Universidade do Minho

Trabalho Prático 2

Programação em Lógica Estendida e Conhecimento Imperfeito

Disciplina de Programação em Lógica, Conhecimento e Raciocínio

Mestrado Integrado em Engenharia Biomédica

Ano Letivo de 2018/2019

Docente

Professor César Analide

Realizado por

Ana Duarte, A74407

Ângela Gonçalves, A76542

Diogo Bessa, A75544

Entregue em

16 de novembro de 2018

Sumário

A utilização do *software* de programação PROLOG permite desenvolver bases de conhecimento de acordo com o conceito e as regras da lógica matemática. Além de permitir a introdução coerente de dados, é também possível estender o conceito de forma a abarcar o conhecimento imperfeito, aquele para o qual não se dispõe da informação completa. O objetivo da realização do presente trabalho é demonstrar que as situações de conhecimento imperfeito podem ser tratadas de forma eficaz e que é possível adicionar-se um valor lógico de desconhecido referente ao conhecimento que não se sabe se é verdadeiro ou falso.

Para tal, é elaborado um programa em PROLOG que permite a manipulação de uma base de conhecimento, que inclui o registo e remoção de utentes, prestadores e cuidados de saúde, considerando as situações de registo de valores desconhecidos incertos, imprecisos e/ou interditos. Além disso, pretendeu-se possibilitar a evolução do conhecimento, de forma a permitir alterar conhecimento já inserido e atualizar conhecimento imperfeito, passando a conhecimento perfeito.

Os registos criados podem ser positivos, onde se declara o conhecimento que se sabe ser verdadeiro; negativos, onde se declara o que se sabe ser falso e podem conter valores de conhecimento imperfeito, em que se desconhece o seu real valor.

O presente trabalho desenvolveu-se em quatro etapas principais: na etapa inicial incluíram-se as declarações do conhecimento positivo inicial; na segunda etapa procedeu-se à elaboração dos predicados auxiliares e dos invariantes que permitem a coerência dos dados inseridos/removidos; na terceira fase foram construídas as regras de inserção, exceções e invariantes relativos ao conhecimento imperfeito e, na última fase, foi criado o sistema de inferência para pesquisa do conhecimento com possibilidade de verificação de uma ou mais questões. No caso de serem múltiplas questões, possibilitou-se que a verificação se efetuasse através de conjunções e/ou disjunções entre elas.

Palavras-chave: PROLOG, Conhecimento Positivo, Conhecimento Negativo, Conhecimento Imperfeito, Sistema de Inferência.

Índice

1.Introdução

Com a elaboração do presente trabalho pretendeu-se abordar a utilização de sistemas baseados em lógica matemática para o desenvolvimento de bases de conhecimento. Destacou-se a sua aplicação aos casos de conhecimento imperfeito, uma vez que este tipo de conhecimento também existe e necessita de ser devidamente retratado na base de conhecimento. Como conhecimento imperfeito, foram consideradas as situações de desconhecimento total, parcial ou impossibilidade de conhecimento de um dado parâmetro de uma base de conhecimento. Por outro lado, acrescentaram-se às declarações positivas, aquelas que refletem um conhecimento que se assume como verdadeiro, as declarações negativas, em que se afirma que o conhecimento aí retratado não é verdadeiro.

Para a implementação do conhecimento imperfeito foram realizadas construções declarativas que simulam esse conhecimento, sobretudo através da introdução de exceções, que permitem a representação de valores desconhecidos. Os valores desconhecidos abordados foram os valores incertos, os imprecisos e os interditos. Esta forma de representar o conhecimento imperfeito está associada a uma aproximação mais real às situações com que se deparam os utilizadores das bases de conhecimento. Deste modo, vão-se assumir três valores de saída para o programa: *verdadeiro*, *falso* e *desconhecido*.

Adicionalmente foram efetuados sistemas de inferência que possibilitam a verificação dos registos e alteração executadas.

