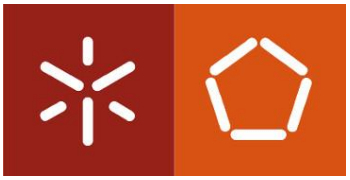

UNIVERSIDADE DO MINHO

DEPARTAMENTO DE INFORMÁTICA

TRABALHO PRÁTICO – Fase 2

Programação em Lógica estendida e Conhecimento imperfeito

Sistemas de Representação de Conhecimento e Raciocínio
2º Semestre – 2020/21

Braga, abril de 2021

Autores:

(Grupo 1)

Ana Filipa Pereira A89589

Carolina Santejo A89500

Raquel Costa A89464

Sara Marques A89477

RESUMO

O relatório que se segue tem como objetivo relatar o desenvolvimento da segunda fase do projeto da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio.

Uma das suas finalidades consistiu em estender o sistema referente à vacinação atual da população portuguesa, que terá sido elaborado ao longo da primeira fase. Desta forma, foi imprescindível a realização de alterações para o aprimoramento do mesmo.

Para tal, introduziu-se a noção de conhecimento imperfeito em contraste ao perfeito, representando vários casos que ilustram o mesmo, sem esquecer a distinção entre conhecimento perfeito positivo e negativo. Além disso, foi fundamental a elaboração de uma solução para a problemática da evolução do conhecimento.

Todo este processo irá resultar num sistema de inferência capaz de implementar e responder às necessidades inerentes à temática apresentada, pondo em prática o conhecimento adquirido na linguagem Prolog.

Índice

Índice de Figuras

Introdução

Nesta segunda fase do trabalho prático da cadeira de SRCR, foi pedido que complementássemos o sistema representação de conhecimento e raciocínio desenvolvido, incluindo agora conhecimento negativo e conhecimento imperfeito. Estes tipos de conhecimentos tiveram de ser manipulados de forma cautelosa de forma a garantir a consistência do nosso sistema. Para isto foi necessário adicionar novos invariantes e predicados.

Ao longo do relatório, serão explicadas não só as alterações feitas à primeira fase, bem como todas as decisões tomadas pelo grupo para resolver o problema.

Preliminares

Na presente fase do projeto abandonou-se o ***Pressuposto do Mundo Fechado***, o ***Pressuposto dos Nomes Únicos*** e o ***Pressuposto do Domínio Fechado***, uma vez que agora o trabalho desenvolvido assenta-se em fundamentos próprios da programação em lógica estendida. Deste modo, considera-se que não é possível assumir que a informação representada é a única que é válida, nem tão pouco que é, exclusivamente, a única existente no mundo exterior. Sendo assim, para a elaboração do trabalho foi fundamental sustentar o mesmo com base nos seguintes princípios:

- ✓ ***Pressuposto do Mundo Aberto*** – podem existir outros factos verdadeiros para além daqueles representados na base de conhecimento;

- ✓ ***Pressuposto dos Nomes Únicos*** – duas constantes diferentes terão de designar obrigatoriamente duas entidades diferentes;

- ✓ ***Pressuposto do Domínio Aberto*** – é admitida a existência de mais objetos do universo de discurso para além daqueles designados na base de conhecimento

Assim sendo, foram considerados três tipos de valores lógicos, o ***Verdadeiro***, o ***Falso*** e o ***Desconhecido***. Este último será o único que se trata de uma novidade, tendo em conta o trabalho desenvolvido na fase anterior, uma vez que, agora introduzimos o conceito de ***conhecimento imperfeito***. Este poderá ser definido através de três tipos: o ***Incerto***, o ***Impreciso*** e o ***Interdito***.

Descrição do Trabalho e Análise de Resultados

1. Alterações Realizadas

De modo a cumprir os requisitos desta fase, e melhorar algumas funcionalidades da fase anterior, foram realizadas várias alterações ao trabalho.

Nomeadamente, foram incluídos predicados representativos das várias fases de vacinação, com vista a tornar o sistema mais flexível no que se refere ao número de fases existente, nos requisitos de acesso específicos a cada uma (condições médicas, idade e profissão dos utentes) e no período de tempo que cada uma abrange.

Naturalmente isto levou a que se tornasse necessária a implementação de novos invariantes, responsáveis pela correta adição e remoção de fases da base de conhecimento. Estes invariantes definem então como requisitos para a adição de novas fases:

- Cada fase deverá ter um nome identificador próprio e único, de tipo *atom*;
- Cada fase deverá incluir uma data de início e de fim, sendo ambas as datas válidas e a data de fim posterior à data de início;
- A fase deverá incluir a idade mínima que um utente deve ter, de maneira a ser vacinado sem qualquer restrição;
- A fase inclui também um valor inteiro e uma lista de *atoms* que indicam a idade mínima e as doenças que o utente terá de ter, de forma a ser vacionado. Por exemplo, se a idade for 40 e a lista for [diabetes,colesterol], então todas as pessoas com idade superior ou igual a 40 que sofram de pelo menos uma dessas doenças, está apto a ser vacinado nesta fase.
- A idade mínima e a lista de doenças deverão estar agrupadas conjuntamente;
- A fase deverá incluir uma lista de profissões que tornem os utentes elegíveis para o seu acesso, sendo cada uma representada pelo tipo *atom*.

```
fase(Fase,DataI,DataF,Idade,(IdadeM,Doencas),Prof)
```

Figura 1-Formato da entidade fase

É de realçar que caso o utilizador do sistema pretenda definir uma fase em qualquer pessoa possa ser vacinada sem restrição de idade, basta colocar as variáveis *Idade* e *IdadeM* com valor zero. Por outro lado, se se pretende garantir que só pessoas com uma determinada idade mínima e que possuam determinadas doenças, deve-se preencher as variáveis *IdadeM* e *Doencas* com o que se deseja, e a variável Idade com um valor suficientemente alto, por exemplo 126. (consideramos neste trabalho que a idade máxima que um utente pode ter é 125 anos)

1.1 Invariantes adicionados

- ***Invariantes modificados da fase 1***

Uma alteração feita, foi nos invariantes para inserir um termo *vacinação_Covid*. Na fase 1, verificamos que havia um erro, pois esses invariantes falhavam sempre o teste. Isto ocorria porque eram feitas as seguintes verificações:

```
+vacinacao_Covid(IdStaff,IdUtente,Data,Vacina,Toma) :: (
        Toma == 2,
```

```
+vacinacao_Covid(IdStaff,IdUtente,Data,Vacina,Toma) :: (
        Toma == 1,
```

Figura 2 - Invariante vacinacao_Covid da 1ª Fase

Ora, isto é errado uma vez que uma vacina não pode ser segunda e a primeira toma em simultâneo. Daí a evolução retornar falso.

