação do conhecimento positivo	23
4.2	Representação do conhecimento imperfeito	24
4.2.1	Conhecimento Incerto:	24
4.2.2	Conhecimento Impreciso:	25
4.2.3	Conhecimento Interdito:	26
4.3	Manipular invariantes que designem restrições à inserção e à remoção de conhecimento do sistema	27
4.4	Desenvolver um sistema de inferência capaz de implementar os mecanismos de raciocínio inerentes a estes sistemas	36
5	Conclusão	38
6	Referências	39

Capítulo 1

Introdução

1.1 Contextualização

O presente relatório provém do projeto proposto no âmbito da Unidade Curricular de Sistemas de Representação de Conhecimento e Raciocínio que motivou a utilização da linguagem de programação em lógica *PROLOG*, no âmbito da representação de conhecimento e construção de mecanismos de raciocínio para a resolução de problemas.

PROLOG apresentou um paradigma de programação totalmente desconhecido ao nosso grupo, o que acarretou uma série de novos desafios intrigantes, colocando-nos à prova durante o desenvolvimento deste projeto que nos motivou sempre a procurar e adquirir novos conhecimentos, tal como aprofundar os conhecimentos lógicos já adquiridos até então.

1.1.1 Breve resumo

Durante a primeira fase, foi-nos proposto 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 *COVID* que a sociedade enfrenta neste momento.

Para tal efeito, foi-nos enunciado um panorama caracterizado por conhecimento, o que nos permitiu ter uma base para desenvolver o nosso sistema. Deste modo, e de forma a ser possível verificar resultados obtidos, foi também desenvolvida uma base de conhecimento base para ser possível interagir com o nosso sistema.

Após a finalização da primeira etapa deste projeto, demos continuidade utilizando a extensão à programação em lógica, no âmbito de conhecimento imperfeito, recorrendo à utilização de valores nulos e da criação de mecanismos de raciocínio adequados. Desta forma, iremos apresentar o projeto já anteriormente desenvolvido com algumas mudanças e melhorias, relativamente aos novos conhecimentos adquiridos nas aulas desta U.C.

Nos capítulos seguintes deste relatório iremos explorar todos os objetivos e requisitos necessário para implementar o nosso sistema, tal toda a descrição de toda a nossa implementação.

Capítulo 2

Objetivos

A partir da caracterização apresentada no enunciado e para a realização do trabalho, o grupo teve de construir um caso prático capaz de demonstrar as funcionalidades subjacentes à utilização da linguagem de programação em lógica *PROLOG*.

Mais concretamente, foi necessário definir um modo de representação de todas as entidades do nosso sistema, através das diversas funcionalidades do *PROLOG*, tal como representar logicamente todos os mecanismos necessários a implementar num universo de discurso na área da vacinação.

Para tal, houve uma necessidade em aprofundar o nosso conhecimento neste caso de estudo, uma vez que não se trata, de modo nenhum, da nossa área de especialização.

2.1 Preliminares

Todas as circunstâncias anteriormente apresentadas implicam que, também, seja necessário elucidar neste relatório todas as nossas pesquisas e conhecimentos adquiridos sobre este tema, de forma a, numa fase posterior, apresentar todos os nossos requisitos necessários e implementados no nosso caso de estudo.

O que é um universo de discurso na área da vacinação? Sem dúvida que se trata da questão inicial. Num contexto de pandemia mundial, o universo da população torna-se algo que toma proporções incalculáveis. Assim, foi necessário cada país efetuar diferentes planos de forma a vacinar a sua população. O nosso caso de estudo utiliza a população portuguesa como inspiração, tal como um sistema semelhante aos planos de vacinação adotados por Portugal.

O que são planos de vacinação? Mais uma vez, apesar do universo da população ter drasticamente reduzido comparativamente ao universo mundial, trata-se de um universo com uma população na ordem dos milhões. Para ser possível vacinar a totalidade da população foi necessário estabelecer diferentes fases e prioridades para cada fase. É impensável imaginar ter centros que concentrem milhares de indivíduos, tal como os recursos (vacinas) são limitados.

O que implicam os planos de vacinação? Além demais, estes recursos poderão não ser de toma única, o que implica múltiplas marcações para cada indivíduo, consoante o número de tomas. Estas marcações requerem um sistema capaz de identificar os indivíduos, quais as tomas a realizar, tal como todos os locais disponíveis para tais marcações.

Sem dúvida que, apesar de se tratar de um assunto não muito complexo, existe um grande número de decisões e variantes que condicionam um plano de vacinação de um país. Assim, e sem nunca esquecer os nossos objetivos deste trabalho, **foram supostas certas condições simplificadas da situação real que a sociedade atravessa** (que iremos expor e aprofundar nos capítulos seguintes) de forma a **não alterar a dificuldade da implementação** do nosso sistema.

Desta forma, na secção seguinte iremos apresentar e especificar todas as suposições e requisitos que estabelecemos para a implementação do nosso caso de estudo.

2.2 Requisitos

Com base no enunciado proposto pela equipa docente, existe uma série de funcionalidades base que, pelo menos, o nosso caso de estudo deverá respeitar:

- **Permitir a definição de fases de vacinação, definindo critérios de inclusão de utentes nas diferentes fases (e.g., doenças crónicas, idade, profissão):** para a nossa implementação foram supostas apenas duas fases de vacinação distintas.
- **Identificar pessoas não vacinadas:** todas as pessoas que não tomaram nenhuma dose de nenhuma vacina.
- **Identificar pessoas vacinadas:** procuramos identificar as pessoas que tomaram pelo menos uma dose de uma vacina e aquelas que tomaram duas doses de uma vacina.
- **Identificar pessoas vacinadas indevidamente:** uma pessoa vacinada indevidamente foi considerada aquela que pertence à segunda fase de vacinação e da qual existe algum registo de vacinação, antes de toda a gente da primeira fase ter tomado as duas doses.
- **Identificar pessoas não vacinadas e que são candidatas a vacinação:** existem duas fases distintas, então existem duas listas de candidatos a vacinação.
- **Identificar pessoas a quem falta a segunda toma da vacina:** todos aqueles que já tomaram a primeira dose de uma vacina.
- **Desenvolver um sistema de inferência capaz de implementar os mecanismos de raciocínio inerentes a estes sistemas:** tratando-se de uma base de conhecimento, foram aplicados mecanismos de evolução e de involução.

Para a implementação destes requisitos básicos através da linguagem de programação em lógica *PROLOG*, foi necessário o desenvolvimento de diversos predicados auxiliares, tal como fragmentar os requisitos de forma a solucionar o problema de forma simples.

Toda esta solução, através deste paradigma de programação, requereu que o nosso grupo se fundamentasse numa fundamentação teórica previamente à implementação, pelo que foi necessário estudar os conceitos da programação em lógica. Os fundamentos teóricos estudados, e que iremos explorar no presente relatório, foram:

- Termo, Predicado, Facto, Regra e Programa;
- Pressuposto do Mundo Fechado;
- Inserção e remoção da base de conhecimento;
- Invariante e sua estrutura;
- Pressuposto do Mundo aberto;
- Extensão à Programação em Lógica:
 - Conhecimento Imperfeito;
 - Sistema de Inferência;

Capítulo 3

Descrição do trabalho

3.1 Fundamentação teórica

3.1.1 Conceitos

Termo: Nesta linguagem, *PROLOG*, apenas existe um tipo de dados: *termos*. Neste tipo de dados existem diversos sub-tipos, tais como átomos (são compostos por uma sequência de caracteres e representam uma simples unidade, facilmente identificados pelo seu primeiro caractere em letra minúscula. Também é possível identificar entre apóstrofos, permitindo definir um átomo do género 'Atomo'), variáveis (utilizam sintaxe praticamente igual aos átomos, é uma sequ—encia de caracteres, mas diferem no primeiro caractere que necessariamente tem de ser maiúsculo. Uma excessão será utilizar o caractere *underscore* que permite representar qualquer variável da qual não se necessita especificar), números, ou até mesmo termos compostos.

Predicado: Os predicados são definidos por um nome e zero ou mais argumentos. No caso do nome, este é tratado como um átomo. Quanto aos argumentos, estes são representados por *termos*. A quantidade de argumentos define a aridade do predicado. Além demais, um predicado é definido por um aglomerado de cláusulas, que podem ser factos ou regras. Esta linguagem é restringida pelas Cláusulas de Horn.

Facto: Um facto define-se como sendo um predicado verdadeiro que é inserido na base de conhecimento.

Regra: Uma regra, caracteriza-se como sendo um predicado que usa variáveis, com o intuito de provar a veracidade de um facto.

Programa: Um programa é um conjunto de predicados. Logicamente, um programa em *PROLOG* afirma tudo aquilo que é válido.

Pressuposto do Mundo Fechado: Este princípio refere que, numa consulta à base de conhecimento, todo o predicado é avaliado como falso no caso não existir nenhum facto que suporte o termo proposto ou nenhuma regra positiva. Por outras palavras, infere-se que toda a informação que não consta na base de conhecimento é considerada falsa.

Inserção e remoção da base de conhecimento: A inserção e remoção de conhecimento da base de conhecimento, é feita utilizando os predicados *assert* e *retract*, respetivamente. Apesar disso, estes predicados não dão garantia de consistência e preservação de algumas restrições que se pretendam implementar, tais como, a inserção de factos repetidos ou a remoção de factos que não se encontram presentes na base de conhecimento. Como forma de contornar esta limitação, recorreu-se ao uso de invariantes.

Invariante e sua estrutura: Um invariante é algo que não apresenta alterações sempre que são aplicadas um conjunto de transformações (operações de inserção e remoção) existentes na base de conhecimento. Um invariante pode ser:

- estrutural: condiciona a base de conhecimento ao nível estrutural.
- referencial: condiciona a base de conhecimento tendo como base o significado do predicado.

Pressuposto do mundo aberto: O pressuposto de mundo aberto admite que podem existir outros factos ou conclusões verdadeiras para além daqueles representados na base de conhecimento. Generalizando, permite distinguir o que é realmente falso de algo que não está presente na base de conhecimento.

A programação em lógica mostra-se um recurso limitado, para representar este tipo de conhecimento, uma vez que as respostas às questões colocadas são do tipo **verdadeiro** ou **falso**. Face a esta limitação, surge como uma solução a criação de uma extensão do mesmo que permite a inclusão de respostas do tipo **desconhecido**.

Tomando este pressuposto como ponto de partida, qualquer questão colocada à base de conhecimento pode ter um de 3 valores diferentes:

- **Verdadeiro:** Existe conhecimento que comprova a sua veracidade.
- **Falso ou negação forte:** Existe conhecimento explícito para negar a questão.
- **Desconhecido ou negação por falha:** Não existe conhecimento, isto é, não existe informação que permita retirar uma conclusão.

3.2 Extensão à Programação em Lógica

Uma extensão de um programa em lógica envolve 2 componentes:

Representação de conhecimento, completo e imperfeito;

Sistema de inferência que permite a interpretação de conhecimento tanto perfeito como imperfeito.

Conhecimento Imperfeito: Numa extensão de um programa em lógica o conhecimento imperfeito é representado sob a forma de valores nulos. Existem 3 tipos de valores nulos:

- **Incerto:** Desconhecido, de um conjunto indeterminado de hipóteses;
- **Impreciso:** Desconhecido, mas de um conjunto determinado de hipóteses;
- **Interdito:** Desconhecido e não é permitido conhecer.

