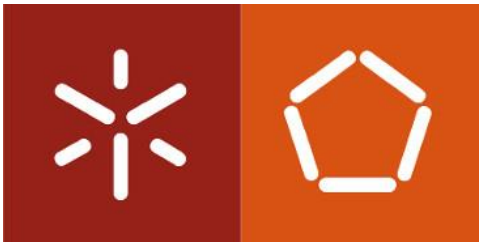
UNIVERSIDADE DO MINHO



Programação em Lógica

Mestrado Integrado em Engenharia Biomédica

Inteligência Artificial em Engenharia Biomédica

(1º Semestre/ Ano Letivo 2021/2022)

Grupo 10

Lara Alexandra Pereira Novo Martins Vaz (A88362)

Mariana Lindo Carvalho (A88360)

Tiago Miguel Parente Novais (A88397)

Braga

3 de dezembro de 2021

Resumo

O principal objetivo do presente trabalho é a utilização da 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 à criação de mecanismos de raciocínio adequados. Para tal, foi desenvolvido um sistema de representação de conhecimento e raciocínio com a capacidade de caracterizar um universo de discurso na área da prestação de cuidados de saúde.

Primeiramente, serão descritas as metodologias aplicadas na construção da base de conhecimento que permitiu a representação do caso em estudo. Deste modo, na base de conhecimento encontram-se disponíveis informações relativas aos utentes, os seus prestadores e os atos realizados. Com o intuito de manipular de forma segura esta mesma base de conhecimento foram desenvolvidos invariantes.

Para a confirmação dos resultados pretendidos recorreu-se ao interpretador de PROLOG SWI, confirmando que todos os predicados e invariantes funcionavam conforme esperado.

Índice

1.Introdução

Este trabalho tem como principal objetivo a utilização da extensão à programação em lógica, tendo como base a linguagem de programação em lógica PROLOG, no âmbito da representação de conhecimento imperfeito. Para isso foram utilizados valores nulos e criados sistemas de inferência apropriados.

Assim, foi desenvolvido um sistema de representação de conhecimento e raciocínio com o objetivo de caraterizar um universo de discurso na área de prestação de cuidados de saúde. Esta caracterização foi baseada em utentes, atos e prestadores.

Tendo como objetivo contruir este sistema, foram necessárias algumas funcionalidades:

- representação de conhecimento positivo e negativo;
- representação casos de conhecimento imperfeito, utilizando os valores nulos estudados;
- representação de invariantes que impõem restrições à inserção e à remoção de conhecimento do sistema;
- resolução da problemática resultante da evolução do conhecimento, criando os procedimentos apropriados;
- desenvolvimento de um sistema de inferência com a capacidade de implementar os mecanismos de raciocínio associados a estes sistemas.

Para assegurar a consistência da base de conhecimento e de forma a não existir informação repetida, foram criados invariantes, que apresentam restrições à manipulação da base de conhecimento, tornando possível fazê-lo de forma segura.

2.Preliminares

De modo a construir um sistema de representação de conhecimento imperfeito, através da extensão à programação em lógica, houve a necessidade de introduzir novos conceitos, referenciados de seguida.

2.1. Pressupostos da Programação em Lógica

As linguagens de manipulação de informação num sistema de bases de dados, têm como base os seguintes pressupostos [1]:

Pressuposto Mundo Fechado – toda a informação que não seja mencionada na base de conhecimento é dada como falsa.

Pressuposto dos Nomes Únicos – duas constantes diferentes (que definam valores atómicos ou objetos) correspondem, obrigatoriamente, duas entidades diferentes do universo de discurso.

Pressuposto do Domínio Fechado - além dos objetos designados por constantes na base de dados, não existem mais no universo de discurso.

Contudo, estes pressupostos não podem ser aplicados a uma base de conhecimento, pois, nem sempre se pretende assumir que a informação representada é a única que é válida e que as entidades apresentadas sejam as únicas existentes no mundo exterior. Deste modo, para as linguagens de manipulação de informação num sistema de bases de conhecimento, como por exemplo, a linguagem de programação PROLOG, têm como base os seguintes pressupostos [1]:

Pressuposto Mundo Aberto – para além dos factos ou conclusões representadas na base de conhecimento, podem existir outros que sejam verdadeiros.

Pressuposto dos Nomes Únicos – duas constantes diferentes (que definam valores atómicos ou objetos) correspondem, obrigatoriamente, duas entidades diferentes do universo de discurso.

Pressuposto Domínio Aberto – podem existir mais objetos no universo de discurso para além dos designados por constantes na base de conhecimento.

Tendo em conta os objetivos do presente trabalho, foram adotados os pressupostos utilizados nas bases de conhecimento. Num programa em lógica, as respostas às questões colocadas são, apenas, de dois tipos: verdadeiro ou falso.

Assim, a generalidade dos programas escritos em lógica representa implicitamente a informação negativa, assumindo a aplicação do raciocínio segundo o PMF. Uma extensão de um programa em lógica pode incluir informação negativa explicitamente, bem como explicitar diretamente o PMF para alguns predicados. Consequentemente, 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 pressupostos [1].

2.2. Extensão à Programação em Lógica

Um programa em lógica tem a capacidade de determinar respostas em termos da veracidade ou falsidade do que é questionado, não tendo a possibilidade de abordar a representação de informação incompleta [1].

De forma a ser possível representar informação negativa explicitamente, utiliza-se a extensão à programação em lógica.

Assim, são necessários dois predicados distintos que representam dois tipos de negação. O primeiro predicado é o *nao* que representa a negação por falha. O outro é o "-" que identifica a informação negativa ou falsa, sendo assim uma negação forte.

2.3. Valores nulos

Perante situações nas quais existe uma representação parcial de informação, surge a necessidade de implementar valores nulos que tornam possível a distinção entre situações em que as respostas a questões devem ser dadas como conhecidas, isto é, verdadeiras ou falsas, ou desconhecidas [1].

Os diferentes valores nulos abordados neste trabalho são os seguintes:

• Valor nulo incerto: representação de valores desconhecidos, mas não necessariamente de um conjunto determinado de valores, ou seja, são desconhecidos genericamente;

• Valor nulo impreciso: representação de valores desconhecidos, mas dentro de um conjunto de valores;

• Valor nulo interdito: representação de valores desconhecidos com interdição da evolução dos mesmos na base de conhecimento, isto é, não é permitido conhecer.

3. Descrição do Trabalho e Análise de Resultados

3.1. Declarações Iniciais

De forma a permitir ao utilizador um maior controlo da execução dos programas, foram implementadas na base de conhecimento declarações iniciais. Estas declarações iniciais, denominadas *flags,* são definidas por um determinado valor, que pode ser alterado consoante as necessidades do utilizador.

Para o caso proposto, implementaram-se as seguintes *flags*:

```
% SWI PROLOG: Declarações iniciais

:- set_prolog_flag( discontiguous_warnings,off ).
:- set_prolog_flag( single_var_warnings,off ).
:- set_prolog_flag( unknown,fail ).
```

Nas duas primeiras declarações, é-lhes atribuído o valor *off,* sendo que na primeira inibe-se o aparecimento de avisos quando cláusulas de um determinado predicado não estão juntas, e na segunda são inibidos os avisos relativos a variáveis únicas. Na última declaração é atribuído o valor *fail*, que apresenta falha quando se chamam predicados que não estão definidos.

3.2. Definições Iniciais

Por defeito, o PROLOG define os predicados como estáticos. Para ser possível introduzir ou remover conhecimento na base, é necessário que seja feita a conversão dos predicados para dinâmicos. Para isso, utilizaram-se as definições iniciais, capazes de realizar esta conversão. Para o trabalho proposto, implementaram-se as seguintes definições iniciais:

```
% SWI PROLOG: Definições iniciais
:- op( 900, xfy, '::' ).
:- op( 300, xfy, ou ).
:- op( 300, xfy, e ).
```

```
:- dynamic(utente/5).
:- dynamic(ato/6).
:- dynamic(prestador/7).
:- dynamic(excecao/1).
:- dynamic('-'/1).
:- dynamic('::'/2).
```

O operador (`'::'`) está relacionado com as invariantes e os operadores (`e`) e (`ou`) estão relacionados com o sistema de inferência desenvolvido no presente trabalho prático, permitindo a representação da conjunção e disjunção de termos.

Relativamente os predicados que vão representar o conhecimento (*utente*, *prestador* e *ato*), estes foram definidos como dinâmicos. Foi também indicado o número de argumentos para cada predicado, sendo 5 para o *utente*, 7 para o *prestador* e 6 para o predicado *ato*.

O predicado *excecao* foi também definido como dinâmico, tendo sido criado para auxiliar na representação de conhecimento imperfeito.

A notação `'-'` foi definida como dinâmica, permitindo a representação de conhecimento negativo.

Por último, a notação `'::'` foi também definida como dinâmica, permitindo assim a introdução de exceções na base de conhecimento.

3.3. Representação de Conhecimento Perfeito Positivo e Negativo

Primeiramente, houve a necessidade de construir uma base de conhecimento que incluísse conhecimento perfeito, dentro do qual se inclui o conhecimento positivo e negativo relativo aos utentes, prestadores e atos. Estes predicados serão apresentados ao longo desta secção.

