Universidade do Minho
Escola de Engenharia

Mestrado Integrado em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio

Vacinação COVID

Grupo 23:
Benjamim Coelho, A89616
Júlio Alves, A89468
Henrique Neto, A89618
Carolina Vila Chã, A89495
Leonardo Marreiros, A89537

30 de abril de 2022

Resumo

O presente relatório constitui a descrição de uma extensão do sistema desenvolvido na primeira fase deste projeto.

Este trabalho tem como objetivo principal a extensão à programação em lógica, usando a linguagem de programação em lógica `PROLOG`, no âmbito da representação de conhecimento imperfeito, recorrendo à utilização de valores nulos e da criação de mecanismos de raciocínio adequados e com isto construir um sistema de representação de conhecimento com capacidade para representar um universo de discurso na área da vacinação global da população portuguesa no contexto Covid.

Assim, ao longo deste relatório será apresentado a forma como foi implementado conhecimento positivo e negativo, conhecimento imperfeito, invariantes que designam restrições à inserção e à remoção de conhecimento do sistema, evolução do conhecimento e finalmente um sistema de inferência capaz de implementar os mecanismos de raciocínio inerentes a estes sistemas.

Conteúdo

1 Introdução

Este trabalho tem como principal objetivo a utilização da linguagem de Programação em Lógica e da construção lógica de mecanismos de conhecimento e raciocínio, com o propósito de representar um universo de discurso de um cenário de vacinação contra o Covid. Deste modo, foi desenvolvido um sistema de representação do conhecimento e raciocínio que demonstrasse as funcionalidades subjacentes à programação em lógica estendida e à representação de conhecimento imperfeito, recorrendo à temática dos valores nulos.

Este sistema conta com entidades como utentes, staff que administra as vacinas, centros de saúde onde estas são administradas e finalmente registos com informações sobre cada vacina como quem foi o utente vacinado, o staff que o vacinou, etc.

Para isto, completamos a nossa base de conhecimento com conhecimento perfeito positivo e negativo assim como conhecimento incerto, impreciso e interdito. Além disto, serão apresentados os invariantes estruturais e referenciais que permitem garantir uma correta evolução e involução da base de conhecimento. Exemplos desta evolução e involução serão também apresentados e também o sistema de inferência criado para lidar com os mecanismos de raciocínio inerentes a estes sistemas.

2 Preliminares

Numa fase introdutória à linguagem de programação *Prolog*, explorada no trabalho prático 1, foi considerado que o conhecimento assenta somente em dois valores de verdade ("verdadeiro"ou "falso"), onde informações inseridas através de factos e regras são do tipo "verdadeiro"e toda a informação que não consta na base de conhecimento é do tipo "falso". A este tipo de conhecimento associa-se os seguintes pressupostos: **Pressuposto do Mundo Fechado** que dita que toda a informação que não existe mencionada na base de dados é considerada falsa; **Pressuposto dos Nomes Únicos** onde duas constantes diferentes (que definam valores atómicos ou objectos) designam, necessariamente, duas entidades diferentes do universo de discurso; e finalmente, **Pressuposto do Domínio Fechado** no qual não existem mais objectos no universo de discurso para além daqueles designados por constantes na base de dados.

Sendo assim, numa primeira fase do trabalho, apenas foi representado conhecimento perfeito positivo, no entanto, num contexto real estes pressupostos revelam claramente várias limitações relativamente à representação de conhecimento, visto que os pressupostos admitem apenas o que está descrito.

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. Com o intuito de evoluir a base de conhecimento, surge a necessidade de fazer uma **extensão à Programação em Lógica** para que no universo de discurso seja possível contemplar objetos que não estejam completamente descritos na base de conhecimento - **Pressuposto do Mundo Aberto** que dita que podem existir outros factos ou conclusões verdadeiros para além daqueles representados na base de conhecimento e **Pressuposto do Domínio Aberto** que garante que podem existir mais objectos do universo de discurso para além daqueles designados pelas constantes da base de conhecimento.

2.1 Extensão à Programação em Lógica

Na programação em *Prolog* existem limitações quanto à representação de conhecimento que podem ser superadas através da utilização de extensões.

No projeto anterior adotava-se o **Pressuposto do Mundo Fechado**, ou seja, a única forma de representar conhecimento negativo era através da sua omissão. Nesta lógica, as respostas às questões colocadas são, apenas, de dois tipos: **verdadeiro** ou

falso. Para ultrapassar esta limitação fez-se uma **extensão à Programação em Lógica** que permite representar conhecimento negativo de forma explícita. Torna-se possível distinguir três tipos de conclusões para uma questão: esta pode ser **verdadeira**, **falsa** ou, quando não existe informação que permita inferir uma ou outra das conclusões anteriores, a resposta à questão será **desconhecida**. Assim, esta extensão introduz dois tipos de negação: a **negação por falha na prova** e a **negação forte**. De forma a esclarecer estes conceitos recorre-se aos seguintes exemplos:

NEGAÇÃO POR FALHA NA PROVA: atravessar :- nao(comboio)

Só se atravessa, se não existir prova de que o comboio se está a aproximar. Tal não quer dizer que este não se está a aproximar, apenas pretende transmitir que não há provas da sua aproximação (quem quer atravessar pode não conseguir ver o comboio a aproximar-se). Representado pela utilização do predicado ***nao***.

NEGAÇÃO FORTE: atravessar :- -(carro)

Só se atravessa, se existir prova de que o carro não está a aproximar. Neste caso, existe prova da aproximação de um carro (que pode ser dada, por exemplo, por um sensor). Representado pela conectiva **'-'**.

Note-se que a distinção entre estes dois tipos de negação é de extrema importância, pois a ausência de prova não implica a falsidade do conhecimento.

2.2 Conhecimento imperfeito

Para representar de forma mais completa o conhecimento foi necessário desenvolver um novo sistema de inferência ***si*** que permite tratar o conhecimento imperfeito. Para trabalhar com este tipo de conhecimento não bastam apenas os valores "verdadeiro"e "falso". Assim, para o conhecimento estar completamente definido foi introduzido o valor "desconhecido", que serve para definir objetos que não estejam totalmente descritos na base de conhecimento.

Aquando do estudo de conhecimento imperfeito, para distinguir situações tidas como "verdadeiras", "falsas"e "desconhecidas", recorre-se a **valores nulos**. Neste projeto destacam-se os valores nulos do tipo **incerto**, **impreciso** e **interdito**.

Incerto: valor desconhecido que pode, eventualmente, ser descoberto.

Exemplo: Um centro de saúde mudou de contacto recentemente e desconhece-se

o seu contacto atual.

Impreciso: valor desconhecido que pode, eventualmente, ser descoberto e pertence a um conjunto finito de possibilidades.