Sistema de Inferência: O sistema de inferência utilizado corresponde a uma extensão de um meta-predicado, representado na seguinte figura.

```
%-----  
% sistema de inferência que permite reconhecer se é V/F ou desconhecido  
si(Questao,verdadeiro) :- Questao.  
si(Questao,falso) :- -Questao.  
si(Questao,desconhecido) :- nao(Questao), nao(-Questao).
```

Figura 3.1: Extensão do meta-predicado si

É de salientar que para representar a negação forte, neste meta-predicado, é utilizado o “-” e a negação por falha é representada através do predicado “não”.

Após a análise e consolidação dos constructos teóricos apresentados, estão reunidas as condições para avançar na implementação e resolução dos objetivos propostos.

3.3 Implementação

A nossa implementação poderá ser repartida em dois módulos diferentes: *base de conhecimento* e o *programa* que define toda a implementação dos requisitos do trabalho prático.

3.3.1 Base de Conhecimento

A base de conhecimento foi desenvolvida de forma a podermos retratar várias situações possíveis que pudessem ocorrer no sistema, permitindo simular um bom caso de estudo para o nosso sistema.

Primeiramente, foi necessário definir diversos factos que representem toda a informação relevante num plano de vacinação, de forma a ser possível preencher a base de conhecimento. Estes factos irão representar toda a informação relativa aos utentes, aos médicos, aos enfermeiros, aos centros de saúde, ao staff e às vacinações ocorridas.

Quanto aos utentes, médicos, enfermeiros e staff foi procurado guardar informação pessoal relevante para os identificar, tal como quais os centros de saúde a que pertencem, também identificados.

Já no que retrata a informação das vacinações e num contexto de pandemia mundial, é necessário reter toda a informação que identifique quem administrou a toma (*staff*), tal como identificar o utente administrado. Sabendo que existem vários tipos de vacinas, é necessário, também, identificar esse mesmo tipo e qual a toma relacionada à vacina.

Mais concretamente, no que retrata à implementação, temos:

- **utente/10** : Identificação, Número Segurança Social, Nome, Ano de Nascimento, Email, Telefone, Cidade de Residência, Profissão, [Doenças Crónicas], Centro Saúde ↗ { V, F }
- **medico/5** : Identificação, Nome, Idade, Género, Centro de Saúde ↗ { V, F }
- **enfermeiro/5** : Identificação, Nome, Idade, Género, Centro de Saúde ↗ { V, F }
- **staff/4** : Identificação, Identificação do Centro de Saúde, Nome, Email ↗ { V, F }
- **centrosaude/5** : Identificação do Centro de Saúde, Nome, Cidade de Residência, Telefone, Email ↗ { V, F }
- **vacinacao/7** : Identificação do Staff, Identificação do Utente, Dia, Mês, Ano, Tipo de Vacina, Identificação da Dose ↗ { V, F }

Sabendo que a nossa base de conhecimento tem de estar preparada para manipulações de conhecimento, isto é, inserções e remoções, tivemos que, também, a preparar para tal efeito. Toda a implementação da nossa base de conhecimento foi realizada num ficheiro chamado *conhecimento.pl* [??].

```

:- module(database,[ utente/10, centrosaude/5, staff/4, vacinacao/7, medico/5, enfermeiro/5]). 
:- dynamic utente/10.
:- dynamic centrosaude/5.
:- dynamic staff/4.
:- dynamic vacinacao/7.
:- dynamic medico/5.
:- dynamic enfermeiro/5.
```

Figura 3.2: Declaração da base de conhecimento.

```

utente(1, 087462728, 'Jose Oliveira', 1965, 'joseoliv@gmail.com', 917263549, braga, engenheirx, [colestrol], 1).
utente(2, 528102846, 'Diogo Espírito Santo', 1956, 'diant@gmail.com', 936489367, coimbra, medicx, [], 7).
utente(3, 658746027, 'Afonso Castro', 1997, 'fonso@gmail.com', 927362918, braga, empresarix, [hipertensao], 2).
utente(4, 129837465, 'Duarte Catalao', 1986, 'dudu@gmail.com', 918374526, lisboa, arquitetx, [osteoporose], 5).
utente(5, 092183746, 'Lara Vilhena', 1996, 'laravilhena12@gmail.com', 9387362907, porto, modelo, [], 8).
```

Figura 3.3: Exemplos de povoamento na base de conhecimento relativamente aos utentes.

```

medico(1,'Ester Domingues',47,'F',1).
medico(2,'Manuel Castro',29,'M',2).
medico(3,'Emanuel Anjo',38,'M',3).
medico(4,'David Ferreira',43,'M',4).
medico(5,'Pedro Melo',40,'M',5).
```

Figura 3.4: Exemplos de povoamento na base de conhecimento relativamente aos médicos.

```

enfermeiro(1,'Márcia Araújo',27,'F',1).
enfermeiro(2,'Luzia Gomes',41,'F',2).
enfermeiro(3,'Mafalda Araújo',50,'F',3).
enfermeiro(4,'Henrique Megre',36,'M',4).
enfermeiro(5,'Daniela Sousa',53,'F',5).
```

Figura 3.5: Exemplos de povoamento na base de conhecimento relativamente aos enfermeiros.

```

staff(1, 1, 'Monica Sintra', 'monicas@gmail.com').
staff(2, 1, 'Cristiano Alves', 'cristianoalves@gmail.com').
staff(3, 1, 'Jorge Pires', 'jorgemanuel@gmail.com').
staff(4, 2, 'Antonio Goncalves', 'antoniogoncalves@gmail.com').
staff(5, 2, 'Marco Barbosa', 'marquitxi@gmail.com').
```

Figura 3.6: Exemplos de povoamento na base de conhecimento relativamente ao staff.

```

centrosaude(1, 'Centro de Saúde de Braga', 'Largo Paulo Orósio, 4700-031 Braga', 253928647, 'csbraga@gmail.com').
centrosaude(2, 'USF Bracara Augusta', 'Praça Gen. Humberto Delgado 47, 4715-213 Braga', 253964876, 'bracaraaugusta@gmail.com').
centrosaude(3, 'USF Sanus Caranda', 'R. André Soares 25, 4715-213 Braga', 253201530, 'usfcaranda@gmail.com'), 2.
centrosaude(4, 'Centro de Saúde de Guimarães', 'R. Francisco Santos Guimarães, 4810-225 Guimarães', 253519923, 'csguimaraes@gmail.com').
centrosaude(5, 'Centro de Saúde Amorosa', 'R. José Pinto Rodrigues 16, Guimarães, 4810-225 Guimarães', 253421340, 'csamorosa@gmail.com').
```

Figura 3.7: Exemplos de povoamento na base de conhecimento relativamente aos Centros de Saúde.

```

vacinacao(8,12,23,03,2021, pfizer, 1).
vacinacao(24,8,01,04,2021, pfizer, 2).
vacinacao(13,2,20,04,2021, pfizer, 1).
vacinacao(1,1,02,05,2021, astrazeneca, 1).
vacinacao(6,16,31,05,2021, pfizer, 1).
```

Figura 3.8: Exemplos do povoamento na base relativamente à toma das vacinas.

3.3.2 Programa

O nosso sistema foi todo implementado num programa descrito inteiramente no ficheiro *Covid.pl* [??]. Este programa utiliza como fonte de base de conhecimento o módulo anteriormente declarado.

Para tal, foi necessário informar o programa da utilização deste módulo, nunca esquecendo de declarar os operandos que serão necessários para a implementação, tal como a declaração de todos os predicados que contêm conhecimento subjetivo a alterações.

```
:— set_prolog_flag( discontiguous_warnings,off ).  
:— set_prolog_flag( single_var_warnings,off ).  
:— set_prolog_flag( unknown,fail ).  
:— set_prolog_flag( answer_write_options,[max_depth(0)] ).  
  
:— op( 900,xfy,'::' ).  
:— dynamic faseVacinacao/2.  
:— use_module(conhecimento).
```

Inicialmente, para organizar vários aspectos do trabalho tal como determinar os utentes já vacinados, optamos por construir o predicado *dataA* que nos indica a data atual através de predicados *built-in* do Prolog. Neste excerto, podemos também observar o predicado *solucoes* que nos permite verificar um conjunto de soluções existentes na nossa base de conhecimento.

```
dataA(Dia, Mes, Ano) :- get_time(TS),  
                      stamp_date_time(TS,DateTime,'local'),  
                      arg(3,DateTime,Dia),  
                      arg(2,DateTime,Mes),  
                      arg(1,DateTime,Ano).  
  
solucoes(X, XS, _) :- XS, assert(tmp(X)), fail.  
solucoes(_, _, R) :- solucoesAux([], R).  
  
solucoesAux(L, R) :- retract(tmp(X)), !, solucoesAux([X|L], R).  
solucoesAux(R, R).
```

Decidimos dividir os utentes de acordo com as suas características, nomeadamente a idade, profissão e doenças, sendo estes fatores de risco para quem contrair o vírus SARS-CoV-2. Desta forma, implementamos 4 predicados: um que indica uma lista de números de identificação de utentes com idade superior a 65 anos (*listaVelhosV*), um que nos fornece os números de identificação de utentes que apresentam doenças de risco (*listaDoentesRiscoV*) e dois que apresentam os números de identificação de utentes com as profissões de médico/a e enfermeiro/a (*listaMedicxV* e *listaEnfermeirxV*).

```
% Identifica utentes com uma ou mais Doenças; listaDoentesRiscoV(lista de IDs de utentes). -> {V,F}
listaDoentesRiscoV(IDs) :- findall(ID, (utente(ID,_,_,_,_,_,Doencas,_), length(Doencas, R), R > 0), IDs).

%-----
% Identifica utentes com mais de 65 anos de idade; listaVelhosV(lista de IDs de utentes). -> {V,F}
listaVelhosV(IDs) :- findall(ID, (utente(ID,_,_,Idade,_,_,_,_), dataA(_,_,A), A-Idade > 65), IDs).

%-----
% Identifica utentes que são médicos de profissão; listaMedicxV(lista de IDs de utentes). -> {V,F}
listaMedicxV(IDs) :- findall(ID, utente(ID,_,_,_,_,medicx,_,_), IDs).

%-----
% Identifica utentes que são enfermeiros de profissão; listaEnfermeirxV(lista de IDs de utentes). -> {V,F}
listaEnfermeirxV(IDs) :- findall(ID, utente(ID,_,_,_,_,enfermeirx,_,_), IDs).
```

Utilizando as funções acima descritas, construimos então o predicado *listafaseVacinacao1* que nos indica uma lista com os números de identificação dos utentes que estão selecionados para a primeira fase de vacinação, através da concatenação sucessiva de listas e com a eliminação dos números de identificação de utentes repetidos.

Os restantes utentes estão automaticamente selecionados para a segunda fase de vacinação, pelo que, tal como é possível observar pelo código do predicado *listafaseVacinacao2*, são todos os utentes cujo número de identificação não se encontra na lista fornecida dos que foram selecionados para a primeira fase.

```
% Identifica todos os eleitos para a primeira fase de vacinação; listafaseVacinacao1(lista de IDs de utentes). -> {V,F}
listafaseVacinacao1(IDs) :- listaEnfermeirxV(E),
    listaMedicxV(M),
    listaVelhosV(V),
    listaDoentesRiscoV(D),
    concat([], D, Z),
    concat(V, Z, W),
    concat(M, W, Y),
    concat(E, Y, X),
    repRemove(X, Xs),
    ordena(Xs, IDs).

%-----
% Identifica todos os eleitos para a segunda fase de vacinação; listafaseVacinacao2(lista de IDs de utentes). -> {V,F}
listafaseVacinacao2(IDs) :- findall(ID,(listafaseVacinacao1(Xs), utente(ID,_,_,_,_,_,_), nao(pertence(ID, Xs))), IDs).
```

Restrições à inserção e remoção de conhecimento

De forma a ser possível criar mecanismos de alteração de informação na base de conhecimento, foram criados predicados e invariantes que nos permitam manter a coerência na base de conhecimento.