3.3.1. Predicado *utente*

O predicado *utente* tem como atributos o ID do utente, o nome, a idade, a cidade e o sexo. De seguida, serão apresentados três exemplos do predicado *utente*. É de realçar que os restantes factos se encontram no Anexo A.

```
utente(1001, laravaz, 22, braga, feminino).
utente(1002, marianalindo, 20, vianadocastelo, feminino).
utente(1003, tiagonovais, 21, fafe, masculino).
```

De forma a representar o conhecimento perfeito negativo relativamente ao predicado *utente*, definiu-se o predicado *–utente*, onde IU é o ID do utente, N o nome do utente, I a idade, C a cidade de residência e por último S o sexo por o qual se identifica.

```
-utente(IU,N,I,C,S) :-
          nao(utente(IU,N,I,C,S)),
          nao(excecao(utente(IU,N,I,C,S))).
-utente(1015,josemaria,45,lisboa,masculino).
```

O predicado acima descrito permite definir que os factos que não estejam representados na base de conhecimento como conhecimento perfeito positivo ou imperfeito, são dados como negativos. Desta forma, quando este é testado utilizando o meta-predicado *si*, que será apresentado posteriormente na secção 3.6.1, resulta em falsidade.

Tendo como objetivo a representação de uma base de conhecimento e não tanto a proximidade à realidade, foi inserido um exemplo de conhecimento negativo relativo ao predicado *utente*.

```
-utente(1015,josemaria,45,lisboa,masculino).
```

3.3.2. Predicado *prestador*

O predicado *prestador* tem como atributos o ID do prestador, o tipo de prestador, a especialidade, o nome, a instituição, a cidade e o sexo. De seguida, serão apresentados três exemplos do predicado *prestador*. É de realçar que os restantes factos se encontram no Anexo A.

```
prestador(2001,medico,cardiologia,ruialves,cuf,braga,masculino).
prestador(2002,enfermeiro,pediatria,anasilva,trofasaude,porto,
feminino).
```

```
prestador(2003,medico,medicinainterna,teresaalves,publico,vianad
ocastelo,feminino).
```

De forma a representar o conhecimento perfeito negativo relativamente ao predicado *prestador*, definiu-se o predicado *–prestador*, onde IP é o ID do prestador, T o tipo de prestador (enfermeiro ou médico), E a sua especialidade, N o nome do prestador, I a instituição onde exerce, C a cidade e por último S o sexo por o qual se identifica.

```
-prestador(IP,T,E,N,I,C,S) :-
           nao(prestador(IP,T,E,N,I,C,S)),
           nao(excecao(prestador(IP,T,E,N,I,C,S))).
```

À semelhança do que foi realizado para o predicado *utente*, para o predicado *prestador*, foi também inserido um exemplo de conhecimento negativo.

```
-prestador(2012,medico,pneumologia,saracosta,publico,porto,
feminino).
```

3.3.3. Predicado *ato*

O predicado *ato* tem como atributos o ID do ato, a data do mesmo (no formato ddmmaa), o ID do utente, o ID do prestador, a descrição do ato e o custo, em euros. De seguida, serão apresentados três exemplo do predicado *ato*. É de realçar que os restantes factos se encontram no Anexo A.

```
ato(3001, 051121, 1001, 2001, curativo,20).
ato(3002, 080921, 1008, 2002, curativo,5).
ato(3003, 080921, 1006, 2008, consulta,15).
```

Mais uma vez, de forma a representar o conhecimento perfeito negativo relativamente ao predicado *ato*, definiu-se o predicado *–ato*, onde IA é o ID do ato, D a data do ato, IU o ID do utente ao qual é realizado o ato, IP o ID do prestador que realiza o ato, D a descrição do mesmo e por fim o custo do ato.

```
-ato(IA,D,IU,IP,D,C):-
```

```
        nao(ato(IA,D,IU,IP,D,C)),
        nao(excecao(ato(IA,D,IU,IP,D,C))).
```

Foi também inserido um exemplo de conhecimento negativo relativo ao predicado *ato*.

```
-ato(3011, 210820, 1002, 2001, consulta,17).
```

3.3.4. Exemplos da representação de conhecimento positivo e negativo

A título de exemplo, são apresentados dois exemplos relativos ao conhecimento perfeito positivo (Figura 1) e ao conhecimento perfeito negativo (Figura 2), respetivamente, presentes na base de conhecimento. Para tal recorreu-se ao interpretador de PROLOG SWI e ao meta-predicado *si*, apresentado posteriormente na Secção 3.7.1.

```
?- si(utente(1008, teresaferreira, 5, porto, feminino),R).
R = verdadeiro .
```

Figura 1. Exemplo de conhecimento perfeito positivo.

Consultando a base de conhecimento (Anexo A), verifica-se a existência do facto `utente(1008, teresaferreira, 5, porto, feminino)`. Logo, há uma prova positiva da sua existência, isto é, o facto está declarado e, por isso, a resposta é verdadeira, conforme se pode observar na Figura 1.

```
?- si(prestador(2025,medico,pneumologia,alexandreramos,publico,braga, masculino),R).
R = falso .
```

Figura 2. Exemplo de conhecimento perfeito negativo.

Neste caso, a primeira cláusula do meta-predicado *si* falha, uma vez que na base de conhecimento não existe uma prova da existência do facto apresentado na Figura 2. No entanto, a segunda cláusula afirma que a resposta a uma questão colocada é falsa se existir uma prova negativa da sua existência. Mais concretamente, o facto colocado como questão é falso se não existir uma prova positiva e se não for uma exceção.

3.4. Representação de Conhecimento Imperfeito - Valores Nulos

No presente trabalho, foram representados vários tipos de conhecimento imperfeito: conhecimento imperfeito incerto, impreciso e interdito. Isto foi possível através da representação de valores nulos dos tipos: desconhecido genericamente, desconhecido, mas de um conjunto de valores possíveis e por último do tipo desconhecido e sem permissão para ser conhecido.

3.4.1. Conhecimento Imperfeito Incerto

Como forma de representar conhecimento imperfeito incerto foram utilizados valores nulos incertos através da criação de cláusulas utilizando o predicado *excecao*. Isto resulta em um dado termo ser considerado uma exceção se se verificar que um dos seus argumentos foi declarado na base de conhecimento com um valor desconhecido.

Como exemplo de conhecimento imperfeito incerto, neste trabalho, tem-se o caso em que a especialidade de um prestador é desconhecida.

```
prestador(2009, enfermeiro, especialidade_desconhecido,
carlossa, publico, vianadocastelo, masculino).
```

Dado que o exemplo acima dado se insere no tipo de conhecimento imperfeito incerto, foi definida uma cláusula do predicado *excecao*. Esta explicita que um prestador cuja especialidade é especialidade_desconhecido é uma exceção ao predicado *prestador*.

```
excecao(prestador(IP,T,E,N,I,C,S)):-
      prestador(IP,T,especialidade_desconhecido,N,I,C,S).
```

Da mesma forma, forma criados valores nulos incertos para os predicados *utente* e *ato*, cujas cidade e descrição são desconhecidas, respetivamente, como se pode observar abaixo.

```
utente(1012, josereis, idade_desconhecido, famalicao, masculino).
```

```
excecao(utente(IU,N,I,C,S)):-
          utente(IU,N,idade_desconhecido,C,S).

ato(3009, 180621, 1005, 2001, descricao_desconhecido,13).

excecao(ato(IA,D,IU,IP,D,C)):-
          ato(IA,D,IU,IP,descricao_desconhecido,C).
```

3.4.2. Conhecimento Imperfeito Impreciso

O conhecimento imperfeito impreciso é também um tipo de conhecimento de carácter desconhecido, tal como o referido anteriormente. Contudo, este difere num ponto relevante, sendo que ao invés de ser genericamente desconhecido, sabe-se que este se encontra num conjunto de valores. Este conjunto de valores pode ser um intervalo ou diferentes opções para o mesmo atributo.

Com o intuito de representar este tipo de conhecimento, foram definidas cláusulas do predicado *excecao*, para que fossem consideradas exceções ao predicado pretendido.

Como foi referido, o conjunto de valores pode ser de tipos diferentes. Assim, no exemplo abaixo apresentado, tem-se um prestador cujo ID é 2010, mas cuja especialidade é psiquiatria ou psicologia, não se sabendo precisamente qual das duas será a sua verdadeira especialidade. Esta imprecisão resulta em duas exceções na base de conhecimento.

É de realçar que apesar de serem apresentados apenas exemplos de conhecimento imperfeito impreciso para os predicados *prestador* e *ato*, este também foi definido para o predicado *utente*, encontrando-se no Anexo A.

```
excecao(prestador(2010,enfermeiro,psiquiatria,lauramiquelina,pub
lico,braga, feminino)).
excecao(prestador(2010,enfermeiro,psicologia,lauramiquelina,publ
ico,braga, feminino)).
```

No exemplo acima, aquando a realização de uma questão para saber se a especialidade do *prestador* 2010 é psiquiatria ou psicologia, é devolvido o valor desconhecido.