Exemplo: O nome da vacina administrada é desconhecido mas sabe-se que é ou: Pfizer, Moderna, Astrazeneca ou Sputnik.

Interdito: valor desconhecido que nunca pode ser descoberto.

Exemplo: O Jonas foi vacinado, no entanto, a designação da vacina é confidencial e ele nunca vai divulgar essa informação.

3 Descrição do Trabalho e Análise de Resultados

A primeira parte deste trabalho passou por descrever os predicados que constituem a base de conhecimento do nosso sistema de representação de conhecimento e raciocínio. Foram adicionadas representações de conhecimento negativo e casos de conhecimento imperfeito na **Base de Conhecimento**.

De seguida, revemos os **Invariantes** estruturais e referenciais que designam restrições à inserção e à remoção de conhecimento do sistema tendo em conta todos os tipos de predicados criados.

A seguir, lidamos com a problemática da **Evolução do Conhecimento** e **Involução do Conhecimento**, criando os procedimentos adequados e testes que irão ser apresentados.

Para finalizar, desenvolvemos um **Sistema de Inferência** que é capaz de implementar os mecanismos de raciocínio inerentes ao sistema.

3.1 Base de Conhecimento

3.1.1 Estrutura

As fontes de conhecimento do nosso sistema de representação de conhecimento ilustram de forma clara aquilo que seria de esperar num contexto de um cenário de vacinação e são os seguintes:

Utente: #IdUtente, Nº Segurança Social, Nome, Data Nasc, Email, Telefone, Morada, Profissão, [Doenças Crónicas], #IdCentroSaúde $\rightsquigarrow$ {V,F,D}

Um utente é representado pelo seu id único, número de segurança social, nome, data de nascimento, email, telefone, morada, profissão, doenças crónicas e id do seu centro de saúde.

```
utente(3,3,'Paulo',data(20,05,1985),'paulo@gmail.com',966123456,'Rua do
↪  fim','Pintor',[],3).
```

Centro de saúde: #IdCentroSaúde, Nome, Morada, Telefone, Email $\rightsquigarrow$ {V,F,D}

Um centro de saúde é representado pelo seu id único, nome, morada, telefone e email.

```
centro_saude(3, 'Centro Maia', 'Rua de Maio', 100300402,
↪  'centromaia@gmail.com').
```

Staff: #IdStaff, #IdCentroSaúde, Nome, email ⇝ {V,F,D}

Um staff é representado pelo seu id único, id do centro de saúde em que atua, nome e email.

```
staff(3,3,'Cecilia Rego','cila@gmail.com').
```

Vacinação Covid: #IdStaff, #IdUtente, Data, Vacina, Toma ⇝ {V,F,D}

Uma vacinação é representado pelo id do utente que a recebeu, id do staff que a administrou, data de administração, designação e toma (1 ou 2).

```
vacinacaoCovid(3,2, data(20,3,2021), 'Pfizer', 1).
```

3.1.2 Representação do Conhecimento Perfeito

O **conhecimento perfeito** divide-se em conhecimento **positivo** e conhecimento **negativo**. Sendo que o positivo já foi tratado no trabalho prático anterior, destaca-se a introdução de conhecimento negativo na base de dados.

Conhecimento Positivo

Começamos por povoar a nossa base de conhecimento com base nas fontes de conhecimento referidas anteriormente acrescentando factos perfeitos positivos ao sistema. De seguida apresentamos alguns exemplos deste tipo de conhecimento:

```
%Extensão do predicado  utente: #IdUtente, Nº Segurança Social, Nome, Data Nasc,
↪  Email, Telefone, Morada, Profissão, [Doenças Crónicas], #IdCentroSaúde ->
↪  {V,F,D}
utente(1,1 , 'Jacarias',data(13,04,2000) , 'jacarias@gmail.com', 780100300, 'Rua do
↪  Moinho','Empreiteiro', ['Asma'], 1).
utente(2,2 , 'Maria',data(1,12,1992) , 'maria@gmail.com', 420100300, 'Rua das Duas
↪  Igrejas','Medica', ['Bronquite'], 2).
utente(3,3,'Paulo',data(20,05,1985),'paulo@gmail.com',966123456,'Rua do
↪  fim','Pintor',[],3).

%Extensão do predicado centro_saúde: #IdCentroSaúde, Nome, Morada, Telefone, Email
↪  -> {V,F,D}
centro_saude(1, 'Centro Freamunde', 'Rua das Couves', 100300400,
↪  'centrofreamunde@gmail.com').
```

```
centro_saude(2, 'Centro Gaia', 'Rua do Hospital', 100300401,
↪  'centrogaia@gmail.com').
centro_saude(3, 'Centro Maia', 'Rua de Maio', 100300402, 'centromaia@gmail.com').

%Extensão do predicado staff: #IdStaff, #IdCentroSaúde, Nome, email -> {V,F,D}
staff(1,1,'Manuela Figueiredo','manuelaf@gmail.com').
staff(2,2,'Alberto Dias','berto@gmail.com').
staff(3,3,'Cecilia Rego','cila@gmail.com').

%Extensão do predicado vacinacaoCovid: #IdStaff, #IdUtente, Data, Vacina, Toma ->
↪  {V,F,D}
vacinacaoCovid(1,1, data(20,3,2021), 'Astrazeneca', 1).
vacinacaoCovid(2,1, data(20,4,2021), 'Astrazeneca', 2).
vacinacaoCovid(3,2, data(20,3,2021), 'Pfizer', 1).
```

Conhecimento Negativo

No caso do conhecimento negativo, adicionamos factos com negação forte. Com a utilização deste tipo de predicado, consegue-se garantir que todo o conhecimento que não se encontra definido (ie, que não existe prova que é verdadeiro ou que corresponde a uma exceção) é considerado como falso quando testado no sistema de inferência.

Procedeu-se à definição dos predicados `-centro_saude`, `-utente`, `-staff` e, por fim, `-vacinacaoCovid` da seguinte forma:

```
-utente(Nu,Niss,N,Dn,E,T,M,P,Doencas,Idc):-
    nao(utente(Nu,Niss,N,Dn,E,T,M,P,Doencas,Idc)),
    nao(excecao(utente(Nu,Niss,N,Dn,E,T,M,P,Doencas,Idc))).

-centro_saude(Idc,N,M,T,E):-
    nao(centro_saude(Idc,N,M,T,E)),
    nao(excecao(centro_saude(Idc,N,M,T,E))).

-staff(Ids,Idc,N,E):-
    nao(staff(Ids,Idc,N,E)),
    nao(excecao(staff(Ids,Idc,N,E))).

-vacinacaoCovid(Ids,Idu,D,V,T):-
    nao(vacinacaoCovid(Ids,Idu,D,V,T)),
    nao(excecao(vacinacaoCovid(Ids,Idu,D,V,T))).
```

Finalmente, os factos inseridos foram:

```
-utente(21,21,'Bernardo Portugal', data(12,10,2000),
↪  'lordbjp9@gmail.com', 935234367, 'Braga', 'Estudante', [], 1).

-centro_saude(11, 'hospital lusiadas','rua abilio mendes', 217704040,
↪  'h-lusiadas@gmail.com').

-staff(10,1,'Kenzo Tenma','kenzoT@gmail.com').

-vacinacaoCovid(10,21,data(24,04,2021),'Moderna',2).
```

3.1.3 Representação do Conhecimento Imperfeito

Com a introdução do conhecimento incerto, temos agora a possibilidade de obter respostas verdadeiras, falsas e **desconhecidas**. Enquanto que as respostas verdadeiras e falsas fazem parte do conhecimento perfeito, as respostas desconhecidas fazem parte do conhecimento imperfeito e podem ser divididas em três tipos de conhecimento nulo: **conhecimento incerto**, **conhecimento impreciso** e **conhecimento interdito**.