Assim, foram modificados os invariantes.

```
+vacinacao_Covid(IdStaff,IdUtente,Data,Vacina,2) :: (
        integer(IdStaff),
        integer(IdUtente),
        dataValida(Data),
        atom(Vacina),
        solucoes(IdStaff,staff(IdStaff,_,_,_),S),
        comprimento(S,1),
        solucoes(IdUtente,utente(IdUtente,_,_,_,_,_,_,_,_,_),S1),
        comprimento(S1,1),
        solucoes((IdStaff,IdUtente,Data,Vacina,2),vacinacao_Covid(IdStaff,IdUtente,Data,Vacina,2),S2),
        comprimento(S2,1),
        nao(naoRecebeuPrimeiraVacina(IdUtente)),
        solucoes((IdStaff,IdUtente,2),-vacinacao_Covid(IdStaff,IdUtente,_,_,2),SA),
        comprimento(SA,0)

    ).

+vacinacao_Covid(IdStaff,IdUtente,Data,Vacina,1) :: (
        integer(IdStaff),
        integer(IdUtente),
        dataValida(Data),
        atom(Vacina),
        solucoes(IdStaff,staff(IdStaff,_,_,_),S),
        comprimento(S,1),
        solucoes(IdUtente,utente(IdUtente,_,_,_,_,_,_,_,_,_),S1),
        comprimento(S1,1),
        solucoes((IdStaff,IdUtente,Data,Vacina,1),vacinacao_Covid(IdStaff,IdUtente,Data,Vacina,1),S2),
        comprimento(S2,1),
        solucoes((IdStaff,IdUtente,1),-vacinacao_Covid(IdStaff,IdUtente,_,_,1),SA),
        comprimento(SA,0)
        ).
```

Figura 3-Novos invariantes da vacinação

- ***Invariantes Gerais***

Como já foi referido anteriormente, neste trabalho é feita a introdução do conhecimento negativo e do imperfeito, o que leva a que seja necessário definir novos invariantes de maneira a garantir a consistência e integridade do nosso sistema.

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

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

Figura 4 - Invariante Termo

Visto que temos em consideração o conhecimento perfeito negativo e positivo, pretende-se evitar que se insira conhecimento positivo quando esse exato termo já existe sobre a forma de conhecimento negativo. Da mesma fora, não é permitida a inserção de conhecimento negativo quando já existe esse conhecimento sob a forma positiva.

```
+(excecao(Termo)) ::(
    solucoes(Termo, excecao(Termo), S),
    comprimento(S, 1)
).
```

Figura 5 - Invariante excecao

Relativamente ao conhecimento imperfeito, e de forma a tornam a sua manipulação mais fácil, foi criado o invariante da figura acima que impede a inserção de exceções repetidas.

- ***Invariantes relativos ao conhecimento perfeito negativo***

Quando estamos a lidar com informação que contenha conhecimento perfeito negativo, é necessário definir invariantes para a sua inserção. É de realçar, que estes invariantes garantem que os termos a adicionar possuem os valores dos campos com os devidos tipos, no entanto, visto que estamos perante a negação de factos, não é verificado se campos como o *id*, o número de segurança social e outros, são únicos. Além disto, é necessário garantir que não introduzimos informação repetida na base do conhecimento.

```
+(-utente(Id,SegurancaSocial,Nome,DataNasc,Email,Telefone,Morada,Profissao,ListaD,IdCentroSaude)) :: (
        integer(Id),
        integer(SegurancaSocial),
        integer(Telefone),
        integer(IdCentroSaude),
        dataValida(DataNasc),
        atom(Nome),
        atom(Email),
        atom(Morada),
        atom(Profissao),
        listaAtoms(ListaD),
        solucoes(Id, -utente(Id,SegurancaSocial,Nome,DataNasc,Email,Telefone,Morada,Profissao,ListaD,IdCentroSaude), S2),
        comprimento(S2,N2),
        N2>=0,
        N2=<1

).
```

Figura 6 - Invariantes utente

```
+(-centro_saude(Id,Nome,Morada,Telefone,Email)) :: (
        integer(Id),
        integer(Telefone),
        atom(Nome),
        atom(Morada),
        atom(Email),
        solucoes(Id, -centro_saude(Id,Nome,Morada,Telefone,Email), S2),
        comprimento(S2,N2),
        N2>=0,
        N2=<1
  ).
```

Figura 7 - Invariantes Centro_saude

```
+(-staff(IdStaff,IdCentroSaude,Nome,Email)) :: (
        integer(IdStaff),
        integer(IdCentroSaude),
        atom(Nome),
        atom(Email),
        solucoes(IdStaff, -staff(IdStaff,IdCentroSaude,Nome,Email), S2),
        comprimento(S2,N2),
        N2>=0,
        N2=<1
 ).
```

Figura 8 - Invariantes Staff

```
+(-vacinacao_Covid(IdStaff,IdUtente,Data,Vacina,Toma)) :: (
       integer(IdStaff),
       integer(IdUtente),
       dataValida(Data),
       atom(Vacina),
       integer(Toma),
       tomaValida(Toma),
       solucoes((IdStaff,IdUtente,Data,Vacina,Toma), -vacinacao_Covid(IdStaff,IdUtente,Data,Vacina,Toma), S2),
       comprimento(S2,N2),
       N2>=0,
       N2=<1
   ).
```

Figura 9 - Invariantes vacinacao_covid

- ***Invariantes relativos ao conhecimento imperfeito***

Neste trabalho, considera-se que apenas as entidades utente e staff possuem conhecimento interdito associado. Como será explicado num próximo tópico, o conhecimento interdito não pode ser evoluído pelo que, é preciso definir invariantes que impeçam tal ação.

```
+utente(Id,SegurancaSocial,Nome,DataNasc,Email,Telefone,Morada,Profissao,ListaD,IdCentroSaude) :: (
       solucoes(M, (utente(Id,SegurancaSocial,Nome,DataNasc,Email,Telefone,M,Profissao,ListaD,IdCentroSaude),nulo(M)), S3),
       comprimento(S3, N),
       N >= 0,
       N =< 1

 ).
```

Figura 10 - Invariantes utente para conhecimento positivo interdito

```
+staff(IdStaff,IdCentroSaude,Nome,Email) :: (
        solucoes(M,(staff(IdStaff,IdCentroSaude,Nome,M),nulo(M)),S2),
        comprimento(S2,N),
        N >= 0,
        N =< 1
).
```