Este relatório dá seguimento ao anterior trabalho prático – *Trabalho Prático 1: Programação em Lógica e Invariantes* – de implementação de uma base de conhecimento que gere o conhecimento relativos aos utentes, aos prestadores, às instituições e aos cuidados médicos prestados num universo da área de prestação de cuidados de saúde. O *software* utilizado para a sua implementação foi o SICStus PROLOG.

2.Preliminares

No trabalho anterior a única forma de descrever informação falsa era através da sua omissão, ou seja, toda a informação que não mencionada na base de conhecimento é falsa - Pressuposto do Mundo Fechado. De forma a combater esta limitação usou-se a extensão à Programação em Lógica, originado assim 2 negações, a primeira por falha na prova (não) e a segunda uma negação forte (¬), tornando-se possível distinguir casos que são falsos de casos que não existe prova de que sejam verdadeiros.

Esta nova forma de representação de conhecimento assenta nos seguintes pressupostos [Analide]:

- **Pressuposto dos Nomes Únicos**: indica que dois nomes distintos correspondem a dois objetos diferentes. Assim, por exemplo, se dois utentes tivessem o mesmo nome teria de se incluir uma forma de os distinguir para o programa ter um funcionamento correto.

- **Pressuposto do Mundo Aberto:** tudo o que é declarado existe mas prevê-se também a existência de valores não declarados. Assim, só se considera como verdade, o que for explicitamente declarado como verdade. No entanto, o que não tiver sido declarado explicitamente, pode ser considerado como desconhecido ou como falso.

- **Pressuposto do Domínio Fechado:** os únicos objetos que existem são que estão declarados. Quaisquer outros objetos são considerados inexistentes.

Estes pressupostos serão tidos em conta na implementação da base de conhecimento em estudo.

3.Descrição do Trabalho e Análise de Resultados

3.1. Declarações Iniciais

Antes da definição do sistema em estudo e da sua base de conhecimento, é necessário aplicar algumas declarações iniciais, para evitar erros e garantir um funcionamento eficiente do programa.

Deste modo, para se desativarem alguns avisos internos, definiu-se:

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

Além disso, para se declarar a simbologia utilizada nos invariantes e para a representação de conjunções e disjunções de questões, usou-se:

```
:- op(900,xfy,'::').
:- op(300,xfy,ou).
:- op(300,xfy,e).
```

Note-se que o primeiro argumento deste predicado é um valor numérico que se relaciona com as precedências. Assim, de forma a declarar que "e" e "ou" têm igual prioridade, convencionou-se este valor como sendo *300* em ambos os casos.

Por último, devem-se ainda indicar os predicados dinâmicos, que podem ser modificados durante a execução do programa, indicando-se, para cada um, o número de argumentos pelos quais é composto.

```
:- dynamic utente/5.
:- dynamic prestador/5.
:- dynamic instituicao/1.
:- dynamic especialidade/1.
:- dynamic tipo_cuidado/1.
:- dynamic cuidado/5.
:- dynamic instituicao_especialidade/2.
:- dynamic instituição_cuidado/2.
:- dynamic data/3.
:- dynamic excecao/1
:- dynamic '-'/1.
.
```

3.2. Conhecimento Positivo e Negativo

O conhecimento positivo considerado, tal como foi descrito no relatório do *Trabalho Prático 1: Programação em Lógica e Invariantes*, contempla os predicados associados aos utentes, aos prestadores, às instituições, às especialidades, aos tipos de cuidados, aos cuidados prestados e às associações instituição-especialidade e instituição-cuidado.