Conhecimento Incerto

O conhecimento imperfeito incerto corresponde a conhecimento que é considerado ”desconhecido”. Por exemplo, sabemos que o staff com id 1 se chama Manuela Figueiredo, no entanto, não conhecemos o id do centro de saúde onde trabalha. Quando se pergunta qual é o centro de saúde da Manuela, a solução é desconhecida, já que não se sabe de nenhum staff em concreto nessas condições; apenas se sabe que existe um id de centro no qual o staff Manuela trabalha.

Para representar conhecimento deste tipo, é necessário criar a entidade da fonte de conhecimento com átomos desconhecidos dependendo da informação que não se conhece, e depois criar uma exceção que dita que quando é questionada a entidade com a informação desconhecida o resultado é **desconhecido**.

Apesar de se tratar de algo desconhecido, neste caso, é permitida a evolução dos predicados. Deste modo, um valor nulo incerto pode vir a ser conhecido.

Alguns exemplos de conhecimento imperfeito inseridos na nossa base de conhecimento acerca do utente (e com semelhanças para as entidades restantes) foram:

```
% Id do utente desconhecido
utente(idu_desc,30,'Pedro Mota', data(21,06,1965), 'p_Mota65@gmail.com', 962737383,
↪  'Faro', 'Padeiro', ['Diabetes'], 7).
excecao(utente(Idu,Niss,N,Dn,E,T,M,P,Doencas,Idc)):-
    utente(idu_desc,Niss,N,Dn,E,T,M,P,Doencas,Idc).

% Data de nascimento do utente desconhecida
utente(33,33, 'Johan Liebert', d_desc, 'johanLiebert@hotmail.com', 936587938,
↪  'Travessa do Emigrante', 'Estudante', [], 4).
excecao(utente(Idu,Niss,N,Dn,E,T,M,P,Doencas,Idc)):-
    utente(Idu,Niss,N,d_desc,E,T,M,P,Doencas,Idc).

% Contactos do utente desconhecidos
utente(34,34, 'João Livre', data(03,12,1950), e_desc, c_desc, 'Avenida da
↪  Liberdade', 'Reformado', ['Asma', 'Doença Pulmonar'], 1).
excecao(utente(Idu,Niss,N,Dn,E,T,M,P,Doencas,Idc)):-
    utente(Idu,Niss,N,Dn,e_desc, c_desc,M,P,Doencas,Idc).
```

Conhecimento Impreciso

A diferença entre o valor nulo do tipo incerto e não necessariamente de um conjunto determinado de valores, visto na subsecção anterior, e o valor nulo impreciso que se pretende apresentar agora, desconhecido mas de um conjunto determinado de valores, está precisamente neste facto: este último valor nulo representara um (ou mais) valores de um conjunto finito de valores bem determinados; só não é conhecido, especificamente, qual dos valores concretizará a questão.

Utilizando novamente o exemplo anterior, imaginemos agora que, em vez da Manuela poder possivelmente trabalhar em qualquer um dos centros de saúde, sabe-se agora que ela trabalha ou no centro de saúde com Id 1 ou no centro de saúde com Id 2. De forma a contornar esta situação, são construidas exceções que permitem a existencia de ambas as alternativas, isto é, quando a base de conhecimento é questionada acerca da staff Manuela Figueiredo, o resultado é **falso** caso o id do centro de saúde não for 1 ou 2 e **desconhecido** caso contrário.

Alguns dos exemplos criados para este tipo de conhecimento são os seguintes:

```
% não se sabe se o ID de um utente e 60 ou 61
excecao(utente(60, 60, 'Maria Vasconcelos', data(03,04,1980), ' ', 936382633, 'Praça
↪  da Alegria', 'Florista', [], 3)).
excecao(utente(61, 60, 'Maria Vasconcelos', data(03,04,1980), ' ', 936382633, 'Praça
↪  da Alegria', 'Florista', [], 3)).
```

```
% não se sabe se o NISS de um utente e 60 ou 61
excecao(utente(60, 60, 'Maria Vasconcelos', data(03,04,1980), ' ', 936382633, 'Praça
↪  da Alegria', 'Florista', [], 3)).
excecao(utente(60, 61, 'Maria Vasconcelos', data(03,04,1980), ' ', 936382633, 'Praça
↪  da Alegria', 'Florista', [], 3)).

% não se sabe se o nome de um utente e 'Maria Vasconcelos' ou 'Mário Vasconcelos'
excecao(utente(60, 60, 'Maria Vasconcelos', data(03,04,1980), ' ', 936382633, 'Praça
↪  da Alegria', 'Florista', [], 3)).
excecao(utente(60, 60, 'Mário Vasconcelos', data(03,04,1980), ' ', 936382633, 'Praça
↪  da Alegria', 'Florista', [], 3)).
```

Nestes exemplos, o grau de incerteza é sempre igual, apenas se desconhece qual de dois valores é a resposta precisa. De forma semelhante, ao invés de um número fixo de alternativas, podemos ter também um intervalo fixo de valores. Os exemplos seguintes representam estes casos.

```
% não se sabe se o ID do centro de saúde um utente é 3,4 ou 5
excecao(utente(60, 60, 'Maria Vasconcelos', data(03,04,1980), ' ', 936382633, 'Praça
↪  da Alegria', 'Florista', [], C)):- pertence(C, [3,4,5]).

% não se sabe se a data de nascimento de um utente mas sabe-se que está entre 1980 e
↪  1990
excecao(utente(60, 60, 'Mário Vasconcelos', data(D,M,A), ' ', 936382633, 'Praça da
↪  Alegria', 'Florista', [], 3)):- A >= 1980 , A =< 1990.

%não se sabe se a data em que foi administrada a vacina foi entre janeiro de 2021 e
↪  abril de 2021
excecao(vacinacaoCovid(1,1, data(D,M,2021), 'Moderna', 2)):- M >= 1, M =< 4.
```

Conhecimento Interdito

O conhecimento imperfeito pode-se caraterizar por um valor nulo do tipo interdito que não é passível de ser conhecido. No caso da Manuela, tomando que, desta vez, o centro de saúde onde opera é impossível de conhecer: o campo que diz respeito à morada nas informações do staff é inicializado como "ids_secret", é necessário criar também uma exceção relativa ao campo impossível, para que não seja considerado conhecimento positivo/negativo.