Num segundo exemplo, agora relativo ao predicado *ato*, tem-se um ato cujo ID é 3010 e custo do mesmo se encontra entre um intervalo de valores de 12 a 22 euros. Esta imprecisão resulta numa exceção na base de conhecimento.

```
excecao(ato(3010, 190221, 1011, 2007, consulta,CD)):-
          CD >= 12,
          CD =< 22.
```

No exemplo acima, aquando a realização de uma questão quanto ao custo do *ato* 3010 ser um valor qualquer entre 12 e 22 euros, é devolvido o valor desconhecido.

3.4.3. Conhecimento Imperfeito Interdito

Por último, foi introduzido conhecimento imperfeito interdito. Este carateriza-se pela representação de um tipo de conhecimento ou atributo que se pretende que não seja conhecido.

De forma a representar este tipo de conhecimento, foi necessária a implementação de um novo predicado *interdito*, apresentando como argumento um valor que não é permitido conhecer.

É de realçar que apesar de serem apresentados apenas exemplos de conhecimento imperfeito interdito para o predicado *utente*, este também foi definido para o predicado *prestador*, encontrando-se no Anexo A.

Para representar um conhecimento imperfeito interdito, foi apresentado o seguinte exemplo relativamente ao predicado *utente*, onde a cidade em que o utente `claudiareis` reside está interdita de ser acedida.

```
utente(1014, claudiareis, 17, cidadeinterdito, feminino).
```

Para que o termo fosse identificado como conhecimento imperfeito e para que sempre que fossem colocadas questões sobre o mesmo resultasse em desconhecido, foi necessário criar uma cláusula com o predicado *excecao*.

```
excecao(utente(IU,N,I,C,S)):-
          utente(IU,N,I,cidadeinterdito,S).
```

Por último, foi introduzido um invariante que impede a introdução de novo conhecimento relativo à cidade onde o utente reside, tornando assim impossível aceder a esse conhecimento.

```
+utente(IU,N,I,C,S)::(solucoes(C,(utente(1014, claudiareis, 17,
C, feminino),
                        nao(interdito(C))),L),
                        comprimento(L,R), R==0).
```

O invariante acima representado impede que seja introduzido um facto na base de conhecimento associado ao utente com ID 1014 que não seja do tipo interdito. Isto deve-se ao facto de que, dado um determinado termo do predicado *utente*, procuram-se todas as cidades não interditas associadas ao mesmo utente e guardam-se os resultados numa lista L. Obtida a lista é utilizado o predicado *comprimento* (apresentado na Secção 3.5.2) que tem como resultado o comprimento da lista. Se esse valor for zero é garantido que não existe nenhum valor de cidade associado ao utente com ID 1014, a não ser o valor `cidadeinterdito`.

3.4.4. Exemplos da representação de conhecimento imperfeito incerto

Na Figura 3 está representado um exemplo da representação de conhecimento imperfeito incerto.

```
?- si(prestador(2009, enfermeiro, obstetricia, carlossa,publico, vianadocastelo, masculino),R).
R = desconhecido.
```

Figura 3. Exemplo de conhecimento imperfeito incerto.

Como se pode verificar na Figura 3, a especialidade do prestador cujo ID é 2009 é desconhecida uma vez que anteriormente se atribuiu um valor nulo a esse atributo, o que representa conhecimento incerto. Assim, a primeira cláusula do meta-predicado *si* falha, porque o facto representado na questão não está explicitamente declarado na base de conhecimento. A segunda cláusula falha, também, sendo que a questão representa uma exceção, e a cláusula que determina o conhecimento negativo para o predicado *prestador*

não considera as exceções como informação negativa. O meta-predicado *si* recorre, então, à terceira instância para responder à questão que é que não existe qualquer tipo de prova de que a especialidade do prestador com ID 2009 é `obstetricia`, mas também não existe nenhuma prova do contrário. Logo, o resultado é desconhecido.

Além disto, é importante salientar que qualquer questão cujo argumento declarado é o valor nulo enunciado na base de conhecimento, resulta num valor lógico verdadeiro pois é verdade que o dado atributo é desconhecido, como se pode observar na Figura 4.

```
?- prestador(2009, enfermeiro, especialidade_desconhecido, carlossa,publico, vianadocastelo, masculino).
true.
```

Figura 4. Exemplo de conhecimento imperfeito incerto com valor nulo.

3.4.5. Exemplos da representação de conhecimento imperfeito impreciso

Na Figura 5 estão representados exemplo da representação de conhecimento imperfeito impreciso.

```
?- si(utente(1013, clarafernandes, 52, viladoconde, feminino),R).
R = desconhecido.

?- si(utente(1013, marafernandes, 52, viladoconde, feminino),R).
R = desconhecido.

?- si(utente(1013, larafernandes, 52, viladoconde, feminino),R).
R = falso .
```

Figura 5. Exemplo de conhecimento imperfeito impreciso.

Dado que não existe informação positiva que afirma se o utente com ID 1013 se chama `clarafernandes` ou `marafernandes`, não se pode concluir que isso é verdade, mas tendo sido declaradas exceções na base de conhecimento em relação a ambos os nomes associados a este IP de utente, o resultado é desconhecido. Porém, todas as restantes questões colocadas que envolvam nomes diferentes dos nomes `clarafernandes` e `marafernandes`, resultam sempre numa resposta falsa. Isto pode ser verificado na última questão colocada no exemplo acima.

3.4.6. Exemplos da representação de conhecimento imperfeito interdito

Na Figura 6 estão representados exemplos da representação de conhecimento imperfeito interdito.

```
?- si(utente(1014, claudiareis, 17, cidadeinterdito, feminino),R).
R = verdadeiro .

?- si(utente(1014, claudiareis, 17, porto, feminino),R).
R = desconhecido.
```

Figura 6. Exemplo de conhecimento imperfeito interdito.

Na base de conhecimento foi declarada uma exceção com o valor *cidadeinterdito* para a cidade onde reside o utente com ID 1014 e por isso justifica-se o valor verdadeiro resultante da primeira questão demonstrada no exemplo acima. Além disso, sabe-se também que quando o sistema de inferência se depara com uma exceção, o resultado é desconhecido. Adicionalmente, a cidade de residência do utente com ID 1014, além de desconhecida é também interdita, impedindo assim a evolução de conhecimento relativamente a esse atributo do utente.

3.5. Predicados Auxiliares

Os predicados desta secção são denominados de Predicados Auxiliares, uma vez que a sua função é ser a base de outros predicados necessários ao trabalho proposto. Estes vão ajudar as extensões dos predicados e os invariantes, que serão apresentados mais à frente, a atingirem o seu objetivo final.

3.5.1. Predicado *solucoes*

O predicado *solucoes* é utilizado quando se pretende listar todas as soluções como resposta a uma dada pergunta. Este predicado contém três argumentos: o X corresponde ao elemento que se pretende obter, o Y é o teorema que se pretende provar quando é feita uma dada pergunta e o Z é a lista que armazena o conjunto de soluções. Note-se que o *findall* é um predicado já implementado no interpretador SWI.

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

3.5.2. Predicado *comprimento*

Este predicado retorna o número de elementos de uma determinada lista. Para isso, recebe como argumentos uma lista L e o número de elementos dessa lista como resultado R. O critério de paragem deste predicado retorna comprimento 0, caso a lista seja vazia. Se o comprimento da lista L for N, sabe-se que o comprimento de uma lista que tem como cabeça um elemento X e como cauda L é N+1. Assim, este predicado utiliza a recursividade para somar o número de elementos de uma lista só termina quando atinge o critério de paragem.

```
% Extensão do predicado comprimento: Lista, Resultado -> {V,F}
comprimento( [],0 ).
      comprimento( [X|L],R ) :-
      comprimento( L,N ),
      R is N+1.
```

3.5.3. Predicado *insercao*

O predicado *insercao* tem como função atualizar a base de conhecimento através da inserção de termos. De forma a remover termos inseridos que são inválidos, é utilizado o *retract* em conjugação com *!, fail*. Isto obriga que o processo falhe, impedindo que seja executada a inserção.

Os predicados *assert* e *retract* estão previamente implementados no SWI.

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

3.5.4. Predicado *teste*

O predicado *teste* tem como objetivo a validação de um termo após a sua inserção, verificando que não existem contradições.

```
% Extensão do predicado teste: Lista -> {V,F}
teste( [] ).
teste( [I|L] ) :-
      I,teste( L ).
```

3.5.5. Predicado *remocao*

O presente predicado tem como finalidade a remoção de termos, recorrendo à função do PROLOG *retract*. Este predicado recebe um termo como argumento e remove da base uma ou mais cláusulas (por retrocesso) que unificam com esse termo.

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

3.5.6. Meta-predicado *nao*

Para a representação de conhecimento negativo foi necessário recorrer ao meta-predicado *nao*:

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

O meta-predicado *nao* é responsável por implementar a negação por falha na prova e está divido em duas instâncias. Na primeira instância, o predicado recebe como parâmetro uma `Questao` e, caso seja encontrada uma prova positiva para essa mesma `Questao`, então esta sucede e é aplicado o mecanismo que impede o retrocesso (!). Desta

forma, o processo falha, sendo devolvida uma resposta negativa por falha na prova. Caso não seja encontrada uma prova positiva para a `Questao`, então a negação dessa mesma `Questao` vai suceder. Isto é, se um determinado facto existir na base de conhecimento, então não se pode dizer que não existe e é retornado no. Por outro lado, se a `Questao` em causa não puder ser provada, então pode-se dizer que a mesma não existe e é retornado yes.