Foram definidos dois predicados: *evolução* e *involução*.

```
%-----  
% Faz a inserção de conhecimento  
% Extensão predicado que permite a evolução conhecimento: Termo - {V,F}  
  
evolucao( Termo ) :- findall( Invariante, +Termo::Invariante, Lista ),  
                  insercao( Termo ),  
                  teste( Lista ).  
  
insercao( Termo ) :- assert( Termo ).  
insercao( Termo ) :- retract( Termo ), !, fail.  
  
teste( [] ).  
teste( [R|LR] ) :- R, teste( LR ).
```

Figura 3.9: Evolução de conhecimento

Foi implementado o mecanismo de *involução*, porém não foram definidos invariantes para este mecanismo, apesar de o termos definido na sua estrutura padrão que utilize esses tipos de invariantes. Achamos que para esta fase não seria relevante implementar mecanismos de coerência de remoção de conhecimento, sendo já possível remover qualquer conhecimento existente.

```
% Faz a remoção de conhecimento  
% Extensão predicado que permite a involução conhecimento: Termo -> {V,F}  
  
involucao( Termo ) :- solucoes( Invariante, -Termo::Invariante, Lista ),  
                   remocao( Termo ),  
                   teste( Lista ).  
  
remocao( Termo ) :- retract( Termo ).  
remocao( Termo ) :- assert( Termo ), !, fail.
```

Figura 3.10: Involução de conhecimento.

Deste modo, foi necessário implementar diversos invariantes para a inserção de conhecimento sobre as entidades mais relevantes do caso de estudo.

Os utentes são, sem dúvida, uma das entidades mais relevantes no sistema. Deste modo, foram criados mecanismos de modo a que nos permitisse permitam ter um utente com um número únicos de identificação, um número único de segurança social, um email único, um número de telefone único e que esteja afiliado em apenas um centro de saúde.

```
% Todos os utentes tem ID's distintos;
+utente(Id,_,_,_,_,_,_,_,_) :: (findall( Id, utente(Id,_,_,_,_,_,_,_), R),
length(R, X), X==1).

%-----
% Todos os utentes tem números da SS distintos;
+utente(_,Ss,_,_,_,_,_,_) :: (findall( Ss, utente(_,Ss,_,_,_,_,_), R),
length(R, X), X==1).

%-----
% Todos os utentes tem emails distintos;
+utente(_,_,_,E,_,_,_,_) :: (findall( E, utente(_,_,_,E,_,_,_), R),
length(R, X), X==1).

%-----
% Todos os utentes tem números de telefone distintos;
+utente(_,_,_,_,T,_,_,_,_) :: (findall( T, utente(_,_,_,T,_,_,_), R),
length(R, X), X==1).

% Todos os utentes estão destacados num centro de saúde existente;
+utente(_,_,_,_,_,Cs) :: (findall( Cs, centrosaude(Cs,_,_,_), R),
length(R, X), X==1).
```

Figura 3.11: Invariantes dos Utentes

Do mesmo modo, os centros de saúde e o staff necessitam de ser registados com informações únicas, de modo a não permitir a existência de conhecimento incoerente.

```
% Todos os centros de saude tem ID's distintos;
+centrosaude(Id,_,_,_,_) :: (findall( Id, centrosaude(Id,_,_,_,_), R),
                                length(R, X), X==1).

%----- %
% Todos os centros de saude tem moradas distintas;
+centrosaude(_,_,M,_,_) :: (findall( M, centrosaude(_,_,M,_,_), R),
                                length(R, X), X==1).

%----- %
% Todos os centros de saude tem numeros de telefone distintos;
+centrosaude(_,_,_,T,_) :: (findall( T, centrosaude(_,_,_,T,_), R),
                                length(R, X), X==1).

%----- %
% Todos os centros de saude tem emails distintos;
+centrosaude(_,_,_,_,E) :: (findall( E, centrosaude(_,_,_,_,E), R),
                                length(R, X), X==1).
```

Figura 3.12: Invariantes dos centros de saúde

```
% Todos os membros do staff tem ID's distintos;
+staff(Id,_,_,_) :: (findall( Id, staff(Id,_,_,_), R),
                        length(R, X), X==1).

%----- %
% Todos os membros do staff tem de estar destacados num centro de saude existente distintos;
+staff(_,Cs,_,_) :: (findall( Cs, centrosaude(Cs,_,_,_,_), R),
                        length(R, X), X==1).

%----- %
% Todos os membros do staff tem emails distintos;
+staff(_,_,_,M) :: (findall( M, staff(_,_,_,M), R),
                        length(R, X), X==1).
```

Figura 3.13: Invariantes do Staff

Já no caso das vacinações, tratam-se de situações muito pontuais que necessitaram da nossa atenção. Mais concretamente, as situações tratadas, para cada registo de vacinação, foram as seguintes:

- **Apenas um membro de staff associado;**
- **Apenas um utente associado;**
- **Um utente associado por uma só toma;**
- **Um utente associado por um só tipo de vacina:** existindo mais que uma toma das vacinas e diferentes tipos de vacinas, este invariante assegura-nos que não se toma vacinas de diferentes tipos.
- **O utente e o staff que administrou a vacina têm de ter o registo da administração no mesmo centro de saúde:** este invariante foi implementado de forma a não existir incoerência entre os locais da administração da vacinação.

- **A segunda toma da vacina terá de ser posterior à primeira toma:** este invariante assegura-nos o caso de que uma segunda toma de uma certa vacina seja administrada depois de uma primeira dose. Porém, não nos assegura que não exista nenhum registo de primeira dose. Isto implica que seja possível administrar segundas doses antes de uma primeira dose. Esta situação achamos que não seria relevante considerar, uma vez que foi considerado que uma primeira dose seja igual à segunda, sendo mais flexível a administrar as duas doses num utente, para o caso de não ser possível vacinar a primeira dose num respetivo dia.

```
% Todos as vacinas tem um membro do staff existente;
+vacinacao(S,_,_,_,_,_) :: (findall( S, staff(S,_,_,_), R),
                             length(R, X), X==1).

+vacinacao(_,U,_,_,_,_) :: (findall( U, utente(U,_,_,_,_,_,_,_), R),
                             length(R, X), X==1).

% Todas as vacinas tem um utente por toma, por vacina;
+vacinacao(_,U,_,_,_,Toma) :: (findall((U, Toma), vacinacao(_,U,_,_,_,Toma), R),
                                length(R, X), X==1).

+vacinacao(_,U,_,_,_,Tipo,Toma) :: (Toma == 1;(findall((U, Tipo), vacinacao(_,U,_,_,_,Tipo,_), R),
                                         length(R, X), X==2)).

% Staff e Utente mesmo centro de saude.
+vacinacao(S,U,_,_,_,_) :: (findall((U, S, Cs), (utente(U,_,_,_,_,_,_,Cs), staff(S,Cs,_,_)), R),
                             length(R, X), X==1).

% Segunda toma ser depois da primeira.
+vacinacao(_,Utente,Dia2,Mes2,Ano2,_,Toma) :: 
    (Toma == 1;(vacinacao(_,Utente,Dia1,Mes1,Ano1,_,1),
                Toma == 2,comparaDatas(Dia1,Mes1,Ano1,Dia2,Mes2,Ano2))).
```

Figura 3.14: Invariantes das vacinações

Quanto ao desenvolvimento de predicados no nosso sistema, apresentamos, primeiramente, o predicado ***peepsVac***. Este predicado foi implementado de forma a encontrar todos os utentes que que tenham sido vacinados pelo menos uma vez. Para tal, foram utilizados outros predicados implementados no nosso sistema de forma a apresentar o resultado de forma ordenada e sem repetidos.

```
% Identificar IDs de pessoas vacinadas pelo menos uma vez; peepsVac(Lista de IDs de utentes). -> {V,F}
peepsVac(R) :- peepsVac1Time(V1),
              peepsVac2Time(V2),
              concat(V1, V2, X),
              repRemove(X, Y),
              ordena(Y, R).
```

Figura 3.15: Predicado **peepsVac**.

Uma vez implementado o predicado anterior e de forma a encontrar os utentes não vacinados, bastou-nos utilizar o predicado anterior e procurar todos os utentes que não pertençam a essa lista de vacinados, originando o predicado ***peepsNoVac***.

```
% Identificar IDs de pessoas não vacinadas; peepsNoVac(Lista de IDs de utentes). -> {V,F}
peepsNoVac(R) :- findall(ID,
                           (utente(ID, _, _, _, _, _, _, _),
                            peepsVac(V),
                            nao(pertence(ID, V))),
                           X),
               repRemove(X, B),
               ordena(B, R).
```

Figura 3.16: Predicado **peepsNoVac**.

O predicado ***peepsVac1Futura*** indica as todos os números de identificação de utentes que têm prevista a primeira toma da vacina numa lista.

```
% Identificar IDs de pessoas que tem a primeira toma prevista; peepsVac1Futura(lista de IDs de utentes). -> {V,F}
peepsVac1Futura(R) :- findall(ID,
                               (vacinacao(_, ID, D, M, A, _, 1),
                                utente(ID, _, _, _, _, _, _),
                                nao(jaPassou(D, M, A))),
                               X),
                               repRemove(X, B),
                               ordena(B, R).
```

Figura 3.17: Predicado **peepsVac1Futura**.

No excerto abaixo apresentado, temos o predicado ***peepsVac1Time*** que nos permite saber o número de identificação de utentes que já estão vacinado com a primeira dose da vacina, na data atual.

```
% Identificar IDs de pessoas vacinadas com a 1a dose; peepsVac1Time(Lista de IDs de utentes). -> {V,F}
peepsVac1Time(R) :- findall(ID,
    (vacinacao(_,ID,D,M,A, _, 1),
     utente(ID,_,_,_,_,_,_,_),
     jaPassou(D, M, A)),
    X),
    repRemove(X,B),
    ordena(B, R).
```

Figura 3.18: Predicado ***peepsVac1Time***.

O predicado abaixo (***peepsVac2Time***), tal como o *peepsVac1Time*, apresenta uma lista dos números de identificação dos utentes que têm já sido vacinados com a segunda dose da vacina. Este predicado pode não aparentar ser muito relevante, uma vez que não se pode inserir na base de conhecimento a segunda toma sem ser feita a primeira, no entanto é utilizado como auxiliar a funções mais elaboradas que veremos mais abaixo.

```
% Identificar IDs de pessoas vacinadas com 2a dose; peepsVac2Time(Lista de IDs de utentes). -> {V,F}
peepsVac2Time(R) :- findall(ID,
    (vacinacao(_,ID,D,M,A, _, 2),
     utente(ID,_,_,_,_,_,_),
     jaPassou(D, M, A)),
    X),
    repRemove(X,B),
    ordena(B, R).
```

Figura 3.19: Predicado ***peepsVac2Time***.

De forma a saber os utentes que ainda não têm nenhuma das vacinações prevista, construimos o predicado ***peepsNoVacFutura***.

```
% Identificar IDs de pessoas não vacinadas e não tem toma prevista; peepsNoVacFutura(lista de IDs de utentes). -> {V,F}
peepsNoVacFutura(R) :- findall(ID,
    (utente(ID,_,_,_,_,_,_),
     peepsNoVac(X),
     pertence(ID,X),
     peepsVac1Futura(Y),
     nao(pertence(ID,Y)),
     peepsVac2Futura(Z),
     nao(pertence(ID,Z))),
    A),
    repRemove(A,B),
    ordena(B, R).
```

Figura 3.20: Predicado ***peepsNoVacFutura***.