Para além disso, este tipo de conhecimento, como os que precedem, representa valores que não se conhecem mas difere no que diz respeito à evolução. Neste caso, nunca se pode vir a conhecer o valor do id, ou seja, não é permitida a evolução do co-

nhecimento o que vai levar à criação de uma instância de nulo interdito com "ids_secret" para que ao construirmos o invariante que impeça a evolução de conhecimento relativo a este staff possamos verificar se se trata de um staff com informação impossível de conhecer e, consequentemente, impossível de ser adicionada.

Alguns dos exemplos criados para este tipo de conhecimento são os seguintes:

```
% Id do utente interdito
utente(idu_secret, 70, 'Guilherme Rodrigues', data(30,09,1940), ' ', 936382634,
↪  'Praça Conde de Agrolongo', 'Reformado', [], 4).
excecao(utente(Idu,Niss,N,Dn,E,T,M,P,Doencas,Idc)):-
    utente(idu_secret,Niss,N,Dn,E,T,M,P,Doencas,Idc).
nulo(idu_secret).
+utente(Idu,Niss,N,Dn,E,T,M,P,Doencas,Idc)::
    (solucoes((idu_secret,Niss,N,Dn,E,T,M,P,Doencas,Idc),
    (utente(IdU, 70, 'Guilherme Rodrigues', data(30,09,1940), ' ', 936382634, 'Praça
    ↪  Conde de Agrolongo', 'Reformado', [], 4),
    nao(nulo(IdU))),S),comprimento(S,N),N==0).

% Id do staff interdito
staff(ids_secret, 6, 'Luís Dias', 'lgamtdias@gov-saude.pt').
excecao(staff(Ids,Idc,N,E)):-
    staff(ids_secret,Idc,N,E).
nulo(ids_secret).
+staff(Ids,Idc,N,E)::(solucoes((ids_secret,Idc,N,E),
    (staff(Ids, 6, 'Luís Dias', 'lgamtdias@gov-saude.pt'),
    nao(nulo(Ids))),S),comprimento(S,N),N==0).
```

3.2 Invariantes

Os invariantes são regras de teste à consistência da base de dados. De entre os invariantes, importa destacar dois tipos, os **invariantes estruturais** e os **invariantes referenciais**.

Os invariantes estruturais não têm em conta o significado dos factos da base de conhecimento, enquanto que os invariantes referenciais captam o significado dos predicados em jogo.

3.2.1 Invariantes de Inserção

Invariantes estruturais

De modo a garantir que não há informação repetida/duplicada na nossa base de conhecimento, criámos os seguintes invariantes que garantem que uma informação específica só pode existir uma única vez. Nesta fase, foram introduzidos o invariante que não permitem a inserção de exceções repetidas e o invariante que impede a inserção de conhecimento positivo relativo a um utente com nome interdito.

```
%Invariante estrutural: Nao permitir a insercao de exceções repetidas
+excecao(T) :: (solucoes(T, excecao(T), R),
                  comprimento(R, 1)).

%Invariante estrutural: Nao permitir a insercao de conhecimento repetido
+utente(Id, NSS,Nome,D,E,T,M,P,Doencas,CentroS) ::
        (solucoes( (Id,NSS,Nome,D,E,T,M,P,Doencas,CentroS),
        (utente(Id,NSS,Nome,D,E,T,M,P,Doencas,CentroS)), S),
        comprimento(S,N),
        N==1).

% Invariante que impede a inserção de conhecimento positivo relativo
% a um utente com nome interdito
+utente(Id,NSS,Nu,D,E,T,M,P,Doencas,CentroS) ::
    (solucoes((Id,NSS,Nome,D,E,T,M,P,Doencas,CentroS),
            (utente(Id,NSS,Nome,D,E,T,M,P,Doencas,CentroS), nulo(Nome)), S),
            comprimento(S,N),
            N==0).
```

```
%Invariante estrutural:nao permitir a insercao de conhecimento repetido
+centro_saude(Id,Nome,Morada,Tel,Email) ::
        (solucoes( (Id,Nome,Morada,Tel,Email),
        (centro_saude(Id,Nome,Morada,Tel,Email)), S),
        comprimento(S,N),
        N==1).

%Invariante estrutural:nao permitir a insercao de conhecimento repetido
+staff(Id, Centro,Nome,Email) ::
        (solucoes( (Id,Centro,Nome,Email), (staff(Id,Centro,Nome,Email)), S),
        comprimento(S,N),
        N==1).
```

Invariantes referenciais

Quando queremos inserir informação de um novo staff na nossa base de conhecimento, temos de garantir que o centro ao qual ele pertence já está na nossa base de conhecimento. O invariante seguinte verifica se o centro de saúde inserido já existe.

```
%Invariante : O centro de saúde da staff existe.
+staff(Id, Centro,Nome,Email) ::
        (solucoes( (Centro,Nome,Morada,Tel,Email),
        (centro_saude(Centro,Nome,Morada,Tel,Email)), S),
        comprimento(S,N),
        N>0).
```

Não é permitido registar a vacinação de uma dose mais do que uma vez, ou seja, um utente não pode ter tomado a primeira nem segunda doses de vacina mais que uma vez. O seguinte invariante garante esta condição sempre que uma nova vacinacaoCovid é inserida na nossa base de conhecimento.

```
%Invariante : Não pode ter tomada a mesma fase da vacina mais que uma vez
+vacinacaoCovid(S, U, D, V, T) ::
        (solucoes((S,U,D,V,T), vacinacaoCovid(S,U,D,V,T) , L),
        comprimento(L,N), N==1).
```

Também não é permitido um utente ser vacinado com a segunda dose sem ter sido vacinado com a primeira anteriormente. Deste modo, o seguinte invariante garante

que, sempre que uma vacinacaoCovid da segunda dose é inserida na nossa base de conhecimento, já existe informação da vacinação da primeira dose.

```
%Invariante : So pode ter tomado a 2 dose se tomou a primeira
+vacinacaoCovid(S,U,D,V,2)::
        (solucoes((S,U,D,V), vacinacaoCovid(S,U,D,V,'1'), Primeiros),
        comprimento(Primeiros,N),
        N==1).
```