3.6. Invariantes

Os invariantes podem ser vistos como testes à consistência da base de conhecimento em momentos de alteração à mesma. Estes garantem uma integridade estrutural e referencial da base de conhecimento, ou seja, impedem a inserção de conhecimento repetido (invariantes estruturais) e asseguram a consistência da informação existente na base de conhecimento (invariantes referenciais).

Tendo em conta os diversos predicados construídos, definiram-se invariantes de inserção e remoção para cada um deles, de acordo com a consistência do conhecimento que se pretende obter.

3.6.1. Predicado *utente*

3.6.1.1. Invariantes de Inserção

Para a inserção de factos positivos relativos a utentes foram criados dois invariantes. O primeiro trata-se de um invariante estrutural, que não permite o registo de utentes já existentes na base de conhecimento. Através do predicado *solucoes* obtiveram-se todos os factos utente, cujos argumentos coincidiam com ID, nome, idade, cidade e sexo do utente que se pretende inserir, ficando estes guardados na lista S. Uma vez que apenas se pretende a existência de factos utente únicos, então, no fim da inserção só poderá existir um facto utente com os argumentos dados, indicando assim que o registo é único. Este resultado obtém-se através do uso do predicado *comprimento*, que determina o comprimento da lista S, impondo neste caso, que o comprimento da lista seja 1, ou seja, apenas uma instância do facto em causa poderá existir na base de conhecimento.

```
+utente(A,B,C,D,E) :: (solucoes((A,B,C,D,E),
```

```
                    utente( A,B,C,D,E),S),
                    comprimento( S,N ), N == 1).
```

No mesmo contexto, foi criado um invariante referencial que não permite que seja adicionado um utente com o mesmo ID. Este invariante não permite a inserção de um utente com um ID já existente noutro utente. A criação deste invariante segue a mesma resolução apresentada para o invariante estrutural, comparando, neste caso, apenas os IDs dos utentes.

```
+utente(A,_,_,_,_) :: (solucoes(A, utente( A,_,_,_,_), S),
                    comprimento( S,N ), N == 1).
```

3.6.1.2. Invariantes de Remoção

Para a remoção de um utente, foram criados novamente dois invariantes. O primeiro permite remover um utente se este já existir na base de conhecimento.

```
-utente(A,B,C,D,E)::(solucoes((A,B,C,D,E),utente(A,B,C,D,E),
                    S), comprimento( S,N ), N == 0).
```

Relativamente ao segundo invariante, para um utente poder ser removido, este não pode estar associado a nenhum ato registado na base de conhecimento.

```
-utente(A,B,C,D,E) :: (solucoes(A, ato( _,_,A,_,_,_ ), S),
                    comprimento( S,N ), N == 0).
```

3.6.2. Predicado *prestador*

3.6.2.1. Invariantes de Inserção

Para não existir a possibilidade de inserção de dados repetidos na base de conhecimento foi criado um invariante estrutural, procedendo-se da mesma forma que para o predicado *utente*, sendo que a explicação do seu funcionamento segue o mesmo padrão.

```
+prestador(A,B,C,D,E,F,G) :: (solucoes((A,B,C,D,E,F,G),
                         prestador( A,B,C,D,E,F,G), S),
```

```
                                comprimento( S,N ), N == 1).
```

Uma vez que na base de conhecimento não podem existir dois prestadores com o mesmo ID, procedeu-se ao desenvolvimento de um invariante referencial para esse efeito.

```
+prestador(A,_,_,_,_,_,_) :: (solucoes(A,
                            prestador(A,_,_,_,_,_,_), S),
                            comprimento( S,N ), N == 1).
```

3.6.2.2. Invariantes de Remoção

O primeiro invariante apresentado abaixo permite a remoção de um prestador caso este exista na base de conhecimento.

```
-prestador(A,B,C,D,E,F,G)::(solucoes((A,B,C,D,E,F,G),
                            prestador(A,B,C,D,E,F,G), S),
                            comprimento( S,N ), N == 0).
```

Quanto ao segundo invariante, este confirma se o prestador não está ligado ao predicado *ato* e nem a nenhuma das suas clausulas, e em caso afirmativo, removo-o.

```
-prestador(A,B,C,D,E,F,G) :: (solucoes(A, ato( _,_,_,A,_,_),
                            S),comprimento( S,N ), N == 0).
```

3.6.3. Predicado *ato*

3.6.3.1. Invariantes de Inserção

Para a inserção de novo conhecimento na base de conhecimento relativamente ao predicado *ato* foi criado um invariante estrutural. Tendo em conta este predicado e seguindo a mesma lógica previamente referida, não podem existir dois atos com os mesmos dados, só se podendo assim, inserir atos com dados diferentes dos já existentes.

```
+ato(A,B,C,D,E,F) :: (solucoes(A, ato(A,B,C,D,E,F), S),
                            comprimento( S,N ), N == 1).
```

No mesmo contexto, foi criado um invariante referencial que não permite que seja adicionado um ato com o mesmo ID. A criação deste invariante segue a mesma resolução apresentada para o invariante estrutural, comparando, neste caso, apenas os IDs dos atos.

```
+ato(A,_,_,_,_,_) :: (solucoes(A, ato(A,_,_,_,_,_), S),
                            comprimento( S,N ), N == 1).
```

3.6.3.2. Invariantes de Remoção

Usando o mesmo padrão que os invariantes já referidos, só é possível remover um ato se este existir na base de conhecimento.

```
-ato(A,B,C,D,E,F)::(solucoes((A,B,C,D,E,F),ato(A,B,C,D,E,F),S),
                        comprimento( S,N ), N == 0).
```

3.7. Sistema de Inferência

Uma vez que o contradomínio dos cenários estudados apresenta três valores de verdade possíveis: {V, F, D}, surgiu a necessidade de construir um meta-predicado, em PROLOG, capaz de lidar com os três diferentes tipos de resposta referidos anteriormente. Esse meta-predicado é designado por *si*.

Para verificar a veracidade de uma questão complexa, isto é, para os casos de conjunção e/ou disjunção de questões, construiu-se um sistema de inferência denominado *siC*.

3.7.1. Meta-predicado *si*

Em primeiro lugar criou-se um meta-predicado *si* que, ao receber uma questão, devolve o valor verdadeiro, falso ou desconhecido.

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

Caso uma `Questao` esteja explicitamente definida na base de conhecimento, o predicado *si* devolve uma resposta afirmativa (verdadeiro). Por outro lado, se uma questão estiver fortemente negada (¬`Questao`) na base de conhecimento, este predicado retornará uma resposta obrigatoriamente negativa (falso). Por último, o predicado devolve como resposta desconhecido, caso não exista uma prova positiva da existência da `Questao` na base de conhecimento e nem existir uma prova negativa da existência da `Questao`.

3.7.2. Meta-predicado *siC*

Este predicado foi construído com o objetivo de obter um sistema de inferência capaz de responder a uma composição de questões. Para se distinguir os casos de conjunção e disjunção foram utilizados entre as questões, e's e ou's, respetivamente.

Para a construção do *siC* foi inicialmente necessário definir a resposta que o programa deve dar perante os diferentes cenários (Tabela 1). Desta forma, criaram-se dois predicados: *conjuncao* e *disjuncao*. Estes têm três argumentos, que são os valores das `Questoes` 1 e 2 e a conjunção/disjunção, respetivamente, desses valores. De notar que as `Questoes` 1 e 2 podem representar quer `Questoes` simples, quer uma composição de `Questoes`.

É de realçar que para uma conjunção ser falsa, basta uma das `Questoes` ser falsa e para uma disjunção ser verdadeira, basta uma das `Questoes` ser verdadeira. Por último, a presença da variável anónima "_" indica que o valor da `Questao` pode ser qualquer um dos três valores possíveis. No Anexo B estão representados alguns exemplos utilizados para testar o meta-predicado *siC*.

Tabela 1. Conjunção e Disjunção.