Figura 11 - Invariantes Staff para conhecimento positivo interdito

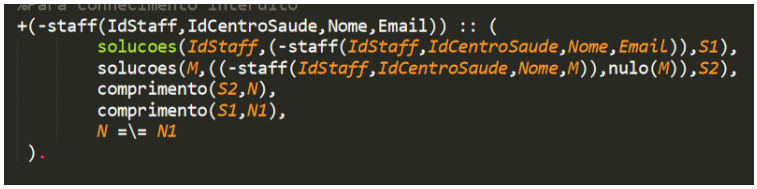

Figura 12 - Invariantes Staff para conhecimento negativo interdito

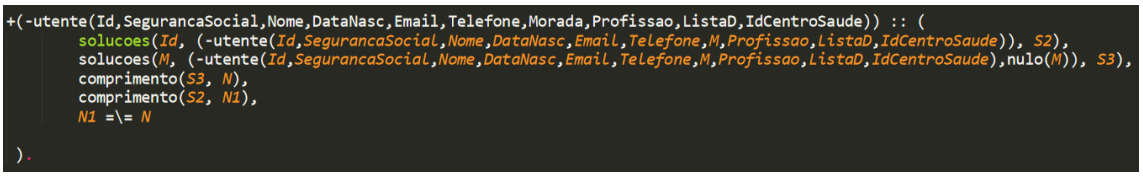

Figura 13 - Invariantes utente para conhecimento negativo interdito

No caso do conhecimento perfeito positivo, quando se tenta inserir um termo quando já existe conhecimento interdito a ele associado, a condição de unicidade de id referida anteriormente falha, falhando também a evolução. No entanto, no conhecimento negativo, não existe essa verificação de *ids*. Assim, os invariantes aqui referidos, garantem a consistência do sistema, pois, verificam se para um dado termo com determinados campos, apenas existe, no máximo uma vez, conhecimento interdito associado. Por exemplo, se tentarmos inserir o termo -staff(1,2,'Manel','manel@gmail.com'), só o poderemos fazer se não existir esse termo na base do conhecimento e se já não houver um termo igual a

-staff(1,2,'Manel',int01).

Por outro lado, se quisermos inserir o termo - ***staff(1,2,'Manel',int01***) só o poderemos fazer se já não existir esse termo na base de informação ou se não existir por exemplo, o termo

-staff(1,2,'Manel','manel@gmail.com').

Em ambas as situações a condição N/==N1 é a responsável por impedir tais contradições.

2. Conhecimento Perfeito

2.1 Conhecimento negativo e positivo

O conhecimento perfeito define-se como o conhecimento onde todos os seus campos deverão ser conhecidos e, ainda, encontrar-se dentro dos critérios e regras definidas. Este poderá ser dividido em dois tipos: Positivo e Negativo.

O conhecimento positivo é aquele que assinala a existência de um predicado, caso contrário, está-se perante conhecimento negativo. Todos os predicados que representam este tipo de conhecimento encontram-se assinalados com o seguinte prefixo "- ".

```
centro_saude(1,'Santa Maria','Rua da Estrela',253214998,'santamaria@gmail.com').
centro_saude(2,'Centro Regional de Monção','Rua da Estrela',253214998,'monçãoCS@gmail.com').
centro_saude(3,'Imaculado Cristo','Rua da Estrela',253214998,'iCristo@gmail.com').
centro_saude(4,'Centro Professor Doutor Afonso Henriques','Rua dos Mouros',252234595,'afonso_MataTudo@gmail.com').
centro_saude(5,'Estrela Brilhante','Rua da Estrela',251214990,'estrela_Brilhante@gmail.com').
```

Figura 14- Predicados de Inserção de Conhecimento Perfeito Positivo

```
-centro_saude(6,'São José','Rua do Pomar',253219983,'saoJose@gmail.com').
-centro_saude(7,'Lusíadas','Avenida dos Mouros',211765234,'lusiadas@gmail.com').
```

Figura 15- Predicados de Inserção de Conhecimento Perfeito Negativo

Portanto, sabemos que os centros de saúde representados na Figura 14 existem na nossa base de conhecimento, e aqueles que se encontram na Figura 15, não poderão existir dentro da mesma. Neste caso, poderemos concluir que os centros de saúde com os identificadores 6 e 7, e com as restantes características especificadas não existem.

Além disso, foi criado o predicado "perfeito" de forma a identificar os predicados de inserção que se caracterizam como conhecimento perfeito, tanto positivo como negativo. Deste modo, é possível identificar se um termo tem conhecimento perfeito ou não.

Na figura seguinte estão representados alguns exemplos ainda referentes aos predicados mostrados anteriormente relativos à entidade "Centro de Saúde"

```
perfeito(centro_saude(1)).
perfeito(centro_saude(2)).
perfeito(centro_saude(3)).
perfeito(centro_saude(4)).
perfeito(centro_saude(5)).
perfeito(centro_saude(6)).
perfeito(centro_saude(7)).
```

Figura 16 - Predicado Perfeito

2.2 Predicados de Negação Forte

Assim sendo, o conhecimento negativo por um lado tem a capacidade de retratar se a existência de um dado termo é admitida na base de conhecimento, e por outro de informar se um dado termo é falso. Por conseguinte, é de extrema importância, notar a distinção entre a utilização do predicado "não" e este tipo de conhecimento.

O predicado "não" tem como objetivo comunicar se existe ou não conhecimento sobre um determinado termo, portanto observando o seguinte exemplo, poderemos concluir que se trata de uma N***egação por Falha***. Isto acontece, uma vez que, tal como representado na Figura 15, o termo exemplificado não poderá ser admitido na Base de Conhecimento:

```
nao(centro_saude(6,'São José','Rua do Pomar',253219983,'saoJose@gmail.com')).
```

Figura 17- Exemplo predicado não

No caso do conhecimento negativo, este poderá ser representado por duas formas. Pela ***Negação Explícita***, que consiste naquela representada na Figura 15, onde se recorre à inserção na base de conhecimento do sistema através de predicados precedidos pelo caracter "- ", negado, desta forma, os mesmos. E, ainda, pela ***Negação Forte***.

Esta descende do ***Pressuposto do Mundo Fechado***, onde toda a informação que não existe ou não está mencionada na Base de Conhecimento é considerada falsa. Visto isto, foi necessário desenvolver e fazer uma extensão dos predicados já definidos que recorrem à Negação Explícita. Para tal desenvolveram-se os seguintes predicados:

```
-utente(Id,SegurancaSocial,Nome,DataNasc,Email,Telefone,Morada,Profissao,ListaD,IdCentroSaude) :-
    nao(utente(Id,SegurancaSocial,Nome,DataNasc,Email,Telefone,Morada,Profissao,ListaD,IdCentroSaude)),
    nao(excecao(utente(Id,SegurancaSocial,Nome,DataNasc,Email,Telefone,Morada,Profissao,ListaD,IdCentroSaude))).

-centro_saude(Id,Nome,Morada,Telefone,Email) :-
    nao(centro_saude(Id,Nome,Morada,Telefone,Email)),
    nao(excecao(centro_saude(Id,Nome,Morada,Telefone,Email))).

-staff(IdStaff,IdCentroSaude,Nome,Email) :-
    nao(staff(IdStaff,IdCentroSaude,Nome,Email)),
    nao(excecao(staff(IdStaff,IdCentroSaude,Nome,Email))).

-vacinacao_Covid(IdStaff,IdUtente,Data,Vacina,Toma) :-
    nao(vacinacao_Covid(IdStaff,IdUtente,Data,Vacina,Toma)),
    nao(excecao(vacinacao_Covid(IdStaff,IdUtente,Data,Vacina,Toma))).

-fase(Fase,DataI, DataF,Idade,(Idade1,Doencas),Prof) :-
    nao(fase(Fase,DataI, DataF,Idade,(Idade1,Doencas),Prof)),
    nao(excecao(fase(Fase,DataI, DataF,Idade,(Idade1,Doencas),Prof))).
```

Figura 18 - Predicados de Negação Forte

Sendo assim, tal como representado na Figura 18, o conhecimento será considerado como negativo caso não seja interpretado como conhecimento positivo, isto é: $nao(centro\_saude(Id, Nome, Morada, Telefone, Email))$.