Um utente não pode ser vacinado com diferentes modelos de vacinas em diferentes doses. Ora, o invariante a seguir só é aplicado quando é inserida informação de uma vacinação da segunda dose. Nestes casos, verifica se a vacinação da primeira dose desse mesmo utente é do mesmo modelo que da segunda.

```
%Invariante : A vacina da segunda toma tem de ser o mesmo modelo
+vacinacaoCovid(S, U, D, V, 2) ::
        (solucoes((S,U,D,V,'1'), vacinacaoCovid(S,U,D,V,T) , L),
        nth0(0,L,(_,_,_,C,_)),C==V).
```

De momento todas as vacinas são administradas em 2 doses. Assim, só é permitida a inserção de informação de vacinações cujo números de dose são 1 ou 2. O invariante seguinte verifica esta condição.

```
%Invariante : A toma da vacina so pode ser 1 ou 2
+vacinacaoCovid(_,_,_,_,T)::(T>0,T<3).
```

O trabalhador que administrou a vacina tem de existir na nossa base de conhecimento.

```
%Invariante : O trabalhador que administrou a vacina tem de estar registado
+vacinacaoCovid(Staff,_,_,_,_)::
        (solucoes( (Staff,Centro,Nome,Email),
        (staff(Staff,Centro,Nome,Email)), L),
        comprimento(L,N),
        N>0).
```

O utente que recebeu a vacina tem de existir na nossa base de conhecimento

```
%Invariante : O Utente que rebeu a vacina tem de estar registado
+vacinacaoCovid(_,Ut,_,_,T)::
            (solucoes( (Ut,NSS,Nome,D,E,T,M,P,Doencas,CentroS),
```

```
        (utente(Ut,NSS,Nome,D,E,T,M,P,Doencas,CentroS)), _),
        comprimento(_,N),
        N>0).
```

3.2.2 Invariantes de Remoção

Invariantes estruturais

O invariante seguinte indica que um utente pode ser removido desde que esteja na base de conhecimento.

```
%permite remover se existir na base de conhecimento:
-utente(Id, NSS,Nome,D,E,T,M,P,Doencas,CentroS) ::
                    (solucoes( (Id,NSS,Nome,D,E,T,M,P,Doencas,CentroS),
                    (utente(Id,NSS,Nome,D,E,T,M,P,Doencas,CentroS)), S),
                    comprimento(S,N),
                        N==0).
```

O invariante seguinte indica que um centro de saúde pode ser removido desde que esteja na base de conhecimento.

```
%permite remover se existir na base de conhecimento:
-centro_saude(Id,Nome,Morada,Tel,Email) ::
                (solucoes( (Id,Nome,Morada,Tel,Email),
                (centro_saude(Id,Nome,Morada,Tel,Email)), S),
                comprimento(S,N),
                N==0).
```

O invariante seguinte indica que um elemento do staff pode ser removido desde que esteja na base de conhecimento.

```
%permite remover se existir na base de conhecimento:
-staff(Id, Centro,Nome,Email) ::
        (solucoes( (Id,Centro,Nome,Email), (staff(Id,Centro,Nome,Email)), S),
        comprimento(S,N),
        N==0).
```

O invariante seguinte indica que uma vacinação pode ser removida desde que esteja na base de conhecimento.

```
%permite remover se existir na base de conhecimento:
-vacinacaoCovid(S, U, D, V, T) ::
        (solucoes((S,U,D,V,T), vacinacaoCovid(S,U,D,V,T) , L),
        comprimento(L,N),
        N==0).
```

Invariantes referenciais

O invariante seguinte indica que um utente que esteja presente na base de conhecimento pode ser removido, se e só se, não tiver sido vacinado.

```
%Não permite remover utentes com registos de vacinação
-utente(Id, NSS,Nome,D,E,T,M,P,Doencas,CentroS) ::
        (nao( vacinacaoCovid(_,Id,_,_,_))).
```

O invariante seguinte indica que o registo de uma vacina que seja referente à primeira toma por parte de um utente não pode ser removida se o registo da segunda toma por parte do mesmo paciente ainda esteja presente na base de conhecimento.

```
%Não permite remover a primeira toma vacina se a segunda esteja na
%base de conhecimento
-vacinacaoCovid(S, U, D, V, 1) :: (nao( vacinacaoCovid(_,Id,_,_,2))).
```

O invariante seguinte indica que um centro de saúde que esteja presente na base de conhecimento pode ser removido, se e só se, não houver nenhum elemento do staff presente na base de conhecimento que esteja associado ao centro de saúde em questão.

```
%Não permite remover um centro de saúde se existir staff associada
-centro_saude(Id,_,_,_,_) ::
        (solucoes((Staff,Id,Nome,Email), (staff(Staff,Id,Nome,Email)), S),
        comprimento(S,N),
        N==0).
```

3.3 Evolução do Conhecimento

Uma base de conhecimento pode sofrer alterações relativamente à informação que armazena. Dado que este trabalho retrata uma situação real torna-se imperativo inserir e remover dados. No entanto, como neste sistema se abrange conhecimento imperfeito leva a que hajam certas limitações e problemáticas no processo de evolução do conhecimento que vão ser colmatadas com a implementação de outros predicados **evolucao**.

Nas próximas secções iremos explicar detalhadamente cada um dos procedimentos criados para o efeito bem como exemplos de aplicação dos mesmos.

3.3.1 Evolução do conhecimento perfeito positivo e negativo

Assim como na fase anterior, para inserirmos nova informação na nossa base de conhecimento é importante haver controlo de consistência, pelo que criámos o meta-predicado evolucao. Este predicado procura todos os invariantes que se aplicam ao tipo de informação que estamos a tentar inserir e verifica as suas condições. Se todos os invariantes forem verificados a informação é inserida na base de conhecimento com sucesso.

```
%Insere novo conhecimento na base de dados
evolucao(Termo):-
    solucoes(Invariante, +Termo::Invariante, Lista),
    insercao(Termo),
    teste(Lista).

insercao(Termo) :- assert(Termo).
insercao(Termo) :- retract(Termo),!,fail.
```

Evolução de conhecimento positivo para conhecimento positivo

Para substituir um conhecimento positivo por outro conhecimento positivo, foi criado o predicado `evolucao_PP`. Este predicado permite substituir conhecimentos antigos por conhecimentos atualizados. Primeiro é imposta a condição que impõem que o argumento existente seja verdadeiro, caso seja, o conhecimento existente é removido e é inserido um novo conhecimento com a informação atualizada.

```
% Extensao do predicado que permite a evolucao do conhecimento POSITIVO para outro
↪  POSITIVO: Qnovo,Qpassado --> {V,F}
```

```
evolucao_PP(Qnovo, Qpassado):-
    si(Qpassado, verdadeiro),
    remocao(Qpassado),
    insercao(Qnovo).
```