Para indentificar todos os IDs das pessoas que foram vacinadas indevidamente criamos um predicado **indevidamente**. Isto é , dá o números de IDs que já foram vacinados antes de toda a gente da primeira fase ter tomado as duas doses.

```
% Identificar IDs de pessoas vacinadas indevidamente; indevidamente(lista de IDs de utentes). -> {V,F}
indevidamente(R) :- findall(ID,
                           (vacinacao(_, ID, D, M, A, _, _),
                            utente(ID, _, _, _, _, _, _, _),
                            jaPassou(D, M, A),
                            listaFaseVacinacao1(F1),
                            nao(pertence(ID, F1)),
                            nao(verifica2toma(F1))),
                           Y),
                           repRemove(Y, X),
                           ordena(X, R).
```

Figura 3.21: Predicado **indevidamente**.

De modo a observar os utentes selecionados para a primeira fase de vacinação que não tomaram a segunda dose da vacina de forma mais simples, edificamos um predicado que lista todos os números de identificação destes utentes.

```
% Identificar IDs de pessoas não vacinadas nenhuma vez e candidatas à primeira fase; candidatosFase1(lista de IDs de utentes). -> {V,F}
candidatosFase1(R) :- findall(ID,
                               (utente(ID, _, _, _, _, _, _, _),
                                listaFaseVacinacao1(F1),
                                pertence(ID, F1),
                                peepsVac(V),
                                nao(pertence(ID, V))),
                               X),
                               repRemove(X, B),
                               ordena(B, R).
```

Figura 3.22: Predicado **candidatosFase1**.

Com intuito de sabermos todos os IDs de pessoas que ainda terão de tomar a segunda dose da vacina, criamos o predicado **falta2toma**.

```
% Identificar IDs de pessoas que falta a segunda toma da vacina; falta2toma(lista de IDs de utentes). -> {V,F}
falta2toma(R) :- findall(ID,
                          (utente(ID, _, _, _, _, _, _, _),
                           peepsVac1Time(V1),
                           pertence(ID, V1),
                           peepsVac2Time(V2),
                           nao(pertence(ID, V2))),
                          X),
                          repRemove(X, B),
                          ordena(B, R).
```

Figura 3.23: Predicado **falta2toma**.

Criamos o predicado ***calculaDosesFuturas*** com o intuito de nos dar uma estimativa das doses futuras para cada uma das fases.

```
% estimativa de doses futuras para cada uma das fases
calculaDosesFuturas(Res1,Res2) :- peepsVac1Futura(R1), peepsVac2Futura(R2), length(R1,X1), length(R2,X2), Res1 is X1, Res2 is X2.
```

Figura 3.24: Predicado ***calculaDosesFuturas***.

Criamos um predicado ***estimativaPorVacinas*** que, dando um tipo de vacina, sabemos o número previsto de vacinações a realizar.

```
% estimativa de doses por vacina dada
estimativaPorVacinas(Tipo, Res) :- findall(Tipo, (vacinacao(_,_,D,M,A,Tipo,_), nao(jaPassou(D,M,A))), R), length(R, X), Res is X.
```

Figura 3.25: Predicado ***estimativaPorVacina***.

De forma a verificar todas os utentes totalmente vacinados, isto é, tomaram as duas doses de uma vacina, foi implementado o predicado ***fullVac***, que tira partido de predicados implementados neste projeto.

```
% Identificar IDs de pessoas vacinadas com as duas doses; fullVac(Lista de IDs de utentes). -> {V,F}
fullVac(R) :- findall(ID,
    (utente(ID,_,_,_,_,_,_,_,_,_),
     peepsVac1Time(V1),
     pertence(ID,V1),
     peepsVac2Time(V2),
     pertence(ID,V2)),
    X),
    repRemove(X,B),
    ordena(B, R).
```

Figura 3.26: Predicado ***fullVac***.

Achamos, também, pertinente que seria necessário verificar todos os utentes sem previsão de vacinação futura. Assim, desenvolvemos o predicado ***peepsNoVacFutura*** que utiliza, de forma intuitiva, diversos outros predicados implementados no nosso sistema.

```
% Identificar IDs de pessoas não vacinadas e não tem toma prevista; peepsNoVacFutura(lista de IDs de utentes). -> {V,F}
peepsNoVacFutura(R) :- findall(ID,
    (utente(ID,_,_,_,_,_,_,_,_),
     peepsNoVac(X),
     pertence(ID,X),
     peepsVac1Futura(Y),
     nao(pertence(ID,Y)),
     peepsVac2Futura(Z),
     nao(pertence(ID,Z))),
    A),
    repRemove(A,B),
    ordena(B, R).
```

Figura 3.27: Predicado ***peepsNoVacFutura***.

Com o intuito de verificar todos todos os utentes de uma lista dos quais já tomaram a segunda toma da vacinação, criamos o predicado denominado ***verifica2toma***.

```
% Verifica se todos os utentes de uma lista ja tomaram a 2a toma da vacina; verifica2toma(Lista de IDs de utentes). -> {V,F}
verifica2toma([]).
verifica2toma([H|T]) :- veraux(H), verifica2toma(T).
```

Figura 3.28: Predicado ***verifica2toma***.

Foi criado o predicado ***veraux*** para verificar se um utente já completou o processo de vacinação, ou seja, tomou as duas doses da vacina. Este predicado também facilita a execução do predicado explicado anteriormente.

```
% Verifica se um utente ja tomou as duas doses da vacina; veraux(ID do utente). -> {V,F}
veraux(ID) :- utente(ID,_,_,_,_,_,_,_),
            vacinacao(_,ID,D1,M1,A1,_,1),
            jaPassou(D1, M1, A1),
            vacinacao(_,ID,D2,M2,A2,_,2),
            jaPassou(D2, M2, A2).
```

Figura 3.29: Predicado auxiliar, ***veraux***, ao predicado ***verifica2toma***.

Por fim, achamos pertinente demonstrar neste relatório a nossa implementação de dois meta-predicados: si (para retratar um sistema de inferência) e nao (para negar um predicado). Este sistema de inferência permite-nos identificar a veracidade de conhecimento na nossa base de dados. Identificamos já o conhecimento desconhecido, apesar de este não ter sido explorado neste trabalho prático. Todo este conhecimento é aquele que não seja verdadeiro e que também não seja falso, pelo que foi necessário definir o predicado de negação.

```
% sistema de inferência que permite reconhecer se é V/F ou desconhecido
si(Questao,verdadeiro) :- Questao.
si(Questao,falso) :- -Questao.
si(Questao,desconhecido) :- nao(Questao), nao(-Questao).
```

Figura 3.30: Meta-predicado que define um sistema de inferência.

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

Figura 3.31: Meta-predicado que define a negação.

Capítulo 4

Resolução das Funcionalidades Propostas

4.1 Representação do conhecimento positivo

Todo o conhecimento positivo foi representado na Fase 1, onde foi explicado o significado dos factos que compõem a base de conhecimento. Posto isto, a representação do conhecimento negativo sera feita nesta fase.

```
%Utente: #Idutente, Nº Segurança_Social, Nome, Data_Nasc, Email, Telefone, Morada, Profissão, [Doenças_Crónicas], #CentroSaúde -> {V,F}

%Base de conhecimento do Utente
utente(1, 087462728, 'Jose Oliveira', 1965, 'joseoliv@gmail.com', 917263549, braga, engenheirx, [colestrol],1).
utente(2, 528102846, 'Diogo Espírito Santo', 1956, 'diant@gmail.com', 936489367, coimbra, medicx, [],7).
utente(3, 658746027, 'Afonso Castro', 1997, 'fonso@gmail.com', 927362918, braga, empresarix, [hipertensao],2).
utente(4, 129837465, 'Duarte Catalao', 1986, 'dudu@gmail.com', 918374526, lisboa, arquitetx, [osteoporose],5).
```

Figura 4.1: Inserção de conhecimento relativo a Utentes.

```
%centro_saúde: #Idcentro, Nome, Morada, Telefone, Email -> {V,F}

%Base de conhecimento dos Centros de Saúde
centrosaude(1, 'Centro de Saúde de Braga', 'Largo Paulo Orósio, 4700-031 Braga', 253928647, 'csbraga@gmail.com').
centrosaude(2, 'USF Bracara Augusta', 'Praça Gen. Humberto Delgado 47, 4715-213 Braga', 253964876, 'bracaraaugusta@gmail.com').
centrosaude(3, 'USF Sanus Caranda', 'R. André Soares 25, 4715-213 Braga', 253201530, 'usfcaranda@gmail.com').
centrosaude(4, 'Centro de Saúde de Guimarães', 'R. Francisco Santos Guimarães, 4810-225 Guimarães', 253519923, 'csguimaraes@gmail.com').
```

Figura 4.2: Inserção de conhecimento relativo a Centros de Saúde.

```
%staff: #Cstaff, #Idcentro, Nome, email -> {V,F}

%Base de conhecimento do staff
staff(1, 1, 'Monica Sintra', 'monicas@gmail.com').
staff(2, 1, 'Cristiano Alves', 'cristianoalves@gmail.com').
staff(3, 1, 'Jorge Pires', 'jorgemanuel@gmail.com').
staff(4, 2, 'Antonio Goncalves', 'antoniongoncalves@gmail.com').
```

Figura 4.3: Inserção de conhecimento relativo a Staffs.

```
%vacinação_Covid: #Staf, #utente, Dia, Mes, Ano, Vacina, Toma -> {V,F}

%Base de conhecimento da vacinação
vacinacao(1,1,02,05,2021, astraZeneca, 1).
vacinacao(13,2,20,04,2021, pfizer, 1).
vacinacao(4,3,31,09,2021, pfizer, 1).
vacinacao(20,4,18,02,2021, pfizer, 2).
```

Figura 4.4: Inserção de conhecimento relativo a Vacinas.

4.2 Representação do conhecimento imperfeito

Nesta fase do projeto, foi necessário adicionar conhecimento imperfeito, que diz respeito às situações em que não é possível obter resposta às questões, ou seja, o resultado é desconhecido. Existe três tipos deste conhecimento, nomeadamente o incerto, o impreciso e o interdito, que serão explicados detalhadamente de seguida.