E, ainda, caso não seja uma exceção.

3. Conhecimento Imperfeito

Como se pôde verificar na implementação da fase anterior, um programa em lógica determina respostas em termos da veracidade ou falsidade das questões, não sendo possível, assim, implementar um mecanismo de raciocínio que possibilite abordar a representação de informação incompleta.

Em sistemas construídos com base na forma de extensão à Programação Lógica, passamos a ter uma terceira hipótese de responder às questões: para além de serem verdadeiras as conclusões obtidas a partir da informação positiva e falsas as obtidas em função da informação negativa, torna-se possível concluir que a resposta é desconhecida, se nenhuma das conclusões anteriores for possível. Nestes casos, diz-se que estamos perante conhecimento imperfeito.

Assim sendo, nesta segunda fase foram incluídos na base de conhecimento os três tipos de conhecimento imperfeito abordados nas aulas: incerto, impreciso e interdito. Para o caso da nossa base de conhecimento, foi considerado apenas um campo desconhecido por entidade.

3.1 Conhecimento incerto

O tipo de conhecimento incerto representa situações em que não se conhece o valor de um determinado campo, isto é, parte da informação é desconhecida e está dentro de um conjunto ilimitado de hipóteses.

Na nossa base de conhecimento, para distinguir um campo incerto é utilizada a notação ‘inc01’. Por outro lado, para saber se um termo tem um campo incerto é utilizado o termo complexo ‘incerto’ que indica as entidades que possuem este tipo de conhecimento e o respetivo ID. Este termo revelou-se útil para posteriormente se poder obter uma listagem de todas as entidades que possuem este tipo de conhecimento.

Desta forma, para representar, por exemplo o centro de saúde ‘Santa Amora’ com id 8, número de telefone 111333444 e email 'amoraHospital@gmail.com' cuja morada é totalmente desconhecida foi adicionado à base de conhecimento o seguinte termo:

```
centro_saude(8,'Santa Amora',inc01,111333444,'amoraHospital@gmail.com').
```

Consequentemente, visto que foi introduzido um termo com conhecimento incerto é necessário introduzir também o termo complexo incerto correspondente ao seu ID. Uma vez que estamos perante conhecimento imperfeito, foi também necessário expandir o predicado dinâmico ‘excecao’. Este predicado devolve o valor *true* quando recebe termos que possuem campos desconhecidos.

```
incerto(centro_saude(8)).
excecao(centro_saude(Id,Nome,_,Telefone,Email)):-
      centro_saude(Id,Nome,inc01,Telefone,Email) .
```

Para finalizar, podemos comprovar pelo predicado demo, que o centro de saúde 'Santa Amora' possui uma morada incerta, uma vez que para qualquer valor que se atribua a este campo, é sempre indicado que estamos perante informação desconhecida.

```
?- demo(centro_saude(8,'Santa Amora','Rua da Vida',111333444,'amoraHospital@gmail.com'),X).
X = desconhecido.

?- demo(centro_saude(8,'Santa Amora','Rua do ceu',111333444,'amoraHospital@gmail.com'),X).
X = desconhecido.

?- demo(centro_saude(8,'Santa Amora','Rua da Morte',111333444,'amoraHospital@gmail.com'),X).
X = desconhecido.
```

Figura 19 – Output predicado demo

Na base de conhecimento implementada existe apenas um campo incerto por cada entidade excetuando a fase em que não se considerou nenhum. Pela tabela seguinte é possível identificar as informações que podem ser incertas de cada uma:

Entidade	Informação que pode ser incerta
Utente	Nome
Centro de saúde	Morada
Elemento do staff	Nome
Vacinação_covid	Nome da vacina
Fase	Nenhuma

3.2 Conhecimento impreciso

O tipo de conhecimento impreciso representa situações em que não se conhece o valor de um determinado campo, mas, ao contrário do conhecimento incerto, a parte da informação que é desconhecida está dentro de um conjunto limitado de hipóteses.

Para representar este tipo de conhecimento, uma vez que estamos perante um número finito de valores incertos foi utilizado o predicado 'excecao', ou seja, para cada entidade que possua este tipo de conhecimento o predicado indica todas as combinações possíveis de valores admissíveis. Por outro lado, para saber se um termo tem um campo impreciso é utilizado o termo complexo 'impreciso' que indica as entidades que possuem este tipo de conhecimento e o respetivo ID. Este termo revelou-se útil para posteriormente se poder obter uma listagem de todas as entidades que possuem este tipo de conhecimento.

Por exemplo, na nossa base de conhecimento para indicar que o centro de saúde 'Gil Vicente' pode ter o email 'gv@gmail.com' ou 'gilVicente@gmail.com' é utilizada a seguinte notação:

```
excecao(centro_saude(7,'Gil Vicente','Rua da vida',111777888,'gv@gmail.com')).
excecao(centro_saude(7,'Gil Vicente','Rua da vida',111777888,'gilVicente@gmail.com')).
impreciso(centro_saude(7)).
```

Como se pode verificar pela utilização do predicado demo, se questionarmos a base de conhecimento acerca do centro de saúde 'Gil Vicente', este deverá responder 'desconhecido' se no o email dado for 'gv@gmail.com' ou 'gilVicente@gmail.com'. Para qualquer outro email, a base de conhecimento deverá responder 'falso' uma vez que, neste caso, temos a certeza que que para o centro de saúde em questão o email só poderá ser um dos dois valores indicados pela 'excecao'.

```
?- demo(centro_saude(7,'Gil Vicente','Rua da vida',111777888,'gv@gmail.com'),X).
X = desconhecido.

?- demo(centro_saude(7,'Gil Vicente','Rua da vida',111777888,'gilVicente@gmail.com'),X).
X = desconhecido.

?- demo(centro_saude(7,'Gil Vicente','Rua da vida',111777888,'gil@gmail.com'),X).
X = falso .

?- demo(centro_saude(7,'Gil Vicente','Rua da vida',111777888,'gilV@gmail.com'),X).
X = falso .
```

Figura 20 - Output

Na base de conhecimento implementada existe apenas um campo impreciso por cada entidade. Pela tabela seguinte é possível identificar as informações que podem ser imprecisas de cada uma:

Entidade	Informação que pode ser imprecisa
Utente	Morada
Centro de saúde	Email
Elemento do staff	Email
Vacinação_covid	Data de vacinação
Fase	Idade mínima sem doenças

3.3 Conhecimento interdito

O tipo de conhecimento interdito representa situações em que existe informação desconhecida, mas, neste caso sabe-se que é impossível vir a conhecer tal informação. É de notar que por esse motivo este tipo de conhecimento nunca poderá sofrer evolução.