Um exemplo deste tipo de evolução é:

```
?- si(staff(1,1,'Manuela Figueiredo','manuelaf@gmail.com'),R).
R = verdadeiro .

?- si(staff(1,1,'Maria Figueiredo','manuelaf@gmail.com'),R).
R = falso .

?- evolucao_PP(staff(1,1,'Maria Figueiredo','manuelaf@gmail.com'),
    staff(1,1,'Manuela Figueiredo','manuelaf@gmail.com')).
true .

?- si(staff(1,1,'Maria Figueiredo','manuelaf@gmail.com'),R).
R = verdadeiro .

?- listing(staff).
:- dynamic staff/4.
(...)
staff(1, 1, 'Maria Figueiredo', 'manuelaf@gmail.com').

true.
```

Neste exemplo, começamos por questionar o sistema acerca a existência do staff com id 1, chamada Manuela Figueiredo, cujo resultado, como esperado, é verdadeiro. Imaginemos agora que, por alguma razão, a base de conhecimento estava errada e o nome deste staff na realidade é Maria Figueiredo. Com isto, primeiro verificamos se já existe uma Maria Figueiredo na base de conhecimento e, como não existe, utiliza-se o predicado `evolucao_PP` para atualizar a informação deste staff. Finalmente, verificamos se de facto tinha sido efetuada a mudança com `listing(staff).`.

Evolução de conhecimento negativo para conhecimento positivo

O predicado `evolucao_NP` permite passar de conhecimento negativo presente na base de conhecimento para conhecimento positivo. Ao contrário daquilo que é feito no predicado anterior, primeiro garante-se que o conhecimento negativo existe e depois que o equivalente conhecimento positivo não existe. Depois disto, remove-se o conhecimento

negativo e é inserido o conhecimento positivo equivalente.

```
% Extensao do predicado que permite a evolucao do conhecimento NEGATIVO para
↪  POSITIVO: Qnovo --> {V,F}
evolucao_NP(Qnovo):-
    si(-Qnovo, verdadeiro),
    si(Qnovo,falso),
    remocao(-Qnovo),
    insercao(Qnovo).
```

Um exemplo deste tipo de evolução é:

```
?- si(-staff(10,1,'Kenzo Tenma','kenzoT@gmail.com'),R).
R = verdadeiro .

?- si(staff(10,1,'Kenzo Tenma','kenzoT@gmail.com'),R).
R = falso .

?- evolucao_NP(staff(10,1,'Kenzo Tenma','kenzoT@gmail.com')).
true .

?- si(staff(10,1,'Kenzo Tenma','kenzoT@gmail.com'),R).
R = verdadeiro .

?- listing(staff).
:- dynamic staff/4.
(...)
staff(10, 1, 'Kenzo Tenma', 'kenzoT@gmail.com').

true.
```

Neste exemplo, primeiro verificamos que o staff com nome Kenzo Tenma de facto existe na base de dados como conhecimento negativo e que não existe como conhecimento positivo. Depois utilizamos o predicado `evolucao_NP` para efetuar esta evolução e por fim utilizamos `listing(staff).` para verificar que de facto este conhecimento passou a ser positivo.

3.3.2 Evolução do conhecimento imperfeito

Evolução do conhecimento incerto para conhecimento positivo

O predicado `evolucao_IncP` possibilita a atualização do conhecimento incerto para conhecimento positivo. Para tal, primeiro verifica-se se o conhecimento existente

(*Qpassado*) é verdadeiro, depois, caso seja, remove-se este facto e é inserido o novo conhecimento positivo atualizado (*Qnovo*).

```
% Extensao do predicado que permite a evolucao do conhecimento INCERTO para
↪ POSITIVO: Qnovo, Qpassado--> {V,F}

evolucao_IncP(Qnovo, Qpassado):-
    si(Qpassado,verdadeiro),
    remocao(Qpassado),
    insercao(Qnovo).
```

Um exemplo deste tipo de evolução é:

```
?- si(staff(ids_desc, 4, 'Dinis Brito', 'dinisbrito@saudenorte.gov.pt'),R
    ).
R = verdadeiro .

?- si(staff(9, 4, 'Dinis Brito', 'dinisbrito@saudenorte.gov.pt'),R).
R = desconhecido.

?- evolucao_IncP(staff(9, 4, 'Dinis Brito', 'dinisbrito@saudenorte.gov.pt
    '),staff(ids_desc, 4, 'Dinis Brito', 'dinisbrito@saudenorte.gov.pt')).
true .

?- si(staff(9, 4, 'Dinis Brito', 'dinisbrito@saudenorte.gov.pt'),R).
R = verdadeiro .

?- listing(staff).
:- dynamic staff/4.
(...)
staff(9, 4, 'Dinis Brito', 'dinisbrito@saudenorte.gov.pt').

true.
```

Neste exemplo, primeiro pretendemos provar que o staff Dinis Brito tem um id desconhecido, depois comprovamos este facto ao substituir o campo desconhecido com um valor preciso, cuja resposta foi **desconhecido**, como seria de esperar. Sabemos agora que existe este conhecimento incerto na base de conhecimento. O próximo passo foi utilizar o predicado `evolucao_IncP` para tornar este conhecimento num conhecimento perfeito com a nova informação precisa. Finalmente, utilizamos `listing(staff).` para verificar que de facto este conhecimento passou a ser positivo.

Evolução do conhecimento impreciso para conhecimento positivo

O `evolucao_ImpP` permite atualizar conhecimento impreciso já existente para conhecimento positivo. Neste predicado são passados como argumentos a questão a inserir (*Qnovo*) e os restantes argumentos correspondem aos termos desconhecidos existentes. Primeiro verifica-se que realmente estes últimos correspondem realmente a conhecimento impreciso e, depois disso, são removidos e atualizados pelo novo conhecimento positivo atualizado.

```
% Extensao do predicado que permite a evolucao do conhecimento IMPRECISO para
↪  POSITIVO: Qnovo, Qpassado1, Qpassado2 --> {V,F}

evolucao_ImpP(Qnovo,Qpassado1,Qpassado2):-
    si(Qpassado1,desconhecido),
    si(Qpassado2,desconhecido),
    remocao(excecao(Qpassado1)),
    remocao(excecao(Qpassado2)),
    insercao(Qnovo).
```

Um exemplo deste tipo de evolução é:

```
?- si(staff(71, 4, 'Bruna Magalhães', 'BrunaMagalhãesMedica@saude.pt'),R)
    .
R = desconhecido.

?- si(staff(70, 4, 'Bruna Magalhães', 'BrunaMagalhãesMedica@saude.pt'),R)
    .
R = desconhecido.

?- evolucao_ImpP(staff(70, 4, 'Bruna Magalhães', 'BrunaMagalhã
    esMedica@saude.pt'), staff(71, 4, 'Bruna Magalhães', 'BrunaMagalhã
    esMedica@saude.pt'), staff(70, 4, 'Bruna Magalhães', 'BrunaMagalhã
    esMedica@saude.pt')).
true .

?- si(staff(70, 4, 'Bruna Magalhães', 'BrunaMagalhãesMedica@saude.pt'),R)
    .
R = verdadeiro .

?- listing(staff).
:- dynamic staff/4.
(...)
staff(70, 4, 'Bruna Magalhães', 'BrunaMagalhãesMedica@saude.pt').

true.
```

Neste exemplo, primeiro pretendemos provar que o staff Bruna Magalhães tem um id impreciso (70 ou 71) ao obter uma resposta desconhecida quando questionado por estes valores. Sabemos agora que existe este conhecimento impreciso na base de conhecimento. O próximo passo foi utilizar o predicado `evolucao_ImpP` para tornar este conhecimento num conhecimento perfeito com a nova informação precisa. Finalmente, utilizamos `listing(staff).` para verificar que de facto este conhecimento passou a ser positivo.