4.2.1 Conhecimento Incerto:

Este tipo de conhecimento está associado às situações em que um determinado valor é desconhecido, ou seja, não se conhece o seu valor, no entanto existe a possibilidade de o vir a descobrir. Para tal, foi necessário criar exceções para cada uma das situações em que determinado valor é desconhecido, como pode ser observado de seguida a título de exemplo:

```
% -----  
% ----- Conhecimento Incerto -----  
% -----  
  
excecao(utente(Id,Ss,N,Dt,E,Tlf,M,P,DC,Cs)) :- utente(Id,desconhecido,N,Dt,E,Tlf,M,P,DC,Cs).  
excecao(utente(Id,Ss,N,Dt,E,Tlf,M,P,DC,Cs)) :- utente(Id,Ss,desconhecido,Dt,E,Tlf,M,P,DC,Cs).  
excecao(utente(Id,Ss,N,Dt,E,Tlf,M,P,DC,Cs)) :- utente(Id,Ss,N,desconhecido,E,Tlf,M,P,DC,Cs).  
excecao(utente(Id,Ss,N,Dt,E,Tlf,M,P,DC,Cs)) :- utente(Id,Ss,N,Dt,desconhecido,Tlf,M,P,DC,Cs).  
excecao(utente(Id,Ss,N,Dt,E,Tlf,M,P,DC,Cs)) :- utente(Id,Ss,N,Dt,E,desconhecido,M,P,DC,Cs).  
excecao(utente(Id,Ss,N,Dt,E,Tlf,M,P,DC,Cs)) :- utente(Id,Ss,N,Dt,E,Tlf,desconhecido,P,DC,Cs).  
excecao(utente(Id,Ss,N,Dt,E,Tlf,M,P,DC,Cs)) :- utente(Id,Ss,N,Dt,E,Tlf,M,desconhecido,DC,Cs).  
excecao(utente(Id,Ss,N,Dt,E,Tlf,M,P,DC,Cs)) :- utente(Id,Ss,N,Dt,E,Tlf,M,P,desconhecido,Cs).  
excecao(utente(Id,Ss,N,Dt,E,Tlf,M,P,DC,Cs)) :- utente(Id,Ss,N,Dt,E,Tlf,M,P,DC,desconhecido).  
  
excecao(centrosaude(Id,N,M,Tlf,E)) :- centrosaude(Id,desconhecido,M,Tlf,E).  
excecao(centrosaude(Id,N,M,Tlf,E)) :- centrosaude(Id,N,desconhecido,Tlf,E).  
excecao(centrosaude(Id,N,M,Tlf,E)) :- centrosaude(Id,N,M,desconhecido,E).  
excecao(centrosaude(Id,N,M,Tlf,E)) :- centrosaude(Id,N,M,Tlf,desconhecido).  
  
excecao(staff(Id,Cs,N,E)) :- staff(Id,desconhecido,N,E).  
excecao(staff(Id,Cs,N,E)) :- staff(Id,Cs,desconhecido,E).  
excecao(staff(Id,Cs,N,E)) :- staff(Id,Cs,N,desconhecido).  
  
excecao(enfermeiro(Id,N,I,G,Cs)) :- enfermeiro(Id,desconhecido,I,G,Cs).  
excecao(enfermeiro(Id,N,I,G,Cs)) :- enfermeiro(Id,N,desconhecido,G,Cs).  
excecao(enfermeiro(Id,N,I,G,Cs)) :- enfermeiro(Id,N,I,desconhecido,Cs).  
excecao(enfermeiro(Id,N,I,G,Cs)) :- enfermeiro(Id,N,I,G,desconhecido).  
  
excecao(medico(Id,N,I,G,Cs)) :- medico(Id,desconhecido,I,G,Cs).  
excecao(medico(Id,N,I,G,Cs)) :- medico(Id,N,desconhecido,G,Cs).  
excecao(medico(Id,N,I,G,Cs)) :- medico(Id,N,I,desconhecido,Cs).  
excecao(medico(Id,N,I,G,Cs)) :- medico(Id,N,I,G,desconhecido).  
  
excecao(vacinacao(A,B,C,D,E,F,G)) :- vacinacao(desconhecido,B,C,D,E,F,G).  
excecao(vacinacao(A,B,C,D,E,F,G)) :- vacinacao(A,desconhecido,C,D,E,F,G).  
excecao(vacinacao(A,B,C,D,E,F,G)) :- vacinacao(A,B,desconhecido,D,E,F,G).  
excecao(vacinacao(A,B,C,D,E,F,G)) :- vacinacao(A,B,C,desconhecido,E,F,G).  
excecao(vacinacao(A,B,C,D,E,F,G)) :- vacinacao(A,B,C,D,desconhecido,F,G).  
excecao(vacinacao(A,B,C,D,E,F,G)) :- vacinacao(A,B,C,D,E,desconhecido,G).  
excecao(vacinacao(A,B,C,D,E,F,G)) :- vacinacao(A,B,C,D,E,F,desconhecido).
```

Figura 4.5: Representação do conhecimento Incerto para utente,centrosaude,staff,enfermeiro,medico e vacina

Com isto, observando a figura acima, por exemplo no caso do utente, com Número de Segurança Social "**desconhecido**" apesar de existir na base de conhecimento, este não a tem na mesma.

4.2.2 Conhecimento Impreciso:

O conhecimento impreciso verifica-se quando determinado facto é desconhecido, no entanto a dúvida incide sobre um determinado conjunto de valores bem determinado. Assim, de seguida são apresentados alguns exemplos deste tipo de conhecimento:

```
% ----- Conhecimento Impreciso -----
%
% Dúvida sobre o nome do utente e morada cujo identificador é 23.
excecao(utente(23, 'Maria Marques', 2000, 'marymar@gmail.com', 939273829, porto, estudante, [],16)).
excecao(utente(23, 'Maria Marques', 2000, 'marymar@gmail.com', 939273829, braga, estudante, [],16)).
excecao(utente(23, 'Mario Marques', 2000, 'marymar@gmail.com', 939273829, porto, estudante, [],16)).
excecao(utente(23, 'Mario Marques', 2000, 'marymar@gmail.com', 939273829, braga, estudante, [],16)).
impreciso(23, utente).

% Dúvida sobre a profissão do utente, cujo identificador é 24.
excecao(utente(24, 'Maria Abelha', 1990, 'mariabel@gmail.com', 123566778, porto, estudante, [],20)).
excecao(utente(24, 'Maria Abelha', 1990, 'mariabel@gmail.com', 123566778, braga, revendedora, [],20)).
impreciso(24, utente).

% Dúvida sobre o telefone associado a um centro de saúde cujo identificador é 21.
excecao(centrosaude(21, 'Centro de Saúde de Ruães', 'R. de Ruães, 4700-565 Mire de Tibães', 253602490, 'csruaes@gmail.com')).
excecao(centrosaude(21, 'Centro de Saúde de Ruães', 'R. de Ruães, 4700-565 Mire de Tibães', 253602400, 'csruaes@gmail.com')).
impreciso(21, centrosaude).

% Dúvida sobre o nome do staff cujo identificador é 37.
excecao(staff(37, 25, 'Tiago Borges', 'tiagoborges@gmail.com')).
excecao(staff(37, 25, 'Tiago Rodrigues', 'tiagoborges@gmail.com')).
impreciso(37, staff).

% Dúvida sobre a idade do enfermeiro cujo identificador é 21.
excecao(enfermeiro(21, 'Sousa Tavares', 45, 'F', 20)).
excecao(enfermeiro(21, 'Sousa Tavares', 39, 'F', 20)).
impreciso(21, enfermeiro).

% Dúvida sobre o centro de saúde e a idade do medico cujo identificador é 21.
excecao(medico(21, 'Borges Carvalho', 41, 'F', 19)).
excecao(medico(21, 'Borges Carvalho', 41, 'F', 20)).
excecao(medico(21, 'Borges Carvalho', 45, 'F', 19)).
excecao(medico(21, 'Borges Carvalho', 45, 'F', 20)).
impreciso(21, medico).

% Dúvida sobre o tipo da vacinação.
excecao(vacinação(25, 13, 29, 06, 2021, astraZeneca, 2)).
imprecisoVacinacao(25, 13, 29, 06, 2021, vacinação).

% Dúvida sobre a toma da vacinação.
excecao(vacinação(25, 13, 29, 06, 2021, pfizer, 1)).
imprecisoVacinacao(25, 13, 29, 06, 2021, vacinação).
```

Figura 4.6: Representação do conhecimento Impreciso para utente,centrosaude,staff,enfermeiro,medico e vacinacao

Com isto, observando a figura acima, não se sabe se a morada do Manuel é Braga ou Povoa de Lanhoso, mas sabe-se que é uma destas.

4.2.3 Conhecimento Interdito:

Este tipo de conhecimento ocorre quando determinado facto não se conhece e nunca se saberá o seu valor. Assim, foi necessário utilizar o predicado nulo para que este conhecimento desconhecido não possa ser evoluído. Deste modo, de seguida serão apresentados alguns exemplos:

```
% -----
% ----- Conhecimento Interdito -----
%
excecao(utente(Id,Ss,N,Dt,E,Tlf,M,P,DC,Cs)) :- utente(Id,semSegurancaSocial,N,Dt,E,Tlf,M,P,DC,Cs).
excecao(utente(Id,Ss,N,Dt,E,Tlf,M,P,DC,Cs)) :- utente(Id,Ss,semNome,Dt,E,Tlf,M,P,DC,Cs).
excecao(utente(Id,Ss,N,Dt,E,Tlf,M,P,DC,Cs)) :- utente(Id,Ss,N,semIdade,E,Tlf,M,P,DC,Cs).
excecao(utente(Id,Ss,N,Dt,E,Tlf,M,P,DC,Cs)) :- utente(Id,Ss,N,Dt,semEmail,Tlf,M,P,DC,Cs).
excecao(utente(Id,Ss,N,Dt,E,Tlf,M,P,DC,Cs)) :- utente(Id,Ss,N,Dt,E,semTelefone,M,P,DC,Cs).
excecao(utente(Id,Ss,N,Dt,E,Tlf,M,P,DC,Cs)) :- utente(Id,Ss,N,Dt,E,Tlf,semMorada,P,DC,Cs).
excecao(utente(Id,Ss,N,Dt,E,Tlf,M,P,DC,Cs)) :- utente(Id,Ss,N,Dt,E,Tlf,M,semProfissao,DC,Cs).
excecao(utente(Id,Ss,N,Dt,E,Tlf,M,P,DC,Cs)) :- utente(Id,Ss,N,Dt,E,Tlf,M,P,semDoencasCronicas,Cs).
excecao(utente(Id,Ss,N,Dt,E,Tlf,M,P,DC,Cs)) :- utente(Id,Ss,N,Dt,E,Tlf,M,P,DC,semCentroSaude).

excecao(centrosaude(Id,N,M,Tlf,E)) :- centrosaude(Id,semNome,M,Tlf,E).
excecao(centrosaude(Id,N,M,Tlf,E)) :- centrosaude(Id,N,semMorada,Tlf,E).
excecao(centrosaude(Id,N,M,Tlf,E)) :- centrosaude(Id,N,M,semTelefone,E).
excecao(centrosaude(Id,N,M,Tlf,E)) :- centrosaude(Id,N,M,Tlf,semEmail).

excecao(staff(Id,Cs,N,E)) :- staff(Id,semCentroSaude,N,E).
excecao(staff(Id,Cs,N,E)) :- staff(Id,Cs,semNome,E).
excecao(staff(Id,Cs,N,E)) :- staff(Id,Cs,N,semEmail).

excecao(enfermeiro(Id,N,I,G,Cs)) :- enfermeiro(Id,semNome,I,G,Cs).
excecao(enfermeiro(Id,N,I,G,Cs)) :- enfermeiro(Id,N,semIdade,G,Cs).
excecao(enfermeiro(Id,N,I,G,Cs)) :- enfermeiro(Id,N,I,semGenero,Cs).
excecao(enfermeiro(Id,N,I,G,Cs)) :- enfermeiro(Id,N,I,G,semCentroSaude).

excecao(medico(Id,N,I,G,Cs)) :- medico(Id,semNome,I,G,Cs).
excecao(medico(Id,N,I,G,Cs)) :- medico(Id,N,semIdade,G,Cs).
excecao(medico(Id,N,I,G,Cs)) :- medico(Id,N,I,semGenero,Cs).
excecao(medico(Id,N,I,G,Cs)) :- medico(Id,N,I,G,semCentroSaude).

excecao(vacinacao(A,B,C,D,E,F,G)) :- vacinacao(semStaff,B,C,D,E,F,G).
excecao(vacinacao(A,B,C,D,E,F,G)) :- vacinacao(A,semUtente,C,D,E,F,G).
excecao(vacinacao(A,B,C,D,E,F,G)) :- vacinacao(A,B,semDia,D,E,F,G).
excecao(vacinacao(A,B,C,D,E,F,G)) :- vacinacao(A,B,C,semMes,E,F,G).
excecao(vacinacao(A,B,C,D,E,F,G)) :- vacinacao(A,B,C,D,semAno,F,G).
excecao(vacinacao(A,B,C,D,E,F,G)) :- vacinacao(A,B,C,D,E,semTipoVacina,G).
excecao(vacinacao(A,B,C,D,E,F,G)) :- vacinacao(A,B,C,D,E,F,semToma).
```

Figura 4.7: Representação do conhecimento Interdito.

```
nulo(semIdade).
nulo(semMorada).
nulo(semAno).
nulo(semMes).
nulo(semDia).
nulo(semSegurancaSocial).
nulo(semNome).
nulo(semTelefone).
nulo(semEmail).
nulo(semProfissao).
nulo(semDoencasCronicas).
nulo(semCentroSaude).
nulo(semGenero).
nulo(semStaff).
nulo(semUtente).
nulo(semTipoVacina).
nulo(semToma).
```

Figura 4.8: Representação do conhecimento Interdito - Valores Nulos.