Na nossa base de conhecimento, para identificar um campo interdito de uma certa entidade é utilizada a notação 'int01'. Para além disso, foi também incluído o predicado 'nulo' que serve para identificar informação deste tipo. Tal como nos casos anteriores, para saber se um termo tem um campo interdito é utilizado o termo complexo 'interdito' que indica quais as entidades que possuem este tipo de conhecimento e o respetivo ID. Este termo revelou-se útil para posteriormente se poder obter uma listagem de todas as entidades que possuem este tipo de conhecimento.

De modo a inserir conhecimento imperfeito do tipo interdito, será então necessário marcar o campo correspondente com uma variável específica, inserir uma nova exceção correspondente a essa variável e declará-la como do tipo nulo:

```
excecao(staff(IdStaff,IdCentroSaude,Nome,_)) :- staff(IdStaff,IdCentroSaude,Nome,int01).
interdito(staff(12)).
nulo(int01).
```

Como se pode verificar pela utilização do predicado demo, se questionarmos a base de conhecimento acerca do elemento do staff 'Rosa' com o email 'rosa@gmail.com' este deverá responder 'desconhecido' uma vez que a base de conhecimento nunca terá acesso ao valor deste campo interdito. Ao contrário do conhecimento incerto verifica-se na figura abaixo que não é possível efetuar evolução deste tipo de conhecimento.

```
?- demo(staff(12,1,'Rosa','rosa@gmail.com'),X).
X = desconhecido.

?- evolucao(staff(12,1,'Rosa','rosa@gmail.com')).
false.
```

Figura 21 - Output

Na base de conhecimento implementada existe apenas um campo interdito para um utente e para um elemento do staff. Pela tabela seguinte é possível identificar as informações que podem ser interditas de cada entidade:

Entidade	Informação que pode ser interdita
Utente	Morada
Centro de saúde	Nenhuma
Elemento do staff	Email
Vacinação_covid	Nenhuma
Fase	Nenhuma

4. Remoção e adição de conhecimento

O sistema de conhecimento desenvolvido, permite a adição e a remoção de conhecimento, sendo que estas ações são designadas evolução e involução, respetivamente. No entanto, foi necessário modificar o sistema, de forma que este respondesse de um modo consistente à adição/remoção de conhecimento negativo e de conhecimento imperfeito, que não tinham sido introduzidos na fase 1. Esta consistência é garantida, essencialmente, pelos invariantes.

4.1 Evolução do sistema

4.1.1 Evolução de conhecimento perfeito

A evolução de conhecimento perfeito é feita utilizando dois predicados***: evolucao*** e ***evolucaoConhecimentoPerfeito***, sendo que, este último é o que deve ser invocado quando se pretende adicionar um termo (de conhecimento perfeito).

```
evolucaoConhecimentoPerfeito(Termo):- nao(imperfeito(Termo)),
                                      evolucao(Termo),
                                      addPerfeito(Termo).

evolucaoConhecimentoPerfeito(-Termo):- nao(imperfeito(Termo)),
                                       evolucao(-Termo),
                                       addPerfeito(Termo).


evolucaoConhecimentoPerfeito(Termo):- imperfeito(Termo),
                                      removeImperfeito(Termo),
                                      evolucao(Termo),
                                      addPerfeito(Termo).

evolucaoConhecimentoPerfeito(-Termo):- imperfeito(Termo),
                                       removeImperfeito(Termo),
                                       evolucao(-Termo),
                                       addPerfeito(Termo).
```

Figura 22 – Predicado evolucaoConhecimentoPerfeito

```
evolucao(Termo) :- solucoes(Invariante, +Termo::Invariante, Lista),
                   insere(Termo),
                   teste(Lista).



evolucao(-Termo) :- solucoes(Invariante, +(-Termo)::Invariante, Lista),
                    insere(-Termo),
                    teste(Lista).
```

Figura 23 - Predicado evolucao

O predicado ***evolucaoConhecimentoPerfeito,*** verifica se o termo que se pretende adicionar, já está na base de conhecimento com conhecimento imperfeito. Se não estiver, ou seja se o termo ***nao(imperfeito(Termo))*** devolver verdade, então o termo é inserido à base de conhecimento através do predicado ***evolução***. É de realçar que se houver conhecimento negativo que contradiga o termo a adicionar o teste dos invariantes irá falhar.

Caso já exista conhecimento imperfeito associado ao termo que pretendemos adicionar, é necessário evoluir esse conhecimento apenas se esse for incerto ou impreciso. Para isto, foi criado o predicado ***removeImperfeito***.

```
removeImperfeito(Termo):- removerImperfeitoIncerto(Termo).
removeImperfeito(Termo):- removerImperfeitoImpreciso(Termo).

removeImperfeito(-Termo):- removerImperfeitoImpreciso(-Termo).
removeImperfeito(-Termo):- removerImperfeitoImpreciso(Termo).
```

Figura 24 - Predicado removeImperfeito

Se o termo a inserir já possuir conhecimento imperfeito incerto, então é invocado o predicado ***removerImperfeitoIncerto***, caso tenha associado conhecimento imperfeito impreciso é chamado o predicado ***removerImperfeitoImpreciso***.

Iremos agora explicar como estes predicados funcionam quando o termo é um centro de saúde. Para as outras entidades, o raciocínio é exatamente o mesmo.

```
%centro_saude
removerImperfeitoIncerto(centro_saude(Id,_,_,_,_)):- incerto(centro_saude(Id)),
                                                     retract(incerto(centro_saude(Id))),
                                                     retract(centro_saude(Id,_,_,_,_)).


removerImperfeitoIncerto(-centro_saude(Id,_,_,_,_)):- incerto(centro_saude(Id)),
                                                      retract(incerto(centro_saude(Id))),
                                                      retract(-centro_saude(Id,_,_,_,_)).


removerImperfeitoImpreciso(centro_saude(Id,Nome,Morada,Telefone,Email)):- impreciso(centro_saude(Id)),
                                                                         excecao(centro_saude(Id,Nome,Morada,Telefone,Email)),
                                                                         removeExcecoesImpreciso(centro_saude(Id,Nome,Morada,Telefone,Email)).


removeExcecoesImpreciso(centro_saude(Id,Nome,Morada,Telefone,_)):- solucoes(excecao(centro_saude(_,_,_,_,_)),excecao(centro_saude(Id,Nome,Morada,Telefone,_)),S),
                                                                  retiraTudo(S),
                                                                  retract(impreciso(centro_saude(Id))).
```

Figura 25 - Predicado removerImperfeitoIncerto e removerImperfeitoImpreciso

O predicado ***removerImperfeitoIncerto***, primeiro verifica se o centro de saúde com um dado *id*, possui conhecimento incerto associado. Se tiver, então é removido da base de conhecimento o termo ***incerto(centro_saude(Id))***, que, como já foi referido, usa-se para se tornam mais fácil saber se uma entidade possui conhecimento incerto associado a si. Posto isto, é removido também o centro de saúde com aquele id, para que se possa inserir o termo desejado (com conhecimento perfeito) sem que haja conflitos com os *ids*.