Evolução do conhecimento incerto para conhecimento impreciso

O predicado `evolucao_IncImp` permite substituir conhecimento incerto por conhecimento impreciso. Este tem 3 argumentos, 2 relativos à nova informação do tipo impreciso (*Qnovo1* e *Qnovo2*) e um terceiro (*Qpassado*) relativo ao conhecimento incerto pré-existente. Primeiro verifica-se a existência do connhecimento incerto, de seguida este é removido e são adicionadas as novas exceções que correspondem ao novo conhecimento impreciso.

```
% Extensao do predicado que permite a evolucao do conhecimento INCERTO para
↪  IMPRECISO: Qnovo1, Qnovo2, Qpassado --> {V,F}

evolucao_IncImp(Qnovo1, Qnovo2, Qpassado):-
    si(Qpassado,verdadeiro),
    remocao(Qpassado),
    insercao(excecao(Qnovo1)),
    insercao(excecao(Qnovo2)).
```

Um exemplo deste tipo de evolução é:

```
?- si(staff(ids_desc, 4, 'Dinis Brito', 'dinisbrito@saudenorte.gov.pt'),R
   ).
R = verdadeiro .

?- si(staff(9, 4, 'Dinis Brito', 'dinisbrito@saudenorte.gov.pt'),R).
R = desconhecido.

?- si(staff(10, 4, 'Dinis Brito', 'dinisbrito@saudenorte.gov.pt'),R).
R = desconhecido.

?- evolucao_IncImp(staff(9, 4, 'Dinis Brito', 'dinisbrito@saudenorte.gov.
   pt'), staff(10, 4, 'Dinis Brito', 'dinisbrito@saudenorte.gov.pt'),
   staff(ids_desc, 4, 'Dinis Brito', 'dinisbrito@saudenorte.gov.pt')).
true .
```

```
?- si(staff(10, 4, 'Dinis Brito', 'dinisbrito@saudenorte.gov.pt'),R).
R = desconhecido.

?- si(staff(9, 4, 'Dinis Brito', 'dinisbrito@saudenorte.gov.pt'),R).
R = desconhecido.
```

Neste exemplo, primeiro pretendemos provar que o staff Dinis Brito tem um id incerto e questionamos a base de dados com dois ids diferentes obtendo a resposta desconhecida esperada. Sabemos agora que existe este conhecimento incerto na base de conhecimento. O próximo passo foi utilizar o predicado `evolucao_IncImp` para tornar este conhecimento num conhecimento impreciso. Finalmente, voltamos a questionar a base de dados com as mesmas questões colocadas previamente para obter a mesma respostas pois o conhecimento continua a ser imperfeito. No entanto, ao consultar o `listing(staff).` é possível verificar que o staff Dinis Brito já não faz parte da base de conhecimento.

Evolução do conhecimento impreciso para conhecimento incerto

O predicado `evolucao_ImpInc` permite atualizar conhecimento impreciso em conhecimento incerto. A este predicado são passados 5 argumentos: *Qnovo*, *Qnovo_ex*, *Qnovo_exx*, *Qpassado1*, *Qpassado2*.

Começa-se por verificar a presença do conhecimento impreciso (*Qpassado1*, *Qpassado2*) na base de dados. Em segundo lugar, remove-se esse conhecimento impreciso. Finalmente, para representar o conhecimento impreciso é inserido o predicado que terá o valor nulo do tipo impreciso (*Qnovo*) e a respetiva exceção que carateriza este tipo de conhecimento.

```
% Extensao do predicado que permite a evolucao do conhecimento IMPRECISO para INC

evolucao_ImpInc(Qnovo,Qnovo_ex, Qnovo_exx, Qpassado1, Qpassado2):-
    si(Qpassado1,desconhecido),
    si(Qpassado2,desconhecido),
    remocao(excecao(Qpassado1)),
    remocao(excecao(Qpassado2)),
    insercao(Qnovo),
    insercao((excecao(Qnovo_ex) :- Qnovo_exx)).
```

Um exemplo deste tipo de evolução é:

```
?- si(staff(70, 4, 'Bruna Magalhães', 'BrunaMagalhãesMedica@saude.pt'),R)
    .
R = desconhecido.

?- si(staff(71, 4, 'Bruna Magalhães', 'BrunaMagalhãesMedica@saude.pt'),R)
    .
R = desconhecido.

?- evolucao_ImpInc(staff(desconhecido, 4, 'Bruna Magalhães', 'BrunaMagalh
    ãesMedica@saude.pt'), staff(A, B, C, D), staff(desconhecido, B, C, D),
     staff(70, 4, 'Bruna Magalhães', 'BrunaMagalhãesMedica@saude.pt'),
    staff(71, 4, 'Bruna Magalhães', 'BrunaMagalhãesMedica@saude.pt')).
true .

?- listing(staff).
:- dynamic staff/4.
(...)
staff(desconhecido, 4, 'Bruna Magalhães', 'BrunaMagalhãesMedica@saude.pt'
    ).

true.
```

Neste exemplo, primeiro pretendemos provar que o staff Bruna Magalhães tem um id impreciso (70 ou 71) ao obter uma resposta desconhecida quando questionado por estes valores. Sabemos agora que existe este conhecimento impreciso na base de conhecimento. O próximo passo foi utilizar o predicado `evolucao_ImpInc` para tornar este conhecimento num conhecimento incerto. Finalmente, utilizamos o `listing(staff).` para verificar que, de facto, o conhecimento incerto foi introduzido com sucesso.

Inserção de conhecimento Imperfeito

Assim como na fase anterior era possível inserir conhecimento positivo, nesta fase faz sentido desenvolver o predicado `evolucao` para permitir a inserção de conhecimento imperfeito. Com foco nisto, criamos algumas extensões a este predicado.