4.3 Manipular invariantes que designem restrições à inserção e à remoção de conhecimento do sistema

É necessário implementar diversos invariantes para que a nossa base de conhecimento funcione corretamente, sendo estes responsáveis por restringir a inserção e remoção de conhecimento da mesma.

Com a utilização de alguns predicados como auxílio, é possível com que base de conhecimento não evolua de forma errada.

Neste exercício é agora possível a inserção de conhecimento positivo e negativo bem como incerto, impreciso e interdito.

Para que a nossa base de conhecimento seja coerente, foram criados invariantes de modo a não ser possível inserir conhecimento repetido:

Utente

- Negativo
- Incerto
- Interdito
- Impreciso

```
% Permite evolução de utentes com diferentes dados desconhecidos de interdito
evolucao( utente(A,B,C,D,E,F,G,H,I,J), Tipo, Desconhecido ) :-%
  Tipo == interdito,
  (
    Desconhecido == segurança_social; Desconhecido == nome; Desconhecido == dataNascimento;
    Desconhecido == email; Desconhecido == telefone; Desconhecido == morada;
    Desconhecido == profissao; Desconhecido == doençasCronicas; Desconhecido == centroSaude
  ),
  (
    findall( Invariante, +utente(A,semSegurançaSocial,C,D,E,F,G,H,I,J)::Invariante, Lista), teste(Lista), insercao(utente(A,semSegurançaSocial,C,D,E,F,G,H,I,J));
    findall( Invariante, +utente(A,B,semNome,D,E,F,G,H,I,J)::Invariante, Lista), teste(Lista), insercao(utente(A,B,semNome,D,E,F,G,H,I,J));
    findall( Invariante, +utente(A,B,C,semIdade,E,F,G,H,I,J)::Invariante, Lista), teste(Lista), insercao(utente(A,B,C,semIdade,E,F,G,H,I,J));
    findall( Invariante, +utente(A,B,C,D,semEmail,F,G,H,I,J)::Invariante, Lista), teste(Lista), insercao(utente(A,B,C,D,semEmail,F,G,H,I,J));
    findall( Invariante, +utente(A,B,C,D,E,semTelefone,G,H,I,J)::Invariante, Lista), teste(Lista), insercao(utente(A,B,C,D,E,semTelefone,G,H,I,J));
    findall( Invariante, +utente(A,B,C,D,E,F,semMorada,H,I,J)::Invariante, Lista), teste(Lista), insercao(utente(A,B,C,D,E,F,semMorada,H,I,J));
    findall( Invariante, +utente(A,B,C,D,E,F,semProfissao,I,J)::Invariante, Lista), teste(Lista), insercao(utente(A,B,C,D,E,F,semProfissao,I,J));
    findall( Invariante, +utente(A,B,C,D,E,F,G,H,semDoençasCronicas,J)::Invariante, Lista), teste(Lista), insercao(utente(A,B,C,D,E,F,G,H,semDoençasCronicas,J));
    findall( Invariante, +utente(A,B,C,D,E,F,G,H,semCentroSaude)::Invariante, Lista), teste(Lista), insercao(utente(A,B,C,D,E,F,G,H,semCentroSaude))
  ),
  insercao( interdito(A,utente)).

```

```
% Permite evolução de utentes com diferentes dados desconhecidos de impreciso
evolucao( [utente(A,B,C,D,E,F,G,H,I,J) | R], Tipo ) :-
  Tipo = impreciso,
  findall( Invariante, +utente(A,B,C,D,E,F,G,H,I,J)::Invariante, Lista),
  insercao(excecao(utente(A,B,C,D,E,F,G,H,I,J))),
  insercao(impreciso(A,utente)),
  teste(Lista),
  removeIncerto(A,utente),
  evolucao( R, impreciso ).

evolucao( [], tipo ).
```

```
% Permite evolucao de utentes com diferentes dados desconhecidos de incerto
evolucao( utente(A,B,C,D,E,F,G,H,I,J), Tipo, Desconhecido ) :-
    Tipo == incerto,
    (
        Desconhecido == segurança_social; Desconhecido == nome; Desconhecido == dataNascimento;
        Desconhecido == email; Desconhecido == telefone; Desconhecido == morada;
        Desconhecido == profissao; Desconhecido == doençasCronicas; Desconhecido == centroSaude
    ),
    (
        findall(Invariante, +utente(A,desconhecido,C,D,E,F,G,H,I,J)::Invariante, Lista), teste(Lista), insercao(utente(A,desconhecido,C,D,E,F,G,H,I,J)));
        findall(Invariante, +utente(A,B,desconhecido,D,E,F,G,H,I,J)::Invariante, Lista), teste(Lista), insercao(utente(A,B,desconhecido,D,E,F,G,H,I,J));
        findall(Invariante, +utente(A,B,C,desconhecido,E,F,G,H,I,J)::Invariante, Lista), teste(Lista), insercao(utente(A,B,C,desconhecido,E,F,G,H,I,J));
        findall(Invariante, +utente(A,B,C,D,desconhecido,F,G,H,I,J)::Invariante, Lista), teste(Lista), insercao(utente(A,B,C,D,desconhecido,F,G,H,I,J));
        findall(Invariante, +utente(A,B,C,D,E,desconhecido,G,H,I,J)::Invariante, Lista), teste(Lista), insercao(utente(A,B,C,D,E,desconhecido,G,H,I,J));
        findall(Invariante, +utente(A,B,C,D,E,F,desconhecido,H,I,J)::Invariante, Lista), teste(Lista), insercao(utente(A,B,C,D,E,F,desconhecido,H,I,J));
        findall(Invariante, +utente(A,B,C,D,E,F,G,desconhecido,I,J)::Invariante, Lista), teste(Lista), insercao(utente(A,B,C,D,E,F,G,desconhecido,I,J));
        findall(Invariante, +utente(A,B,C,D,E,F,G,H,desconhecido,J)::Invariante, Lista), teste(Lista), insercao(utente(A,B,C,D,E,F,G,H,desconhecido,J));
        findall(Invariante, +utente(A,B,C,D,E,F,G,H,I,desconhecido)::Invariante, Lista), teste(Lista), insercao(utente(A,B,C,D,E,F,G,H,I,desconhecido))
    ),
    insercao(incerto(A,utente)).
```

```
% Permite evolucao de utentes com diferentes dados desconhecidos de incerto
evolucao( utente(A,B,C,D,E,F,G,H,I,J), Tipo, Desconhecido ) :-
    Tipo == incerto,
    (
        Desconhecido == segurança_social; Desconhecido == nome; Desconhecido == dataNascimento;
        Desconhecido == email; Desconhecido == telefone; Desconhecido == morada;
        Desconhecido == profissao; Desconhecido == doençasCronicas; Desconhecido == centroSaude
    ),
    (
        findall(Invariante, +utente(A,desconhecido,C,D,E,F,G,H,I,J)::Invariante, Lista), teste(Lista), insercao(utente(A,desconhecido,C,D,E,F,G,H,I,J));
        findall(Invariante, +utente(A,B,desconhecido,D,E,F,G,H,I,J)::Invariante, Lista), teste(Lista), insercao(utente(A,B,desconhecido,D,E,F,G,H,I,J));
        findall(Invariante, +utente(A,B,C,desconhecido,E,F,G,H,I,J)::Invariante, Lista), teste(Lista), insercao(utente(A,B,C,desconhecido,E,F,G,H,I,J));
        findall(Invariante, +utente(A,B,C,D,desconhecido,F,G,H,I,J)::Invariante, Lista), teste(Lista), insercao(utente(A,B,C,D,desconhecido,F,G,H,I,J));
        findall(Invariante, +utente(A,B,C,D,E,desconhecido,G,H,I,J)::Invariante, Lista), teste(Lista), insercao(utente(A,B,C,D,E,desconhecido,G,H,I,J));
        findall(Invariante, +utente(A,B,C,D,E,F,desconhecido,H,I,J)::Invariante, Lista), teste(Lista), insercao(utente(A,B,C,D,E,F,desconhecido,H,I,J));
        findall(Invariante, +utente(A,B,C,D,E,F,G,desconhecido,I,J)::Invariante, Lista), teste(Lista), insercao(utente(A,B,C,D,E,F,G,desconhecido,I,J));
        findall(Invariante, +utente(A,B,C,D,E,F,G,H,desconhecido,J)::Invariante, Lista), teste(Lista), insercao(utente(A,B,C,D,E,F,G,H,desconhecido,J));
        findall(Invariante, +utente(A,B,C,D,E,F,G,H,I,desconhecido)::Invariante, Lista), teste(Lista), insercao(utente(A,B,C,D,E,F,G,H,I,desconhecido))
    ),
    insercao(incerto(A,utente)).
```