O predicado ***removerImperfeitoImpreciso,*** primeiro verifica se o centro de saúde com um dado *id*, já possui conhecimento impreciso associado na base de informação. Se tiver, é necessário garantir que o centro de saúde que queremos inserir tem um campo email *(*campo desconhecido*)* válido. Para fazer esta verificação basta ver se o termo a adicionar respeita alguma exceção. Como já foi dito, consideramos, neste trabalho, que um termo só pode ter, no máximo, um campo desconhecido.

Ora se o centro de saúde já possuir conhecimento impreciso, então irá falhar nas exceções relativas ao conhecimento incerto e interdito, e só falha nas exceções relativas ao conhecimento impreciso se o campo *email* não é nenhuma das opções definidas. Posto isto, falta retirar todas as exceções definidas, através do predicado ***removerExcecoesImpreciso***. Este predicado recorre ao ***solucoes*** para colocar em S todas as exceções da forma ***excecao(centro_saude(Id,Nome,Morada,Telefone,_))***, sendo que S será argumento do predicado ***retiraTudo*** que remove da base de conhecimento tudo que está nessa lista S. Após isto, remove-se o termo ***impreciso(centro_saude(Id)).***

Voltando ao predicado ***evolucaoConhecimentoPerfeito,*** após ser feita a evolução do termo, adiciona-se o termo ***perfeito(centro_saude(Id))***.

```
addPerfeito(centro_saude(Id,_,_,_,_)):- insere(perfeito(centro_saude(Id))).
```

Figura 26 - Predicado addPerfeito

É de realçar que, neste trabalho, quando tentamos introduzir um termo do qual já existe conhecimento interdito, o predicado ***removeImperfeito*** devolve falso, impedindo assim a sua evolução. No entanto, uma alternativa possível, seria deixar para os invariantes essa verificação.

Iremos agora mostrar alguns exemplos que verificam tudo o que foi explicado neste tópico.

```
?- evolucaoConhecimentoPerfeito(centro_saude(1,'Santa Maria','Rua da Estrela',253214998,'santamaria@gmail.com')).
false.
```

Figura 27 - Resultado de tentar adicionar conhecimento perfeito repetido

```
?- listing(incerto).
:- dynamic incerto/1.

incerto(utente(17)).
incerto(centro_saude(8)).
incerto(staff(11)).
incerto(vacinacao_Covid(3, 8, 2)).

true.
```

Figura 28 - Centro de saude com id 8 possui conhecimento incerto

```
?- evolucaoConhecimentoPerfeito(centro_saude(8,'Santa Amora','Rua Wally',111333444,'amoraHospital@gmail.com')).
true .
```

Figura 29 - Evolução do termo centro_saude com id 8. A rua é agora conhecida

```
?- listing(incerto).
:- dynamic incerto/1.

incerto(utente(17)).
incerto(staff(11)).
incerto(vacinacao_Covid(3, 8, 2)).
```

Figura 30 - O centro de saúde com id 8 deixou de ter conhecimento incerto associado

```
?- listing(perfeito).
:- dynamic perfeito/1.

perfeito(fase('1B')).
perfeito(fase('2')).
perfeito(fase('3')).
perfeito(fase('4')).
perfeito(centro_saude(8)).
```

Figura 31 - O centro de saúde passa a ter todos os campos conhecidos

uando o predicado ***evolucaoConhecimentoPerfeito*** é invocado, prossupõem-se que o termo a adicionar possui todos os campos conhecidos. Se, por exemplo, se tentar inserir um centro de saúde cujo campo nome é *inc01*, o predicado vai funcionar, pois encara o *inc01* como sendo o nome da entidade e não um campo desconhecido.

4.1.2 Evolução de conhecimento imperfeito

A evolução de conhecimento imperfeito é feita utilizando 3 predicados***: evolucao_Incerto***, ***evolucao_Impreciso***, ***evolucao_Interdito***.

- **Evolução do conhecimento Incerto**

A evolução do conhecimento incerto é feita recorrendo ao predicado ***evolucaoIncerto***.

```
%staff
evolucaoIncerto(staff(Id,IdCentroSaude,Nome,Email)):- nao(impreciso(staff(Id))),
                                                      evolucao(staff(Id,IdCentroSaude,Nome,Email)),
                                                      insere(incerto(staff(Id))),
                                                      checkExcecao(staff(Id,IdCentroSaude,Nome,Email)).

evolucaoIncerto(-staff(Id,IdCentroSaude,Nome,Email)):- nao(impreciso(staff(Id))),
                                                      evolucao(-staff(Id,IdCentroSaude,Nome,Email)),
                                                      insere(incerto(staff(Id))),
                                                      checkExcecao(staff(Id,IdCentroSaude,Nome,Email)).
```

Figura 32 - Predicado evolucaoIncerto

Utilizemos, para a explicação, o staff como exemplo.

Quando queremos adicionar um membro do staff, que contém um campo incerto, temos que ter algumas atenções. Primeiro, verificamos se o termo a adicionar já existe com conhecimento impreciso associado. Se sim, então a inserção desse termo falha (pois não se pode substituir o conhecimento impreciso pelo incerto), caso contrário faz-se a evolução. Quanto se tenta fazer a evolução deste staff, se já existir conhecimento interdito, incerto ou perfeito associado a ele, então, o teste dos invariantes falha pois não se pode adicionar um termo com um *Id* que já exista na base de informação. Posto isto, e sendo possível inserir o staff com conhecimento incerto, adiciona-se o termo ***incerto(staff(Id))***. Uma vez que no caso do conhecimento incerto, basta apenas definir uma exceção geral, foi definido o predicado ***checkExcecao*** que verifica se essa excecao geral existe. Se existir não faz nada, caso contrário, adiciona-a.

```
checkExcecao(staff(Id,IdCentroSaude,Nome,Email)):- excecao(staff(Id,IdCentroSaude,Nome,Email)).

checkExcecao(staff(_,_,_,_)):- insere(excecao(centro_saude(I,N,M,T,E)):-centro_saude(I,N,inc01,T,E)).
```

Figura 33 - Predicado checkExcecao

É de realçar que este raciocínio é idêntico para as outras entidades do sistema, exceto para a fase que exigiu uma ligeira diferença. Neste trabalho, consideramos que uma fase com conhecimento incerto tem como campo desconhecido a idade mínima de vacinação sem restrições de doenças, e sendo esta do tipo *integer*, ao invocarmos o predicado ***evolução*** a inserção falhava sempre. Isto porque, quando o campo contem a expressão int01 (que é do tipo *atom* e simboliza que o campo é desconhecido do tipo incerto), ao fazermos ***integer(Idade)*** dá falso. Desta maneira, criou-se um predicado ***evolucaoIncertoFaseAux***, que apenas invoca um invariante que não faz a verificação da idade ser um valor inteiro.

```
%fase-ligeiramente diferente por causa dos inteiros
evolucaoIncerto(fase(Fase,DataI,DataF,Idade,(IdadeM,Doencas),Prof)):- nao(impreciso(fase(Fase))),
                                                                      evolucaoIncertoFaseAux(fase(Fase,DataI,DataF,Idade,(IdadeM,Doencas),Prof)),
                                                                      insere(incerto(fase(Fase))),
                                                                      checkExcecao(fase(Fase,DataI,DataF,Idade,(IdadeM,Doencas),Prof)).

evolucaoIncerto(-fase(Fase,DataI,DataF,Idade,(IdadeM,Doencas),Prof)):- nao(impreciso(fase(Fase))),
                                                                       evolucaoIncertoFaseAux(-fase(Fase,DataI,DataF,Idade,(IdadeM,Doencas),Prof))
                                                                       insere(incerto(fase(Fase))),
                                                                       checkExcecao(fase(Fase,DataI,DataF,Idade,(IdadeM,Doencas),Prof)).


evolucaoIncertoFaseAux(Termo) :- solucoes(Invariante, +Termo::Invariante, Lista),
                                 insere(Termo),
                                 teste(Lista).

evolucaoIncertoFaseAux(-Termo) :- solucoes(Invariante, +(-Termo)::Invariante, Lista),
                                  insere(Termo),
                                  teste(Lista).
```