Quanto ao conhecimento incerto, um exemplo desta evolução para um staff com nome incerto foi (com implementações semelhantes para as restantes fontes de comnhecimento):

```
%Extensão do predicado que permite inserir conhecimento incerto no
↪  caso de um staff com nome desconhecido
```

```
evolucao(staff(Ids,Idc,Nome_desconhecido,E), staff, incerto, nome):-
    evolucao(staff(Ids,Idc,Nome_desconhecido,E)),
    insercao((excecao(staff(Ids,Idc,Nome,E)) :-
                    staff(Ids,Idc,Nome_desconhecido,E))).
```

Em relação ao conhecimento impreciso, extendemos o predicado evolução para ser capaz de inserir um utente com uma data de nascimento entre um intervalo finito de valores:

```
%Extensão do predicado que permite inserir conhecimento impreciso no
↪  caso de um utente com ano de nascença dentro de um conjunto de
↪  valores

evolucao(utente(Idu,Niss,N,Ano_impreciso,E,T,M,P,Doencas,Idc), utente,
↪  impreciso, ano, LimiteInferior, LimiteSuperior):-
    insercao((excecao(utente(Idu,Niss,N,data(D,M,Ano_impreciso),E, T,
    ↪  M, P, Doencas,Idc)):-
        Ano_impreciso >= LimiteInferior, Ano_impreciso =<
        ↪  LimiteSuperior)).
```

Finalmente, para o conhecimento interdito, é possível inserir conhecimento no caso de um utente com nome confidencial. De notar que, para esta extensão, foi necessária a existência de um invariante apresentado anteriormente que impede a inserção de conhecimento positivo relativo a um utente com nome interdito.

```
%Extensão do predicado que permite inserir conhecimento interdito no
↪  caso de um utente com nome secreto

evolucao(utente(Idu,Niss,Nome_interdito,Dn,E,T,M,P,Doencas,Idc),
↪  utente, interdito, nome):-
    evolucao(utente(Idu,Niss,Nome_interdito,Dn,E,T,M,P,Doencas,Idc)),
    insercao((excecao(utente(IdUt,NissUt,Nome,Data,Emp,Tel,Mor, Prof,
    ↪  DoencasUt, IdcUt)):-
                utente(IdUt,NissUt,Nome_interdito,Data,Emp, Tel, Mor,
                ↪  Prof, DoencasUt, IdcUt)),
    insercao(nulo(Nome_interdito)).
```

3.4 Involução do Conhecimento

Ao remover informação da nossa base de conhecimento temos de nos certificar que não há dependências. Por exemplo, não podemos remover um utente antes de remover as vacinações que o envolvem, senão o ID que o referencia na vacinação deixará de ser válido. Deste modo definimos o meta-predicados seguinte, já utilizado na fase anterior, para verificarmos todos os invariantes sempre que removemos informação.

```
involucao(Termo) :-
        solucoes(Invariante,-Termo::Invariante,Lista),
        remocao(Termo),
        teste(Lista).

remocao(Termo) :- retract(Termo).
remocao(Termo) :- assert(Termo),!,fail.
```

3.4.1 Involução do Conhecimento Imperfeito

Da mesma forma que foi feita a evolução de conhecimento imperfeito, também foi feita a sua involução. Este predicados constituem apenas as operações inversas àquelas feitas na evolução, utilizando o meta-predicado `remocao` ao invés do `insercao`.

Um exemplo da involução do conhecimento incerto é o seguinte:

```
%Extensão do predicado que permite remover conhecimento incerto no
↪  caso de um staff com nome desconhecido

involucao(staff(Ids,Idc,Nome_desconhecido,E), staff, incerto, nome):-
    involucao(staff(Ids,Idc,Nome_desconhecido,E)),
    remocao((excecao(staff(Ids,Idc,Nome,E)) :-
                    staff(Ids,Idc,Nome_desconhecido,E))).
```

Quanto à involução de conhecimento impreciso:

```
%Extensão do predicado que permite remover conhecimento impreciso no
↪  caso de um utente com ano de nascença dentro de um conjunto de
↪  valores

involucao(utente(Idu,Niss,N,Ano_impreciso,E,T,M,P,Doencas,Idc), utente,
↪  impreciso, ano, LimiteInferior, LimiteSuperior):-
```

```
    remocao((excecao(utente(Idu,Niss,N,data(D,M,Ano_impreciso),E,T, M,
    ↪  P, Doencas, Idc)):-
        Ano_impreciso >= LimiteInferior, Ano_impreciso =<
        ↪  LimiteSuperior)).
```

Finalmente, para o conhecimento interdito:

```
%Extensão do predicado que permite remover conhecimento interdito no
↪  caso de um utente com nome secreto

involucao(utente(Idu,Niss,Nome_interdito,Dn,E,T,M,P,Doencas,Idc),
↪  utente, interdito, nome):-
    involucao(utente(Idu,Niss,Nome_interdito,Dn,E,T,M,P,Doencas,Idc)),
    remocao((excecao(utente(IdUt,NissUt,Nome,Data,Emp,Tel, Mor, Prof,
    ↪  DoencasUt, IdcUt)):-
                utente(IdUt,NissUt,Nome_interdito,Data,Emp, Tel, Mor,
                ↪  Prof, DoencasUt, IdcUt))),
    remocao(nulo(Nome_interdito)).
```

3.5 Sistema de Inferência

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

Para isto, construímos o predicado `si`.

```
% Extensao do meta-predicado si: Questao,Resposta -> {V,F,D}

si( Questao,verdadeiro ) :-
    Questao.
si( Questao,falso ) :-
    -Questao.
si( Questao,desconhecido ) :-
    nao( Questao ),
    nao( -Questao ).
```

Se *Questao* for verdadeiro em todos os conjuntos e respostas do programa, a resposta é "verdadeiro". Se ¬ *Questao* for verdadeiro em todos os conjuntos de respostas dos programa, ie, que se consegue provar por negação da questão, a resposta é "falso". No caso de nenhuma destas situações se verificar, a resposta é "desconhecido".

4 Conclusões e Sugestões

Com este trabalho prático consolidar e explorar melhor o conhecimento relativo à programação relacional no âmbito dos temas de Programação Lógica Estendida e Conhecimento Imperfeito.

Desta forma, o sistema de inferência previamente desenvolvido permite agora testar qualquer forma de conhecimento, quer este seja perfeito (positivo ou negativo) ou imperfeito (incerto, impreciso, interdito) a partir de meta predicados desenvolvidos para as representação de exceções, valores nulos e invariantes que garantem a evolução correta da base de conhecimento.

Nesta fase, a nossa maior dificuldade foi estabelecer os invariantes responsáveis por gerir os diferentes tipos de conhecimento imperfeito e as suas evoluções, por se tratarem de tipos de conhecimento mais imprevisíveis e cuja manipulação não é tão imediata como no conhecimento positivo.

Numa oportunidade futura gostariamos de inserir mais invariantes assim como evolução e involução de mais casos de conhecimento imperfeito.