Q1	Q2	Conjunção ("e")	Disjunção ("ou)
V	**V**	V	V
V	**D**	D	V
V	**F**	F	V
D	**V**	D	V

D	**D**	D	D
D	**F**	F	D
F	**V**	F	V
F	**D**	F	D
F	**F**	F	F

```
% Extensão do predicado conjuncao: Valor da Questão 1, Valor da Questão 2, Conjunção -> {V,D,F}
conjuncao(verdadeiro,verdadeiro,verdadeiro).
conjuncao(_,falso, falso).
conjuncao(falso, _, falso).
conjuncao(verdadeiro,desconhecido,desconhecido).
conjuncao(desconhecido,verdadeiro,desconhecido).
conjuncao(desconhecido,desconhecido,desconhecido).

% Extensão do predicado disjuncao: Valor da Questão 1, Valor da Questão 2, Disjunção -> {V,D,F}
disjuncao(verdadeiro,_,verdadeiro).
disjuncao(_,verdadeiro,verdadeiro).
disjuncao(verdadeiro,desconhecido,verdadeiro).
disjuncao(desconhecido,desconhecido,desconhecido).
disjuncao(falso,falso,falso).
disjuncao(desconhecido,falso,desconhecido).
     disjuncao(falso,desconhecido,desconhecido).

% Extensão do meta-predicado siC: Composição de Questões, Resposta -> {V,D,F}
siC( Q1 e Q2,R ) :-
     siC( Q1,R1 ),
     siC( Q2,R2 ),
     conjuncao( R1,R2,R ).
siC( Q1 ou Q2,R ) :-
     siC( Q1,R1 ),
     siC( Q2,R2 ),
     disjuncao( R1,R2,R ).
siC(Q,R) :- si(Q,R).
```

3.8. Evolução e Involução do Conhecimento

Uma determinada base de conhecimento pode sofrer alterações quanto à informação que armazena. Assim, para resolver a problemática da evolução do conhecimento foi necessário desenvolver predicados que permitissem manipular a inserção e remoção de factos da base de conhecimento.

Estes predicados foram construídos de modo que a adição de novo conhecimento ao sistema e a eliminação de conhecimento já presente no sistema respeitasse as restrições impostas pelos invariantes criados.

Desta forma, para a evolução de conhecimento podem-se considerar dois casos: inserção de conhecimento novo na base de conhecimento e manipulação de conhecimento já existente.

Para o caso prático em estudo, não se considerou a inserção direta de conhecimento negativo na base de conhecimento, uma vez que tal não faria sentido. A presença de conhecimento explicitamente negativo é, então, obtida pela manipulação de conhecimento já existente.

3.8.1. Inserir conhecimento

O predicado *registar* possibilita a inserção de novos factos relativos a um novo utente, prestador ou ato. Este predicado recebe como argumento um Termo, que corresponde ao conhecimento que queremos inserir. Através do predicado auxiliar *solucoes* (secção 3.5.1) gera-se uma lista (Linv), que inclui todos os invariantes descritos para o termo introduzido. De seguida, é feita a inserção desse Termo através da invocação do predicado *insercao*, que recorre à funcionalidade do *assert*. Posteriormente, invoca-se o predicado *teste*, responsável por verificar se cada um dos invariantes continua válido ou não. Caso algum invariante não seja válido retrocede-se à segunda cláusula do predicado *insercao*, sendo esta responsável pela remoção do facto que se tentou inserir, através da funcionalidade do *retract*. A presença do !, fail (insucesso na prova) na segunda cláusula do predicado *insercao* obriga a que o procedimento falhe aí e impede o seu retrocesso. Desta forma, é garantida a validade da base de conhecimento consoante os invariantes que foram criados, ou seja, é feito um teste à sua consistência.

```
% Extensão do predicado que permite a inserção de conhecimento: %
Termo -> {V,F}
registar( Termo ) :-
            solucoes(I, +Termo :: I, Linv ),
            insercao( Termo ),
            teste( Linv ).
```

3.8.2. Remover conhecimento

O predicado *remover* possibilita a remoção de um determinado facto da base de conhecimento, ou seja, remover informação relativa a um determinado utente, prestador ou ato. O predicado *remover* recebe como argumento um Termo, que corresponde ao conhecimento que se pretende remover. Através do predicado auxiliar *solucoes*, gera-se uma lista (Linv), que inclui todos os invariantes descritos para o Termo introduzido. De seguida, invoca-se o predicado *teste*, responsável por verificar se cada um dos invariantes continua válido ou não. Caso o invariante seja válido, invoca-se o predicado auxiliar *remocao,* responsável pela remoção do facto pretendido, através da funcionalidade do *retract*. Ao contrário do que se sucedia anteriormente para o predicado *registar*, em que os invariantes testavam a consistência da base de conhecimento, neste caso, os invariantes fazem um "teste à permissão" para remover conhecimento. Assim, o predicado *teste* é invocado antes do predicado *remocao*.

```
% Extensão do predicado que permite a remoção de conhecimento:
Termo % -> {V,F}
remover( Termo ) :-
            solucoes( I, -Termo::I, Linv ),
            remocao( Termo ),
            teste( Linv ).
```

3.8.3. Atualizar conhecimento

Nesta secção foram desenvolvidos três predicados para atualizar conhecimento perfeito positivo. Um para o utente, *atualizarUtente*, caso seja necessário atualizar o argumento idade de um determinado utente com um dado ID, outro semelhante para o ato, *atualizarCustoAto*, que apenas permite atualizar o custo associado ao mesmo, e outro para o prestador, *atualizarPrestador*, em que se pode alterar a especialidade, a instituição onde o prestador exerce

e a respetiva cidade. Os predicados funcionam de maneira semelhante. Primeiramente, removem o conhecimento desatualizado (através do predicado *remover*) que é aquele que se pretende modificar da base de conhecimento e, posteriormente, adicionam o conhecimento atualizado (através do predicado *registar*).

```
% Extensão do predicado que permite a atualizar um utente: Termo
-> {V,F}
atualizarUtente(utente(IU,N,I,C,S), utente(IU,N,In,C,S)) :-
          remover(utente(IU,N,I,C,S)),
          registar(utente(IU,N,In,C,S)).


% Extensão do predicado que permite atualizar o custo de um ato:
Termo -> {V,F}
atualizarCustoAto(ato(IA,D,IU,IP,D,C), ato(IA,D,IU,IP,D,Cn)):-
          remover(ato(IA,D,IU,IP,D,C)),
          registar(ato(IA,D,IU,IP,D,Cn)).


% Extensão do predicado que permite a atualizar um prestador:
Termo -> {V,F}
atualizarPrestador(prestador(IP,TP,E,N,I,C,S),
          prestador(IP,TP,En,N,In,Cn,S)):-
          remover(prestador(IP,TP,E,N,I,C,S)),
          registar(prestador(IP,TP,En,N,In,Cn,S)).
```

De seguida apresenta-se um exemplo de utilização do predicado *atualizarUtente* (Figura 7) e do predicado *atualizarPrestador* (Figura 8), uma vez que este permite modificar mais do que um atributo. É de realçar que não é apresentado um exemplo para o predicado *atualizarCustoAto*, pois, este funciona da mesma forma que o *atualizarUtente.*

```
?- listing(utente).
:- dynamic utente/5.

utente(1001, laravaz, 22, braga, feminino).
utente(1002, marianalindo, 20, vianadocastelo, feminino).
utente(1003, tiagonovais, 21, fafe, masculino).
utente(1004, luisvaz, 47, braga, masculino).
utente(1005, manuelmartins, 86, barcelos, masculino).
utente(1006, mariasilva, 3, famalicao, feminino).
utente(1007, luisparente, 44, braga, masculino).
utente(1008, teresaferreira, 5, porto, feminino).
utente(1009, nunocosta, 29, viladoconde, masculino).
utente(1010, catarinacameira, 15, vianadocastelo, feminino).
utente(1011, paulolopes, 58, porto, masculino).
utente(1012, josereis, idade_desconhecido, famalicao, masculino).
utente(1014, claudiareis, 17, cidadeinterdito, feminino).

true.

?- atualizarUtente(utente(1009,nunocosta,29,viladoconde,masculino),utente(1009,nunocosta,30
,viladoconde,masculino)).
true .

?- listing(utente).
:- dynamic utente/5.

utente(1001, laravaz, 22, braga, feminino).
utente(1002, marianalindo, 20, vianadocastelo, feminino).
utente(1003, tiagonovais, 21, fafe, masculino).
utente(1004, luisvaz, 47, braga, masculino).
utente(1005, manuelmartins, 86, barcelos, masculino).
utente(1006, mariasilva, 3, famalicao, feminino).
utente(1007, luisparente, 44, braga, masculino).
utente(1008, teresaferreira, 5, porto, feminino).
utente(1010, catarinacameira, 15, vianadocastelo, feminino).
utente(1011, paulolopes, 58, porto, masculino).
utente(1012, josereis, idade_desconhecido, famalicao, masculino).
utente(1014, claudiareis, 17, cidadeinterdito, feminino).
utente(1009, nunocosta, 30, viladoconde, masculino).

true.
```