Figura 34 - Predicado evolucaoIncertoFaseAux e evolucaoIncerto

```
+fase(Fase,DataI,DataF,Idade,(_,Doencas),Prof)::(
    atom(Fase),
    dataValida(DataI),
    dataValida(DataF),
    dataAntesIgual(DataI,DataF),
    integer(Idade),
    listaAtoms(Doencas),
    listaAtoms(Prof),
    solucoes(Fase,fase(Fase,_,_,_,(_,_),_),S),
    comprimento(S,1)
).
```

Figura 35 - Invariante Fase (conhecimento positivo)

```
+(-fase(Fase,DataI,DataF,Idade,(IdadeM,Doencas),Prof))::(
    atom(Fase),
    dataValida(DataI),
    dataValida(DataF),
    dataAntesIgual(DataI,DataF),
    integer(Idade),
    listaAtoms(Doencas),
    listaAtoms(Prof),
    solucoes(Fase ,-fase(Fase,DataI,DataF,Idade,(IdadeM,Doencas),Prof), S2),
    comprimento(S2,N2),
     N2>=0,
     N2=<1

    ).
```

Figura 36 - Invariante Fase (conhecimento negativo)

- **Evolução do conhecimento Impreciso**

A evolução do conhecimento impreciso é feita recorrendo aos predicados ***evolução_Impreciso*** e ***evolucaoImpreciso***.

```
evolucao_Impreciso(Termo):- evolucaoImpreciso(Termo).
evolucao_Impreciso(-Termo):- evolucaoImpreciso(Termo).
```

Figura 37 - Predicado evolucao_Impreciso

```
evolucaoImpreciso(staff(Id,IdCentroSaude,Nome,Email)):- removerImperfeitoIncerto(staff(Id,IdCentroSaude,Nome,Email)),
                                                        insere(impreciso(staff(Id))),
                                                        insere(excecao(staff(Id,IdCentroSaude,Nome,Email))).

evolucaoImpreciso(staff(Id,IdCentroSaude,Nome,Email)):- nao(perfeito(staff(Id))),
                                                        nao(interdito(staff(Id))),
                                                        verificaQueExiste(staff(Id)),
                                                        evolucao(excecao(staff(Id,IdCentroSaude,Nome,Email))).
```

Figura 38 - Predicado evolucaoImpreciso

Utilizemos, para a explicação, novamente, o staff.

Em primeiro lugar, verificamos se já existe na base de conhecimento esse membro do staff com conhecimento incerto. Se existir, é necessário substituir o conhecimento incerto pelo impreciso. Ora, começamos por usar o predicado ***removerImperfeitoIncerto***, que como já foi referido neste relatório, retira o staff incerto em questão e o termo ***incerto(staff(id)).*** Por fim, insere-se o termo ***impreciso(staff(Id))*** e adiciona-se a exceção ***excecao(staff(Id,IdCentroSaude,Nome,Email))***.

No entanto, caso o membro do staff não tenha conhecimento incerto, entramos na segunda definição do predicado ***evolucaoImpreciso***. Aqui, temos de ter em conta que o termo não pode ser inserido se já existir conhecimento perfeito ou interdito sobre ele. Além disto, como no conhecimento impreciso se podem inserir diversas entidades com os mesmos parâmetros diferindo apenas no parâmetro que é desconhecido, temos de verificar se o termo a inserir é o primeiro impreciso. Se for, então adiciona-se o termo ***impreciso(staff(Id)),***caso contrário, não é necessário***.*** Por fim, efetua-se a evolução da exceção, que falha se esta for repetida(esta falha é provocada pelos invariantes).

```
verificaQueExiste(staff(Id)):- nao(impreciso(staff(Id))),
                               insere(impreciso(staff(Id))).

verificaQueExiste(staff(_)).
```

Figura 39 - Predicado verificaQueExiste

- **Evolução do conhecimento Interdito**

A evolução do conhecimento impreciso é feita recorrendo ao predicado ***evolucaoInterdito***.

```
evolucaoInterdito(staff(Id,IdCentroSaude,Nome,Email)):- nao(impreciso(staff(Id))),
                                                        evolucao(staff(Id,IdCentroSaude,Nome,Email)),
                                                        insere(interdito(staff(Id))),
                                                        checkExcecaoInterdito(staff(Id,IdCentroSaude,Nome,Email)).

evolucaoInterdito(-staff(Id,IdCentroSaude,Nome,Email)):- nao(impreciso(staff(Id))),
                                                         evolucao(-staff(Id,IdCentroSaude,Nome,Email)),
                                                         insere(interdito(staff(Id))),
                                                         checkExcecaoInterdito(staff(Id,IdCentroSaude,Nome,Email)).
```

Figura 40 - Predicado evolucaoInterdito

Utilizemos, para a explicação, novamente, o staff.

Primeiramente, só podemos inserir um membro do staff com conhecimento interdito se não existir conhecimento perfeito, interdito, impreciso ou incerto relativo a esse termo. Para verificar isto, primeiro invocamos o predicado ***impreciso***, que caso dê falso, a sua negação dá verdadeira. De seguida, tentamos fazer a evolução do termo. Se já existir conhecimento interdito, perfeito ou incerto, então esse termo staff já está inserido na base de informação e por isso a evolução falha, pois não é permitida a adição de *ids* repetidos.