Centro de Saúde

- Negativo
- Incerto
- Interdito
- Impreciso

```
% Permite evolução de utentes com diferentes dados desconhecidos de impreciso
evolucao( [centrosaude(A,B,C,D,E) | R], Tipo ) :-
    Tipo = impreciso,
    findall( Invariante, +centrosaude(A,B,C,D,E)::Invariante, Lista),
    insercao(excecao(centrosaude(A,B,C,D,E))),
    insercao(impreciso(A,centrosaude)),
    teste(Lista),
    removeCentroSaude(A),
    removeIncerto(A,centrosaude),
    evolucao( R, impreciso ).

evolucao( [], tipo ).
```

```
% Permite evolução de utentes com diferentes dados negativos
evolucao( centrosaude( A,B,C,D,E ), Tipo ) :-
    Tipo == negativo,
    findall( Invariante, +(-centrosaude( A,B,C,D,E ))::Invariante, Lista),
    insercao( centrosaude( A,B,C,D,E ) ),
    insercao(negativo(A,centrosaude)),
    teste( Lista ),
    removeCentroSaude(A),
    removeIncerto(A,centrosaude),
    removeImpreciso(A,centrosaude).
```

```
% Permite evolução de utentes com diferentes dados desconhecidos de incerto
evolucao( centrosaude(A,B,C,D,E), Tipo, Desconhecido ) :-
    Tipo == incerto,
    (
        Desconhecido == nome; Desconhecido == morada; Desconhecido == telefone; Desconhecido == email
    ),
    (
        findall( Invariante, +centrosaude(A,desconhecido,C,D,E)::Invariante, Lista),teste(Lista),insercao(centrosaude(A,desconhecido,C,D,E));
        findall( Invariante, +centrosaude(A,B,desconhecido,D,E)::Invariante, Lista),teste(Lista),insercao(centrosaude(A,B,desconhecido,D,E));
        findall( Invariante, +centrosaude(A,B,C,desconhecido,E)::Invariante, Lista),teste(Lista),insercao(centrosaude(A,B,C,desconhecido,E));
        findall( Invariante, +centrosaude(A,B,C,D,desconhecido)::Invariante, Lista),teste(Lista),insercao(centrosaude(A,B,C,D,desconhecido))
    ),
    insercao(incerto(A,centrosaude)).
```

```
% Permite evolucao de utentes com diferentes dados desconhecidos de interdito
evolucao( centrosaude(A,B,C,D,E), Tipo, Desconhecido ) :-
    Tipo == interdito,
    (
        | Desconhecido == nome; Desconhecido == morada; Desconhecido == telefone; Desconhecido == email
    ),
    (
        | findall( Invariante, +centrosaude(A,semNome,C,D,E)::Invariante, Lista), teste(Lista), insercao(centrosaude(A,semNome,C,D,E));
        | findall( Invariante, +centrosaude(A,B,semMorada,D,E)::Invariante, Lista), teste(Lista), insercao(centrosaude(A,B,semMorada,D,E));
        | findall( Invariante, +centrosaude(A,B,C,semTelefone,E)::Invariante, Lista), teste(Lista), insercao(centrosaude(A,B,C,semTelefone,E));
        | findall( Invariante, +centrosaude(A,B,C,D,semEmail)::Invariante, Lista), teste(Lista), insercao(centrosaude(A,B,C,D,semEmail))
    ),
    insercao(interdito(A,centrosaude)).
```

Medico

- Negativo
- Incerto
- Interdito
- Impreciso

```
% Permite evolucao de utentes com diferentes dados desconhecidos de interdito
evolucao( medico(A,B,C,D,E), Tipo, Desconhecido ) :-
    Tipo == interdito,
    (
        | Desconhecido == nome; Desconhecido == idade; Desconhecido == genero; Desconhecido == centrosaude
    ),
    (
        | findall( Invariante, +medico(A,semNome,C,D,E)::Invariante, Lista), teste(Lista), insercao(medico(A,semNome,C,D,E));
        | findall( Invariante, +medico(A,B,semIdade,D,E)::Invariante, Lista), teste(Lista), insercao(medico(A,B,semIdade,D,E));
        | findall( Invariante, +medico(A,B,C,semGenero,E)::Invariante, Lista), teste(Lista), insercao(medico(A,B,C,semGenero,E));
        | findall( Invariante, +medico(A,B,C,D,semCentroSaude)::Invariante, Lista), teste(Lista), insercao(medico(A,B,C,D,semCentroSaude))
    ),
    insercao(interdito(A,medico)).
```

```
% Permite evolucao de utentes com diferentes dados desconhecidos de impreciso
evolucao( [medico(A,B,C,D,E) | R], Tipo ) :-
    Tipo = impreciso,
    findall( Invariante, +medico(A,B,C,D,E)::Invariante, Lista),
    insercao(excecao(medico(A,B,C,D,E))),
    insercao(impreciso(A,medico)),
    teste(Lista),
    removeMedico(A),
    removeIncerto(A,medico),
    evolucao( R, impreciso ).

evolucao( [], tipo ).
```

```
% Permite evolucao de medicos com diferentes dados negativos
evolucao( medico( A,B,C,D,E ), Tipo ) :-
    Tipo == negativo,
    findall( Invariante, +(-medico( A,B,C,D,E ))::Invariante, Lista),
    insercao( medico( A,B,C,D,E ) ),
    insercao(negativo(A,medico)),
    teste( Lista ),
    removeMedico(A),
    removeInceto(A,medico),
    removeImpreciso(A,medico).
```

```
% Permite evolucao de medicos com diferentes dados desconhecidos de incerto
evolucao( medico(A,B,C,D,E), Tipo, Desconhecido ) :-
    Tipo == incerto,
    (
        Desconhecido == nome; Desconhecido == idade; Desconhecido == genero; Desconhecido == centrosaude
    ),
    (
        findall( Invariante, +medico(A,desconhecido,C,D,E)::Invariante, Lista),teste(Lista),insercao(medico(A,desconhecido,C,D,E));
        findall( Invariante, +medico(A,B,desconhecido,D,E)::Invariante, Lista),teste(Lista),insercao(medico(A,B,desconhecido,D,E));
        findall( Invariante, +medico(A,B,C,desconhecido,E)::Invariante, Lista),teste(Lista),insercao(medico(A,B,C,desconhecido,E));
        findall( Invariante, +medico(A,B,C,D,desconhecido)::Invariante, Lista),teste(Lista),insercao(medico(A,B,C,D,desconhecido))
    ),
    insercao(incerto(A,medico)).
```

Enfermeiro

- Negativo
- Incerto
- Interdito
- Impreciso

```
% Permite evolucao de enfermeiros com diferentes dados negativos
evolucao( enfermeiro( A,B,C,D,E ), Tipo ) :-
    Tipo == negativo,
    findall( Invariante, +(-enfermeiro( A,B,C,D,E ))::Invariante, Lista),
    insercao( enfermeiro( A,B,C,D,E ) ),
    insercao(negativo(A,enfermeiro)),
    teste( Lista ),
    removeEnfermeiro(A),
    removeInceto(A,enfermeiro),
    removeImpreciso(A,enfermeiro).
```

```
% Permite evolucao de medicos com diferentes dados desconhecidos de incerto
evolucao( enfermeiro(A,B,C,D,E), Tipo, Desconhecido ) :-
    Tipo == incerto,
    (
        Desconhecido == nome; Desconhecido == idade; Desconhecido == genero; Desconhecido == centrosaude
    ),
    (
        findall( Invariante, +enfermeiro(A,desconhecido,C,D,E)::Invariante, Lista), teste(Lista), insercao(enfermeiro(A,desconhecido,C,D,E));
        findall( Invariante, +enfermeiro(A,B,desconhecido,D,E)::Invariante, Lista), teste(Lista), insercao(enfermeiro(A,B,desconhecido,D,E));
        findall( Invariante, +enfermeiro(A,B,C,desconhecido,E)::Invariante, Lista), teste(Lista), insercao(enfermeiro(A,B,C,desconhecido,E));
        findall( Invariante, +enfermeiro(A,B,C,D,desconhecido)::Invariante, Lista), teste(Lista), insercao(enfermeiro(A,B,C,D,desconhecido))
    ),
    insercao(incerto(A,enfermeiro)).
```

```
% Permite evolucao de utentes com diferentes dados desconhecidos de interdito
evolucao( enfermeiro(A,B,C,D,E), Tipo, Desconhecido ) :-
    Tipo == interdito,
    (
        Desconhecido == nome; Desconhecido == idade; Desconhecido == genero; Desconhecido == centrosaude
    ),
    (
        findall( Invariante, +enfermeiro(A,semNome,C,D,E)::Invariante, Lista), teste(Lista), insercao(enfermeiro(A,semNome,C,D,E));
        findall( Invariante, +enfermeiro(A,B,semIdade,D,E)::Invariante, Lista), teste(Lista), insercao(enfermeiro(A,B,semIdade,D,E));
        findall( Invariante, +enfermeiro(A,B,C,semGenero,E)::Invariante, Lista), teste(Lista), insercao(enfermeiro(A,B,C,semGenero,E));
        findall( Invariante, +enfermeiro(A,B,C,D,semCentroSaude)::Invariante, Lista), teste(Lista), insercao(enfermeiro(A,B,C,D,semCentroSaude))
    ),
    insercao(interdito(A,enfermeiro)).
```

```
% Permite evolucao de utentes com diferentes dados desconhecidos de impreciso
evolucao( [enfermeiro(A,B,C,D,E) | R], Tipo ) :-
    Tipo = impreciso,
    findall( Invariante, +enfermeiro(A,B,C,D,E)::Invariante, Lista),
    insercao(excecao(enfermeiro(A,B,C,D,E))),
    insercao(impreciso(A,enfermeiro)),
    teste(Lista),
    removeEnfermeiro(A),
    removeInceto(A,enfermeiro),
    evolucao( R, impreciso ).

evolucao( [], tipo ).
```

Staff

- Negativo
- Incerto
- Interdito
- Impreciso

```
% Permite evolucao de enfermeiros com diferentes dados negativos
evolucao( staff( A,B,C,D ), Tipo ) :-
    Tipo == negativo,
    findall( Invariante, +(-staff( A,B,C,D ))::Invariante, Lista),
    insercao( staff( A,B,C,D ) ),
    insercao(negativo(A,staff)),
    teste( Lista ),
    removeStaff(A),
    removeIncerto(A,staff),
    removeImpreciso(A,staff).
```

```
% Permite evolucao de medicos com diferentes dados desconhecidos de incerto
evolucao( staff(A,B,C,D), Tipo, Desconhecido ) :-
    Tipo == incerto,
    (
        Desconhecido == idCentro; Desconhecido == nome; Desconhecido == email
    ),
    (
        findall( Invariante, +staff(A,desconhecido,C,D)::Invariante, Lista),teste(Lista),insercao(staff(A,desconhecido,C,D));
        findall( Invariante, +staff(A,B,desconhecido,D)::Invariante, Lista),teste(Lista),insercao(staff(A,B,desconhecido,D));
        findall( Invariante, +staff(A,B,C,desconhecido)::Invariante, Lista),teste(Lista),insercao(staff(A,B,C,desconhecido))
    ),
    insercao(incerto(A,staff)).
```

```
% Permite evolucao de utentes com diferentes dados desconhecidos de interdito
evolucao( staff(A,B,C,D), Tipo, Desconhecido ) :-
    Tipo == interdito,
    (
        Desconhecido == idCentro; Desconhecido == nome; Desconhecido == email
    ),
    (
        findall( Invariante, +staff(A,semCentroSaude,C,D)::Invariante, Lista),teste(Lista),insercao(staff(A,semCentroSaude,C,D));
        findall( Invariante, +staff(A,B,semNome,D)::Invariante, Lista),teste(Lista),insercao(staff(A,B,semNome,D));
        findall( Invariante, +staff(A,B,C,semEmail)::Invariante, Lista),teste(Lista),insercao(staff(A,B,C,semEmail))
    ),
    insercao(interdito(A,staff)).
```

```
% Permite evolucao de utentes com diferentes dados desconhecidos de impreciso
evolucao( [staff(A,B,C,D) | R], Tipo ) :-
    Tipo = impreciso,
    findall( Invariante, +staff(A,B,C,D)::Invariante, Lista),
    insercao(excecao(staff(A,B,C,D))),
    insercao(impreciso(A,staff)),
    teste(Lista),
    removeStaff(A),
    removeInceto(A,staff),
    evolucao( R, impreciso ).

evolucao( [], tipo ).
```