Figura 7. Resultado da utilização do predicado *atualizarUtente*.

```
?- listing(prestador).
:- dynamic prestador/7.

prestador(2001, medico, cardiologia, ruialves, cuf, braga, masculino).
prestador(2002, enfermeiro, pediatria, anasilva, trofasaude, porto, feminino).
prestador(2003, medico, medicinainterna, teresaalves, publico, vianadocastelo, feminino).
prestador(2004, enfermeiro, oftalmologia, goretiparente, publico, vianadocastelo, feminino)
.
prestador(2005, medico, pneumologia, luismartins, cuf, braga, masculino).
prestador(2006, medico, dentaria, jorgeamorim, trofasaude, porto, masculino).
prestador(2007, medico, dermatologia, diogopereira, trofasaude, braga, masculino).
prestador(2008, medico, pediatria, rosasilva, publico, braga, feminino).
prestador(2009, enfermeiro, especialidade_desconhecido, carlossa, publico, vianadocastelo,
masculino).
prestador(2011, medico, dentaria, ivonerosas, cuf, cidadeinterdito, feminino).

true.

?- atualizarPrestador(prestador(2007,medico,dermatologia,diogopereira,trofasaude,braga,masc
ulino),prestador(2007,medico,obstetricia,diogopereira,privadoporto,lisboa,masculino)).
true .

?- listing(prestador).
:- dynamic prestador/7.

prestador(2001, medico, cardiologia, ruialves, cuf, braga, masculino).
prestador(2002, enfermeiro, pediatria, anasilva, trofasaude, porto, feminino).
prestador(2003, medico, medicinainterna, teresaalves, publico, vianadocastelo, feminino).
prestador(2004, enfermeiro, oftalmologia, goretiparente, publico, vianadocastelo, feminino)
.
prestador(2005, medico, pneumologia, luismartins, cuf, braga, masculino).
prestador(2006, medico, dentaria, jorgeamorim, trofasaude, porto, masculino).
prestador(2008, medico, pediatria, rosasilva, publico, braga, feminino).
prestador(2009, enfermeiro, especialidade_desconhecido, carlossa, publico, vianadocastelo,
masculino).
prestador(2011, medico, dentaria, ivonerosas, cuf, cidadeinterdito, feminino).
prestador(2007, medico, obstetricia, diogopereira, privadoporto, lisboa, masculino).

true.
```

Figura 8. Resultado da utilização do predicado *atualizarPrestador*.

3.8.4. Manipular conhecimento

3.8.4.1. Predicado *evolucao_PP*

O predicado *evolucao_PP* permite alterar um conhecimento positivo já existente para um conhecimento novo positivo, passando como argumento, *Cnovo*, que corresponde ao novo termo a adicionar na base de conhecimento.

Para tal, recorreu-se ao predicado *remover* para remover o conhecimento positivo existente e ao predicado *registar* para adicionar o novo termo, conhecimento positivo. É de notar que o meta-predicado *si* impõe que o conhecimento positivo esteja definido na base de conhecimento, para que seja possível a sua eliminação.

```
% Extensão do predicado que permite a evolucao do conhecimento
positivo para positivo: Cnovo -> {V,F}
evolucao_PP(Cnovo,Cpassado):-
        si(Cpassado, verdadeiro),
        remocao(Cpassado),
        insercao(Cnovo).
```

Na Figura 9 é apresentado um exemplo de utilização do predicado *evolucao_PP*.

```
?- evolucao_PP(utente(1001,luisvaz,25,lisboa,masculino),utente(1001,laravaz,22,braga,femini
no)).
true .

?- listing(utente).
:- dynamic utente/5.

utente(1002, marianalindo, 20, vianadocastelo, feminino).
utente(1003, tiagonovais, 21, fafe, masculino).
utente(1004, luisvaz, 47, braga, masculino).
utente(1005, manuelmartins, 86, barcelos, masculino).
utente(1006, mariasilva, 3, famalicao, feminino).
utente(1007, luisparente, 44, braga, masculino).
utente(1008, teresaferreira, 5, porto, feminino).
utente(1009, nunocosta, 29, viladoconde, masculino).
utente(1010, catarinacameira, 15, vianadocastelo, feminino).
utente(1011, paulolopes, 58, porto, masculino).
utente(1014, claudiareis, 17, cidadeinterdito, feminino).
utente(1001, luisvaz, 25, lisboa, masculino).

true.
```

Figura 9. Resultado da utilização do predicado *evolucao_PP*.

3.8.4.2. Predicado *evolucao_PN*

O predicado *evolucao_PN* permite alterar o conhecimento positivo já existente para conhecimento negativo, passando como argumento, *Cnovo*, que corresponde ao novo termo (conhecimento negativo) a adicionar na base de conhecimento.

Para isso, recorreu-se ao predicado *remover* para remover o conhecimento positivo existente e ao predicado *registar* para adicionar o novo termo. É de notar que o meta-predicado *si* impõe que o conhecimento positivo esteja definido na base de conhecimento para que seja possível a sua eliminação.

```
% Extensão do predicado que permite a evolução do conhecimento
positivo para negativo: Cnovo -> {V,F}
evolucao_PN(Cnovo):-
            si(Cnovo, verdadeiro),
            registar(-Cnovo),
            remover(Cnovo).
```

Na Figura 10 é apresentado um exemplo de utilização do predicado *evolucao_PN*.

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

-utente(IU, N, I, C, S) :-
    nao(utente(IU, N, I, C, S)),
    nao(excecao(utente(IU, N, I, C, S))).
-utente(1015, josemaria, 45, lisboa, masculino).
-prestador(IP, T, E, N, I, C, S) :-
    nao(prestador(IP, T, E, N, I, C, S)),
    nao(excecao(prestador(IP,
                          T,
                          E,
                          N,
                          I,
                          C,
                          S))).
-ato(IA, D, IU, IP, D, C) :-
    nao(ato(IA, D, IU, IP, D, C)),
    nao(excecao(ato(IA, D, IU, IP, D, C))).

true.

?- evolucao_PN(utente(1009,nunocosta,29,viladoconde,masculino)).
true .

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

-utente(IU, N, I, C, S) :-
    nao(utente(IU, N, I, C, S)),
    nao(excecao(utente(IU, N, I, C, S))).
-utente(1015, josemaria, 45, lisboa, masculino).
-prestador(IP, T, E, N, I, C, S) :-
    nao(prestador(IP, T, E, N, I, C, S)),
    nao(excecao(prestador(IP,
                          T,
                          E,
                          N,
                          I,
                          C,
                          S))).
-ato(IA, D, IU, IP, D, C) :-
    nao(ato(IA, D, IU, IP, D, C)),
    nao(excecao(ato(IA, D, IU, IP, D, C))).
-utente(1009, nunocosta, 29, viladoconde, masculino).

true.
```

Figura 10. Resultado da utilização do predicado *evolucao_PN*.

3.8.4.3. Predicado *evolucao_NP*

O predicado *evolucao_NP* funciona de forma semelhante ao predicado *evolucao_PN*, e permite alterar conhecimento negativo já existente para conhecimento positivo. Este predicado tem apenas um argumento, *-Cnovo*, que corresponde ao novo conhecimento negativo a adicionar na base de conhecimento.

```
% Extensão do predicado que permite a evolucao do conhecimento
negativo para positivo: Cnovo -> {V,F}
evolucao_NP(Cnovo):-
      si(-Cnovo, verdadeiro),
      remover(-Cnovo),
      registar(Cnovo).
```

Na Figura 11 é apresentado um exemplo de utilização do predicado *evolucao_NP*.

```
true.

?- evolucao_NP(utente(1015,josemaria,45,lisboa,masculino)).
true .

?- listing(utente).
:- dynamic utente/5.

utente(1003, tiagonovais, 21, fafe, masculino).
utente(1004, luisvaz, 47, braga, masculino).
utente(1005, manuelmartins, 86, barcelos, masculino).
utente(1006, mariasilva, 3, famalicao, feminino).
utente(1007, luisparente, 44, braga, masculino).
utente(1008, teresaferreira, 5, porto, feminino).
utente(1009, nunocosta, 29, viladoconde, masculino).
utente(1010, catarinacameira, 15, vianadocastelo, feminino).
utente(1011, paulolopes, 58, porto, masculino).
utente(1014, claudiareis, 17, cidadeinterdito, feminino).

utente(1001, luisvaz, 25, lisboa, masculino).
utente(1002, marianalindo, 20, vianadocastelo, feminino).
utente(1015, josemaria, 45, lisboa, masculino).
```

Figura 11. Resultado da utilização do predicado *evolucao_NP*.

3.8.4.4. Predicado *evolucao_ImpP*

O predicado *evolucao_ImpP* permite modificar o conhecimento impreciso da base de conhecimento para conhecimento positivo. Este predicado contém dois argumentos: *Cnovo* que corresponde à nova informação que se pretende inserir como conhecimento

positivo, e uma lista com o conhecimento impreciso já existente que se pretende modificar.

Primeiramente, este predicado verifica que é passada uma lista com um ou mais argumentos. Em seguida, através do predicado *verificaImp*, verifica se todos os elementos da lista têm o valor *desconhecido*. Por fim, através do predicado *removeExcecoes*, remove todas as exceções e regista o conhecimento novo através do predicado *registar*.

```
% Extensão do predicado verificaImp: Lista -> {V,F}
verificaImp([]).
verificaImp( [H|T] ) :-
      si(H,desconhecido),
      verificaImp(T).

% Extensão do predicado removeExcecoes: Lista -> {V,F}
removeExcecoes([]).
removeExcecoes([H|T]):-
      remover(excecao(H)),
      removeExcecoes(T).

% Extensão do predicado que permite a evolucao do conhecimento
impreciso para positivo: Cnovo, ListaCpassado -> {V,F}
evolucao_ImpP(Cnovo,[Cpassado1|T]):-
      T\=[],
      verificaImp([Cpassado1|T]),
      removeExcecoes([Cpassado1|T]),
      registar(Cnovo).
```

Na Figura 12 encontra-se um exemplo de utilização do predicado *evolucao_ImpP*.