Por último, inserimos o termo ***interdito(staff(Id))***, e assim como acontecia no ***checkExcecao*** referido na evolução do conhecimento incerto, usamos o predicado ***checkExcecaoInterdito,*** para inserir a exceção referente ao conhecimento interdito do termo staff, caso esta não exista.

```
checkExcecaoInterdito(staff(Id,IdCentroSaude,Nome,Email)):- excecao(staff(Id,IdCentroSaude,Nome,Email)).

checkExcecaoInterdito(staff(_,_,_,_)):- insere(excecao(centro_saude(I,N,M,T,E)):-centro_saude(I,N,M,T,int01)).
```

Figura 41 - Predicado checkExcecaoInterdito

4.2 Involução do sistema

4.2.1 Involução de conhecimento perfeito

A remoção do conhecimento perfeito é feita utilizando os predicados ***involucaoPerfeito***, e ***removerPerfeito***.

```
involucaoPerfeito(Termo):- removerPerfeito(Termo).
involucaoPerfeito(-Termo):- removerPerfeito(-Termo).
```

Figura 42 - Predicado involucaoPerfeito

Utilizemos como exemplo, o centro de saúde.

```
removerPerfeito(centro_saude(Id,Nome,Morada,Telefone,Email)):- perfeito(centro_saude(Id)),
                                                               retract(perfeito(centro_saude(Id))),
                                                               retract(centro_saude(Id,Nome,Morada,Telefone,Email)).

removerPerfeito(-centro_saude(Id,Nome,Morada,Telefone,Email)):- perfeito(centro_saude(Id)),
                                                               retract(perfeito(centro_saude(Id))),
                                                               retract(-centro_saude(Id,Nome,Morada,Telefone,Email)).
```

Figura 43 - Predicado removerPerfeito

Primeiramente, verificamos se o termo que queremos remover está na base de conhecimento e se possui conhecimento perfeito. Caso isto seja falso, então a involução falha, caso contrário, é retirado o termo que indica que o centro de saúde tem conhecimento perfeito bem como o próprio termo centro_saude.

Este raciocínio é o mesmo para todas as outras entidades do sistema.

4.2.1 Involução de conhecimento imperfeito

- **Involução de conhecimento incerto**

A involução do conhecimento incerto é feita utilizando o predicado ***involucaoIncerto***.

```
involucaoIncerto(Termo):- removerImperfeitoIncerto(Termo).
involucaoIncerto(-Termo):- removerImperfeitoIncerto(-Termo).
```

Figura 44 - Predicado involucaoIncerto

Como podemos verificar, esta involução é bastante simples, na medida em que basta reaproveitar o predicado ***removerImperfeitoIncerto***, que remove a entidade passada como argumento, bem como os termos que indicam que essa entidade possui conhecimento incerto.

- **Involução de conhecimento impreciso**

A involução do conhecimento incerto é feita utilizando o predicado ***involucaoImpreciso*** e ***involucaoImprecisoAux***.

```
involucaoImpreciso(Termo):- involucaoImprecisoAux(Termo).
involucaoImpreciso(-Termo):- involucaoImprecisoAux(Termo).
```

Figura 45 - Predicado involucaoImpreciso

```
involucaoImprecisoAux(centro_saude(Id,Nome,Morada,Telefone,Email)):- solucoes(excecao(centro_saude(_,_,_,_,_)),excecao(centro_saude(Id,Nome,Morada,Telefone,_)),S),
                                                                    comprimento(S,1),
                                                                    retract(excecao(centro_saude(Id,Nome,Morada,Telefone,Email))),
                                                                    retract(impreciso(centro_saude(Id))).
```

Figura 46 - involucaoImprecisoAux

Utilizemos novamente, como exemplo, o centro de saúde.

Como já foi referido, quando temos conhecimento imperfeito do tipo impreciso associado a uma entidade, apenas é adicionada base de informação uma exceção, ao contrário dos outros tipo de conhecimento, nos quais se adiciona também um termo. Desta forma, quando tentamos reduzir o conhecimento relativo a um centro de saúde com conhecimento impreciso, aquilo que se faz é retirar a exceção associada a esse mesmo termo.

Primeiramente, recorre-se ao predicado ***solucoes*** para colocar em S, todas as exceções (de conhecimento impreciso) associadas ao centro de saúde em questão. Se apenas houver uma exceção, significa que o que vamos remover é ultima exceção e por isso, temos de retirar também o termo ***impreciso(centro_saude(Id)).*** Por outro lado, se existir mais do que uma exceção do conhecimento impreciso, a única coisa a fazer é retirar aquele que contem o campo desconhido igual ao passado como argumento do predicado ***involucaoImpreciso***.

```
involucaoImprecisoAux(Termo):- retract(excecao(Termo)).
```

Figura 47 - Predicado involucaoImprecisoAux

- **Involução de conhecimento interdito**

A involução do conhecimento incerto é feita utilizando os predicados ***involucaoIncerto*** e ***involucaoInterditoAux***.

```
involucaoInterdito(Termo):- involucaoInterditoAux(Termo).
involucaoInterdito(-Termo):- involucaoInterditoAux(-Termo).
```

Figura 48 - Predicado involucaoInterdito

Utilizemos o staff para exemplicar.

```
involucaoInterditoAux(staff(IdStaff,IdCentroSaude,Nome,Email)):- retract(interdito(staff(IdStaff))),
                                                                  retract(staff(IdStaff,IdCentroSaude,Nome,Email)).

involucaoInterditoAux(-staff(IdStaff,IdCentroSaude,Nome,Email)):- retract(interdito(staff(IdStaff))),
                                                                   retract(-staff(IdStaff,IdCentroSaude,Nome,Email)).
```

Figura 49 - Predicado involucaoInterditoAux

A involução deste tipo de conhecimento é bastante simples. Primeiro é necessário remover o termo que indica que o membro do staff em questão tem conhecimento interdito, sendo para isto usado o predicado ***retract***. É de realçar que caso não exista na base do conhecimento o termo ***interdito(staff(Id))*** (ou seja o termo não é interdito), o ***retract*** falha e consequentemente a involução também falha. Posto isto, resta apenas retirar a entidade staff da base do conhecimento.

5. Sistema de inferência

Visto que estamos perante um caso de programação de lógica estendida, torna-se essencial criar um sistema de inferência capaz de lidar com os três valores lógicos abordados: desconhecido, verdadeiro e falso.

Desta forma, foi criado o predicado ***demo***, que ao contrário daquele que foi apresentado na fase 1, já tem em conta o desconhecido.

Este predicado, devolve verdadeiro caso o conhecimento esteja explicito na base de informação, devolve falso se a negação da questão existe através do predicado de negação forte e devolve desconhecido se existir conhecimento imperfeito associado à questão.

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

Figura 50 - Predicado demo

Conclusão

Dada como finalizada esta fase, consideramos que o trabalho desenvolvido cumpre as expectativas, tendo todas as funcionalidades pedidas implementadas e sendo capaz de devolver as soluções esperadas em todos os casos. A elaboração deste trabalho permitiu também a consolidação dos conhecimentos adquiridos nas aulas da unidade curricular, em particular no que se refere ao tema do conhecimento imperfeito, e levou ao desenvolvimento de uma maior competência no que se refere à programação em lógica.

Como sugestão para trabalho futuro, julgamos ser possível e pertinente a implementação de mais funcionalidades, dada a diversidade e flexibilidade do tema do trabalho proposto.

Bibliografia

Bratko, I. (1986). *Prolog: Programming for Artificial Intelligence.*

C. A., & J. N. (2016). Documento Pedagógico. *Representação de Informação Incompleta* .

Vacinação Covid-19. (s.d.). Obtido de covid19 - DGS: https://covid19.min-saude.pt/vacinacao/