Vacinação

- Negativo
- Incerto
- Interdito
- Impreciso

```
% Permite evolucao de utentes com diferentes dados negativos
evolucao( vacinacao( A,B,C,D,E,F,G ), Tipo ) :-
    Tipo == negativo,
    findall( Invariante, +(-vacinacao( A,B,C,D,E,F,G ))::Invariante, Lista),
    insercao( vacinacao( A,B,C,D,E,F,G ) ),
    insercao(negativoVacinacao(A,B,C,D,E,vacinacao)),
    teste( Lista ),
    removeInceto(A,B,C,D,E,vacinacao),
    removeImpreciso(A,B,C,D,E,vacinacao).
```

```
% Permite evolucao de utentes com diferentes dados desconhecidos de incerto
evolucao( vacinacao(A,B,C,D,E,F,G), Tipo, Desconhecido ) :-
    Tipo == incerto,
    (
        Desconhecido == staff; Desconhecido == utente; Desconhecido == dia;
        Desconhecido == mes; Desconhecido == ano; Desconhecido == tipoVaccina;
        Desconhecido == toma
    ),
    (
        findall( Invariante, +vacinacao(desconhecido,B,C,D,E,F,G)::Invariante, Lista), teste(Lista), insercao(vacinacao(desconhecido,B,C,D,E,F,G));
        findall( Invariante, +vacinacao(A,desconhecido,C,D,E,F,G)::Invariante, Lista), teste(Lista), insercao(vacinacao(A,desconhecido,C,D,E,F,G));
        findall( Invariante, +vacinacao(A,B,desconhecido,D,E,F,G)::Invariante, Lista), teste(Lista), insercao(vacinacao(A,B,desconhecido,D,E,F,G));
        findall( Invariante, +vacinacao(A,B,C,desconhecido,E,F,G)::Invariante, Lista), teste(Lista), insercao(vacinacao(A,B,C,desconhecido,E,F,G));
        findall( Invariante, +vacinacao(A,B,C,D,desconhecido,F,G)::Invariante, Lista), teste(Lista), insercao(vacinacao(A,B,C,D,desconhecido,F,G));
        findall( Invariante, +vacinacao(A,B,C,D,E,desconhecido,G)::Invariante, Lista), teste(Lista), insercao(vacinacao(A,B,C,D,E,desconhecido,G));
        findall( Invariante, +vacinacao(A,B,C,D,E,F,desconhecido)::Invariante, Lista), teste(Lista), insercao(vacinacao(A,B,C,D,E,F,desconhecido))
    ),
    insercao(incertoVacinacao(A,B,C,D,E,vacinacao)).
```

```
% Permite evolucao de utentes com diferentes dados desconhecidos de interdito
evolucao( vacinacao(A,B,C,D,E,F,G), Tipo, Desconhecido ) :-
    Tipo == interdito,
    (
        Desconhecido == staff; Desconhecido == utente; Desconhecido == dia;
        Desconhecido == mes; Desconhecido == ano; Desconhecido == tipoVaccina;
        Desconhecido == toma
    ),
    (
        findall( Invariante, +vacinacao(semStaff,B,C,D,E,F,G)::Invariante, Lista), teste(Lista), insercao(vacinacao(semStaff,B,C,D,E,F,G));
        findall( Invariante, +vacinacao(A,semUtente,C,D,E,F,G)::Invariante, Lista), teste(Lista), insercao(vacinacao(A,semUtente,C,D,E,F,G));
        findall( Invariante, +vacinacao(A,B,semDia,D,E,F,G)::Invariante, Lista), teste(Lista), insercao(vacinacao(A,B,semDia,D,E,F,G));
        findall( Invariante, +vacinacao(A,B,C,semMes,E,F,G)::Invariante, Lista), teste(Lista), insercao(vacinacao(A,B,C,semMes,E,F,G));
        findall( Invariante, +vacinacao(A,B,C,D,semAno,F,G)::Invariante, Lista), teste(Lista), insercao(vacinacao(A,B,C,D,semAno,F,G));
        findall( Invariante, +vacinacao(A,B,C,D,E,semTipoVaccina,G)::Invariante, Lista), teste(Lista), insercao(vacinacao(A,B,C,D,E,semTipoVaccina,G));
        findall( Invariante, +vacinacao(A,B,C,D,E,semToma)::Invariante, Lista), teste(Lista), insercao(vacinacao(A,B,C,D,E,semToma))
    ),
    insercao(interditoVacinacao(A,B,C,D,E,vacinacao)).
```

```
% Permite evolucao de utentes com diferentes dados desconhecidos de impreciso
evolucao( [vacinacao(A,B,C,D,E,F,G) | R], Tipo ) :-
    Tipo = impreciso,
    findall( Invariante, +vacinacao(A,B,C,D,E,F,G)::Invariante, Lista),
    insercao(excecao(vacinacao(A,B,C,D,E,F,G))),
    insercao(imprecisoVacinacao(A,B,C,D,E,vacinacao)),
    teste(Lista),
    removeIncerto(A,B,C,D,E,vacinacao),
    evolucao( R, impreciso ).

evolucao( [], tipo ).
```

4.4 Desenvolver um sistema de inferência capaz de implementar os mecanismos de raciocínio inerentes a estes sistemas

Com o intuito de obter respostas relativamente a determinadas questões, tornou-se essencial desenvolver sistemas de inferências capazes de implementar os mecanismos de raciocínio inerentes a estes.

Assim, começamos por implementar o caso simples em que o sistema apenas recebe uma questão.

Na eventualidade de a questão estar na base de conhecimento a resposta será verdadeiro e se não estiver será falso.

Caso não haja nenhuma informação acerca da mesma na base de conhecimento esta será desconhecido.

```
%--  
% Extensão do meta-predicado nao: Questao -> {V,F}  
nao( Questao ) :- Questao, !, fail.  
nao( _ ).  
  
%--  
% sistema de inferência que permite reconhecer se é V/F ou desconhecido  
si(Questao,verdadeiro) :- Questao.  
si(Questao,falso) :- not(Questao).  
si(Questao,desconhecido) :- nao(Questao), nao(not(Questao)).  
  
% meta-predicado conjuncao:  
% Q1, Q2, Resposta -> {V,F, D}  
conjuncao( verdadeiro, verdadeiro, verdadeiro ).  
conjuncao( falso, _, falso ).  
conjuncao( _, falso, falso ).  
conjuncao( desconhecido, verdadeiro, desconhecido ).  
conjuncao( V, desconhecido, desconhecido ) :- V != falso.  
  
% meta-predicado disjuncao:  
% Q1, Q2, Resposta -> {V,F, D}  
disjuncao( verdadeiro, _, verdadeiro ).  
disjuncao( _, verdadeiro, verdadeiro ).  
disjuncao( desconhecido, V, desconhecido ) :- V != verdadeiro.  
disjuncao( V, desconhecido, desconhecido ) :- V != verdadeiro.  
disjuncao( falso, falso, falso ).  
  
% meta-predicado siLista:  
% Lista de questões, Lista de respostas -> {V,F, D}  
siLista([],[]).  
siLista([Q|T],[R|S]) :- si(Q,R), siLista(T,S).  
  
% meta-predicado siComposicao  
% Composição de questões, resposta -> {V,F, D}  
siComposto(Q && C, R) :- si(Q,RQ), siComposto(C,RC), conjuncao(RQ,RC,R).  
siComposto(Q $$ C, R) :- si(Q,RQ), siComposto(C,RC), disjuncao(RQ,RC,R).  
siComposto(Q, R) :- si(Q,R).
```

Para definir este predicado, foi necessário definir os meta-predicado conjuncao e disjuncao. É importante referir que consideramos que `&&` representa uma conjunção e `$$` uma disjunção. O resultado de uma conjunção só deverá ser verdadeiro, caso ambas as questões sejam verdadeiras e deverá ser falso na eventualidade de existir uma questão falsa. Nas restantes situações o resultado deverá ser desconhecido. Tal comportamento, pode ser verificado na tabela seguinte:

QUESTÃO	QUESTÃO	RESULTADO
VERDADEIRO	VERDADEIRO	VERDADEIRO
VERDADEIRO	DESCONHECIDO	DESCONHECIDO
VERDADEIRO	FALSO	FALSO
FALSO	FALSO	FALSO
FALSO	VERDADEIRO	FALSO
FALSO	DESCONHECIDO	FALSO
DESCONHECIDO	DESCONHECIDO	DESCONHECIDO
DESCONHECIDO	VERDADEIRO	DESCONHECIDO
DESCONHECIDO	FALSO	FALSO

Figura 4.9: Resultados de uma conjunção

Na disjunção o resultado deverá ser verdadeiro sempre que uma das questões seja verdadeira e será falso caso ambas sejam falsas. Nas outras situações o resultado deverá ser desconhecido, como se pode observar de seguida:

QUESTÃO	QUESTÃO	RESULTADO
VERDADEIRO	VERDADEIRO	VERDADEIRO
VERDADEIRO	DESCONHECIDO	VERDADEIRO
VERDADEIRO	FALSO	VERDADEIRO
FALSO	FALSO	FALSO
FALSO	VERDADEIRO	VERDADEIRO
FALSO	DESCONHECIDO	DESCONHECIDO
DESCONHECIDO	DESCONHECIDO	DESCONHECIDO
DESCONHECIDO	VERDADEIRO	VERDADEIRO
DESCONHECIDO	FALSO	FALSO

Figura 4.10: Resultados de uma disjunção

Capítulo 5

Conclusão

Sendo este a segunda fase do trabalho prático da Unidade Curricular, foi, de certa forma, mais acessível para nós relativamente à primeira, uma vez que já havíamos realizado o projeto anterior na linguagem de programação lógica PROLOG, e desta forma, já tínhamos algum do conhecimento necessário para iniciarmos a resolução do presente exercício.

Apesar de estarmos mais ambientados àquilo que são os sistemas de representação de conhecimento, foram, inevitavelmente, surgindo algumas dificuldades no que toca à descoberta de estratégias que melhor se ajustassem à implementação de alguns requisitos. A nossa maior dificuldade foi conseguir implementar de forma mais correta possível os três tipos de conhecimento imperfeito e ao mesmo tempo implementar os invariantes que com estes devem surgir. No entanto, depois de alguma dedicação e esforço, conseguimos ultrapassar esta dificuldade e várias outras com que mais à frente nos deparamos.

Assim, a realização deste exercício foi bastante importante para continuarmos a adquirir experiência e conhecimento, à medida que vamos entendendo como é que esta linguagem de programação lógica funciona e que problemas podem ser resolvidos e analisados com ela. Conseguimos, desta forma, consolidar não só todos os conhecimentos adquiridos nas aulas práticas, como também vários outros que foram sendo necessários durante a realização deste exercício.

O desenvolvimento deste trabalho ajudou-nos a aprofundar o conhecimento sobre a linguagem de programação PROLOG e a sua utilização na construção de sistemas que tiram partido do pressuposto de mundo aberto para a representação de conhecimento perfeito e conhecimento imperfeito.

Os desafios encontrados ao longo do desenvolvimento do projeto vão de encontro aos exercícios realizados nas aulas práticas, sendo possível aplicar os conhecimentos adquiridos nas mesmas neste trabalho.

Inicialmente, houve uma certa dificuldade em interpretar o enunciado e conseguir aplicar o nosso conhecimento da linguagem ao contexto proposto, no entanto, recorrendo à bibliografia recomendada conseguimos prosseguir com o desenvolvimento.

Por fim, considerámos que todos os objetivos foram alcançados com sucesso, de acordo com as especificações do enunciado.

Capítulo 6

Referências

Contratos Públicos Online. Disponível em:

- [Direção-Geral da Saúde](#)
- ANALIDE, Cesar, NOVAIS, Paulo, NEVES, José, “Sugestões para a Elaboração de Relatórios”, Relatório Técnico, Departamento de Informática, Universidade do Minho, Portugal, 2001.
- BRATKO, Ivan, ”PROLOG: Programming for Artificial Intelligence”, 3rd Edition, Addison-Wesley Longman Publishing Co., Inc., 2000