```
?- evolucao_ImpP(prestador(2010,enfermeiro,psiquiatria,lauramiquelina,publico,braga,feminin
o),[prestador(2010,enfermeiro,psicologia,lauramiquelina,publico,braga,feminino),prestador(2
010,enfermeiro,psiquiatria,lauramiquelina,publico,braga,feminino)]).
true .

?- listing(prestador).
:- dynamic prestador/7.

prestador(2001, medico, cardiologia, ruialves, cuf, braga, masculino).
prestador(2002, enfermeiro, pediatria, anasilva, trofasaude, porto, feminino).
prestador(2003, medico, medicinainterna, teresaalves, publico, vianadocastelo, feminino).
prestador(2004, enfermeiro, oftalmologia, goretiparente, publico, vianadocastelo, feminino)
.
prestador(2005, medico, pneumologia, luismartins, cuf, braga, masculino).
prestador(2006, medico, dentaria, jorgeamorim, trofasaude, porto, masculino).
prestador(2007, medico, dermatologia, diogopereira, trofasaude, braga, masculino).
prestador(2008, medico, pediatria, rosasilva, publico, braga, feminino).
prestador(2009, enfermeiro, especialidade_desconhecido, carlossa, publico, vianadocastelo,
masculino).
prestador(2011, medico, dentaria, ivonerosas, cuf, cidadeinterdito, feminino).
prestador(2010, enfermeiro, psiquiatria, lauramiquelina, publico, braga, feminino).

true.
```

Figura 12. Resultado da utilização do predicado *evolucao_NP*.

3.8.4.5. Predicado *evolucao_IncP*

O predicado *evolucao_IncP* pretende substituir conhecimento incerto por conhecimento positivo. Este predicado tem dois argumentos: *Cnovo* (conhecimento positivo) e *Cpassado* (conhecimento incerto). Neste predicado, recorreu-se ao predicado *remover* para remover o conhecimento antigo incerto e ao predicado *registar* para adicionar o novo conhecimento positivo. O meta-predicado *si* impõe que o conhecimento desatualizado seja verdadeiro, para que este possa ser eliminado.

```
% Extensão do predicado que permite a evolucao do conhecimento
incerto para positivo: Cnovo, Cpassado -> {V,F}
evolucao_IncP(Cnovo, Cpassado):-
      si(Cpassado,verdadeiro),
      remover(Cpassado),
      registar(Cnovo).
```

Na Figura 13 encontra-se um exemplo de utilização do predicado *evolucao_IncP*.

```
[debug]  ?- evolucao_IncP(utente(1012,josereis,40,famalicao,masculino),utente(1012,josereis
,idade_desconhecido,famalicao,masculino)).
true .

[debug]  ?- listing(utente).
:- dynamic utente/5.

utente(1001, laravaz, 22, braga, feminino).
utente(1002, marianalindo, 20, vianadocastelo, feminino).
utente(1003, tiagonovais, 21, fafe, masculino).
utente(1004, luisvaz, 47, braga, masculino).
utente(1005, manuelmartins, 86, barcelos, masculino).
utente(1006, mariasilva, 3, famalicao, feminino).
utente(1007, luisparente, 44, braga, masculino).
utente(1008, teresaferreira, 5, porto, feminino).
utente(1009, nunocosta, 29, viladoconde, masculino).
utente(1010, catarinacameira, 15, vianadocastelo, feminino).
utente(1011, paulolopes, 58, porto, masculino).
utente(1014, claudiareis, 17, cidadeinterdito, feminino).
utente(1012, josereis, 40, famalicao, masculino).
```

Figura 13. Resultado da utilização do predicado evolucao_IncP.

3.8.4.5. Predicado *evolucao_IncImp*

O predicado *evolucao_IncImp* permite transformar conhecimento do tipo incerto em conhecimento do tipo impreciso. Este predicado necessita de três argumentos, *Cnovo1*, *Cnovo2* e *Cpassado*. Os dois primeiros correspondem à informação relativa ao conhecimento impreciso que se quer introduzir na base de conhecimento e o último diz respeito ao conhecimento incerto já presenta na base de conhecimento.

O predicado *remover* elimina o conhecimento incerto existente, se o meta-predicado *si* afirmar que este é verdadeiro e o predicado *registar* permite inserir as duas novas exceções referentes ao novo conhecimento impreciso.

```
% Extensão do predicado que permite a evolucao do conhecimento
incerto para impreciso: Cnovo1, Cnovo2, Cpassado-> {V,F}
evolucao_IncImp(Cnovo1,Cnovo2,Cpassado):-
            si(Cpassado,verdadeiro),
            remover(Cpassado),
            registar(excecao(Cnovo1)),
            registar(excecao(Cnovo2)).
```

Na Figura 14 encontra-se um exemplo de utilização do predicado *evolucao_IncImp*.

```
?- evolucao_IncImp(utente(1012,josereis,42,famalicao,masculino),utente(1012,josereis,54,fam
alicao,masculino),utente(1012,josereis,idade_desconhecido,famalicao,masculino)).
true .

?- listing(excecao).
:- dynamic excecao/1.

excecao(utente(IU, N, I, C, S)) :-
    utente(IU, N, idade_desconhecido, C, S).
excecao(utente(1013, clarafernandes, 52, viladoconde, feminino)).
excecao(utente(1013, marafernandes, 52, viladoconde, feminino)).
excecao(utente(IU, N, I, C, S)) :-
    utente(IU, N, I, cidadeinterdito, S).
excecao(prestador(IP, T, E, N, I, C, S)) :-
    prestador(IP,
              T,
              especialidade_desconhecido,
              N,
              I,
              C,
              S).
excecao(prestador(2010, enfermeiro, psiquiatria, lauramiquelina, publico, braga, feminino))
.
excecao(prestador(2010, enfermeiro, psicologia, lauramiquelina, publico, braga, feminino)).

excecao(prestador(IP, T, E, N, I, C, S)) :-
    prestador(IP,
              T,
              E,
              N,
              I,
              cidadeinterdito,
              S).
excecao(ato(IA, D, IU, IP, D, C)) :-
    ato(IA, D, IU, IP, descricao_desconhecido, C).
excecao(ato(3010, 190221, 1011, 2007, consulta, CD)) :-
    CD>=12,
    CD=<22.
excecao(utente(1012, josereis, 42, famalicao, masculino)).
excecao(utente(1012, josereis, 54, famalicao, masculino)).

true.
```

Figura 14. Resultado da utilização do predicado *evolucao_IncImp*.

3.8.4.6. Predicado *evolucao_ImpInc*

O predicado *evolucao_ImpInc* permite transformar conhecimento do tipo impreciso em conhecimento do tipo incerto.

Assim sendo, este abrange quatro argumentos, *Cnovo*, *Cnovo_x*, *Cpassado1*, *Cpassado2*. Os dois primeiros argumentos correspondem ao conhecimento incerto, sendo que o primeiro argumento se refere ao novo facto a adicionar e o segundo à sua exceção correspondente. Quanto aos dois últimos argumentos, estes dizem respeito ao conhecimento impreciso presente na base de conhecimento.

Através do predicado *remover* elimina-se os termos antigos e o predicado *registar* permite a inserção dos termos que estão relacionados com o conhecimento incerto. O predicado *si* impõe que os termos passados sejam desconhecidos.

```
% Extensão do predicado que permite a evolucao do conhecimento
impreciso para incerto: Cnovo, Cnovo_x, Cpassado1, Cpassado2 ->
{V,F}.
evolucao_ImpInc(Cnovo,Cnovo_x,Cpassado1,Cpassado2):-
      si(Cpassado1,desconhecido),
      si(Cpassado2,desconhecido),
      remover(excecao(Cpassado1)),
      remover(excecao(Cpassado2)),
      registar(Cnovo),
      registar(excecao(Cnovo_x)).
```

Na Figura 15 está um exemplo de utilização do predicado *evolucao_ImpInc*.

```
?- evolucao_ImpInc(utente(1013, desconhecido, 52, viladoconde, feminino),utente(IU,desconhe
cido,I,C,S),utente(1013, clarafernandes, 52, viladoconde, feminino),utente(1013, marafernan
des, 52, viladoconde, feminino)).
true .

?- listing(utente).
:- dynamic utente/5.

utente(1001, laravaz, 22, braga, feminino).
utente(1002, marianalindo, 20, vianadocastelo, feminino).
utente(1003, tiagonovais, 21, fafe, masculino).
utente(1004, luisvaz, 47, braga, masculino).
utente(1005, manuelmartins, 86, barcelos, masculino).
utente(1006, mariasilva, 3, famalicao, feminino).
utente(1007, luisparente, 44, braga, masculino).
utente(1008, teresaferreira, 5, porto, feminino).
utente(1009, nunocosta, 29, viladoconde, masculino).
utente(1010, catarinacameira, 15, vianadocastelo, feminino).
utente(1011, paulolopes, 58, porto, masculino).
utente(1012, josereis, idade_desconhecido, famalicao, masculino).
utente(1014, claudiareis, 17, cidadeinterdito, feminino).
utente(1013, desconhecido, 52, viladoconde, feminino).

true.
```

Figura 15. Resultado da utilização do predicado *evolucao_ImpInc*.

4. Conclusões e Sugestões

Com a elaboração deste trabalho foi possível aplicar, num caso prático, os conceitos aprendidos na Unidade Curricular de Inteligência Artificial em Engenharia Biomédica, sobre os conhecimentos do tipo perfeito (positivo e negativo) e imperfeito (impreciso, incerto e interdito).

Através da programação em lógica estendida incluiu-se na representação do conhecimento, dois tipos de negação, a negação por falha na prova e a negação forte, passando assim ser possível distinguir situações falsas de situações para as quais não existe prova de que sejam verdadeiras. Foram também introduzidas regras para a formalização do Pressuposto do Mundo Fechado, passando a considerar que todo o conhecimento que não existe mencionado é considerado falso.

Com o intuito de assegurar a consistência da base de conhecimento e de forma a garantir a coerência estrutural e lógica do programa implementado, impedindo a inserção de conhecimento repetido, foram criados invariantes para a inserção e remoção de conhecimento.

Após a realização deste trabalho, considera-se que todos os objetivos estabelecidos foram satisfeitos, nomeadamente com a realização de predicados que possibilitam o registo de novo conhecimento, a sua manipulação e atualização. Para além disto, com os sistemas de inferência desenvolvidos, foi possível testar qualquer um dos tipos de conhecimento implementados no sistema, como era pretendido.

Ainda assim, conclui-se que há margem para melhorias, nomeadamente, através da introdução de mais conhecimento do tipo perfeito e imperfeito para aumentar a diversidade da base de conhecimento.

5. Referências Bibliográficas

[1] ANALIDE , César, NEVES, José
"Representação de Informação Incompleta"

Anexo A. Base de Conhecimento

Esta secção apresenta os factos que constituem a base de conhecimento construída para a elaboração deste trabalho.

Conhecimento perfeito positivo

```
% Extensão do predicado utente: ID Utente, Nome, Idade, Cidade, Sexo -> {V,F,D}
utente(1001, laravaz, 22, braga, feminino).
utente(1002, marianalindo, 20, vianadocastelo, feminino).
utente(1003, tiagonovais, 21, fafe, masculino).
utente(1004, luisvaz, 47, braga, masculino).
utente(1005, manuelmartins, 86, barcelos, masculino).
utente(1006, mariasilva, 3, famalicao, feminino).
utente(1007, luisparente, 44 , braga, masculino).
utente(1008, teresaferreira, 5, porto, feminino).
utente(1009, nunocosta, 29, viladoconde, masculino).
utente(1010, catarinacameira, 15, vianadocastelo, feminino).
utente(1011, paulolopes, 58, porto, masculino).


% Extensão do predicado prestador: Id Prestador, Tipo, Especialidade, Nome,
Instituição, Cidade e Sexo -> {V,F}
prestador(2001,medico,cardiologia,ruialves,cuf,braga,masculino).
prestador(2002,enfermeiro,pediatria,anasilva,trofasaude,porto,feminino).
prestador(2003,medico,medicinainterna,teresaalves,publico,vianadocastelo,femin
ino).
prestador(2004,enfermeiro,oftalmologia,goretiparente,publico,vianadocastelo,
feminino).
prestador(2005,medico,pneumologia,luismartins,cuf,braga,masculino).
prestador(2006,medico,dentaria,jorgeamorim,trofasaude,porto, masculino).
prestador(2007,medico,dermatologia,diogopereira,trofasaude,braga,masculino).
prestador(2008,medico,pediatria,rosasilva,publico,braga, feminino).


% Extensão do predicado ato: Id Ato, Data, ID Utente, ID Prestador, Descrição,
Custo-> {V,F,D}
ato(3001, 051121, 1001, 2001, consulta,20).
ato(3002, 080921, 1008, 2002, curativo,5).
ato(3003, 080921, 1006, 2008, consulta,15).
ato(3004, 070520, 1003, 2003, internamento,50).
ato(3005, 070520, 1007, 2004, exame,16).
ato(3006, 231120, 1002, 2005, consulta,21).
ato(3007, 010921, 1010, 2006, limpezadentaria,34).
```

```
ato(3008, 050720, 1004, 2004, consulta,8).
```

Conhecimento perfeito negativo

```
% Formalização do PMF
-utente(IU,N,I,C,S) :-
            nao(utente(IU,N,I,C,S)),
            nao(excecao(utente(IU,N,I,C,S))).
-utente(1015,josemaria,45,lisboa,masculino).


-prestador(IP,T,E,N,I,C,S) :-
            nao(prestador(IP,T,E,N,I,C,S)),
            nao(excecao(prestador(IP,T,E,N,I,C,S))).

-ato(IA,D,IU,IP,D,C):-
                  nao(ato(IA,D,IU,IP,D,C)),
                  nao(excecao(ato(IA,D,IU,IP,D,C))).
```

Conhecimento nulo incerto

```
utente(1012, josereis, idade_desconhecido, famalicao, masculino).

excecao(utente(IU,N,I,C,S)):-
            utente(IU,N,idade_desconhecido,C,S).


prestador(2009, enfermeiro, especialidade_desconhecido, carlossa,publico,
vianadocastelo, masculino).

excecao(prestador(IP,T,E,N,I,C,S)):-
            prestador(IP,T,especialidade_desconhecido,N,I,C,S).


ato(3009, 180621, 1005, 2001, descricao_desconhecido,13).

excecao(ato(IA,D,IU,IP,D,C)):-
            ato(IA,D,IU,IP,descricao_desconhecido,C).
```

Conhecimento nulo impreciso

```
excecao(utente(1013, clarafernandes, 52, viladoconde, feminino)).
excecao(utente(1013, marafernandes, 52, viladoconde, feminino)).


excecao(prestador(2010,enfermeiro,psiquiatria,lauramiquelina,publico,braga,
feminino)).
excecao(prestador(2010,enfermeiro,psicologia,lauramiquelina,publico,braga,
feminino)).

excecao(ato(3010, 190221, 1011, 2007, consulta,CD)):-
                    CD >= 12,
                    CD =< 22.
```

Conhecimento nulo interdito

```
utente(1014, claudiareis, 17, cidadeinterdito, feminino).

interdito(cidadeinterdito).

excecao(utente(IU,N,I,C,S)):-
             utente(IU,N,I,cidadeinterdito,S).

+utente(IU,N,I,C,S)::(solucoes(C,(utente(1014, claudiareis, 17, C, feminino),
                    nao(interdito(C))),L),
                    comprimento(L,R), R==0).


prestador(2011, medico, dentaria, ivonerosas, cuf, cidadeinterdito, feminino).

interdito(cidadeinterdito).

excecao(prestador(IP,T,E,N,I,C,S)):-
             prestador(IP,T,E,N,I,cidadeinterdito,S).

+prestador(IP,T,E,N,I,C,S)::(solucoes(C,(prestador(2011,   medico,   dentaria,
ivonerosas, cuf, C, feminino),
             nao(interdito(C))),L),
             comprimento(L,R), R==0).
```

Anexo B. Exemplos de aplicação do meta-predicado *siC*

Esta secção apresenta exemplos de resultados relativos ao teste do meta-predicado *siC*, obtidos através do interpretador SWI.

Na Figura B1 está representado um exemplo da aplicação do meta-predicado *siC*, que resulta da disjunção da conjunção de duas questões verdadeiras com a disjunção de duas questões desconhecidas, dando obviamente verdadeiro, pois, a disjunção de verdadeiro e desconhecido é verdadeiro.

```
?- siC((((utente(1001, laravaz, 22, braga, feminino)) e (utente(1012, josereis, idade_desconhecido, famalicao, masculino))) ou ((utente(1012, josereis,5 , famalicao,
masculino)) ou (prestador(2010,enfermeiro,psiquiatria,lauramiquelina,publico,braga, feminino)))),R).
R = verdadeiro ,
```

Figura B1. Exemplo de resposta verdadeira com o meta-predicado *siC*.

Na Figura B2, estão representadas as mesmas questões, mas em vez de ser a disjunção da conjunção de duas questões verdadeiras com a disjunção de duas questões desconhecidas, é a conjunção. Como consequência, a resposta é desconhecida.

```
?- siC((((utente(1001, laravaz, 22, braga, feminino)) e (utente(1012, josereis, idade_desconhecido, famalicao, masculino))) e ((utente(1012, josereis,5 , famalicao, m
asculino)) ou (prestador(2010,enfermeiro,psiquiatria,lauramiquelina,publico,braga, feminino)))),R).
R = desconhecido ,
```

Figura B2. Exemplo de resposta desconhecida com o meta-predicado *siC*.

Por último, na Figura B3, está representada a conjunção entre uma questão falsa e uma questão desconhecida, sendo o resultado falso.

```
?- siC((prestador(2012,medico,pneumologia,saracosta,publico,porto, feminino) e ato(3009, 180621, 1005, 2001, consulta,13)),R).
R = falso ,
```

Figura B3. Exemplo de resposta falsa com o meta-predicado *siC*.