Sempre que se declara explicitamente um facto como verdadeiro, esse conhecimento é considerado como positivo. Nessa declaração, afirma-se que esse facto existe e que, por consequência, é verdadeiro. Por outro lado, os factos podem indicar que uma dada situação não acontece, ou seja, pode-se afirmar que um determinado facto não existe e que, por isso, é falso. Neste caso, inicia-se a declaração com o carater "-". No caso da base de conhecimento em estudo, considerou-se que todo o conhecimento é falso se não existir prova de que seja verdade e se esse conhecimento não for uma exceção do predicado a que pertence. Para isso, para os predicados *utente*, *prestador*, *instituição*, *especialidade*, *tipo-cuidado*, *cuidado*, *instituição-especialidade* e *instituição-cuidado* foram criadas cláusulas deste tipo. De forma exemplificativa, no caso do predicado *prestador* deve inserir-se:

```
- prestador(IdP, N, E, I, T) :- nao(prestador(IdP, N, E, I, T)), nao(excecao(prestador(IdP, N, E, I, T))).
```

A existência do conhecimento positivo e negativo é fundamental para manter a coerência do conhecimento quando se lida com o Pressuposto do Mundo Aberto. Neste pressuposto, podem considerar-se afirmações que não tenham sido declaradas explicitamente, seja de forma positiva, seja de forma negativa.

A título de exemplo de uma declaração de conhecimento positivo, tem-se:

```
utente(u1, rita_pinto, 41, feminino, braga).
```

Por sua vez, o conhecimento negativo é declarado como se representa no seguinte exemplo:

```
-utente(u11, sonia_ribas, 56, feminino, guimaraes).
```

A inserção deste conhecimento negativo é importante para impedir a inserção de conhecimento positivo igual a esse registo. Para que isto se verifique, é preciso inserir-se um invariante que impeça a inserção de conhecimento positivo que esteja negado na base de conhecimento como, por exemplo:

```
+utente(IdU, N, I, G, M)::(solucoes((IdU, N, I, G, M), -utente(IdU, , I, G, M), S), comprimento(S,L), L==0).
```

Note-se ainda que se permitiu que o conhecimento positivo pudesse evoluir, isto é, que determinados registos inicialmente declarados de uma determinada forma positivamente pudessem ser alterados, sem colocar em causa os invariantes restritivos à sua alteração. No entanto, optou-se por não permitir a evolução do conhecimento negativo

uma vez que se considera preferível a remoção deste tipo de conhecimento, em detrimento da sua atualização.

3.3. Conhecimento Imperfeito

Em situações reais, o conhecimento positivo e/ou negativo não é suficientemente representativo da informação disponível. Uma forma de se ultrapassar essa limitação consiste na introdução do conceito de *Conhecimento Imperfeito*. Este conceito deriva da alteração do Pressuposto do Mundo Fechado para o Pressuposto do Mundo Aberto, onde se considera possível a existência de mais conhecimento, que seja considerado *desconhecido*, para além do que tenha sido declarado expressamente. Este tipo de conhecimento divide-se em três situações distintas, que terão uma forma de tratamento diferenciadas:

- **Conhecimento Imperfeito Incerto**, onde se possibilita que um determinado tipo de valor seja do tipo desconhecido. Por exemplo, pode não se conhecer a morada de um utente em particular;

- **Conhecimento Imperfeito Impreciso**, onde se considera que existe um conjunto de valores possíveis para uma determinada área de conhecimento. A título de exemplo, pode não se conhecer quem foi o prestador de um determinado cuidado, mas saber-se que se tratou ou do prestador p2 ou o do prestador p11, por serem os únicos de serviço na data indicada;

- **Conhecimento Imperfeito Interdito**, onde se verifica que um determinado valor é desconhecido e se impossibilita o seu conhecimento. Por exemplo, por questões de privacidade, pode não ser possível revelar-se o nome de um determinado utente.

3.3.1. Conhecimento Imperfeito Incerto (Tipo I)

O conhecimento imperfeito incerto corresponde às situações em que se desconhece pelo menos um valor de uma determinada afirmação. Como referido, um utente poderá ter a sua morada como *desconhecida* e, devido a esse facto, a inserção do conhecimento desse utente fica incompleta, ou seja, o conhecimento é imperfeito e incerto porque se desconhece um dos campos do predicado.

Para se introduzir um conhecimento deste tipo, deve-se utilizar a seguinte metodologia:

- Inserir-se o conhecimento positivo detalhado e atribuir ao valor do campo que é desconhecido um nome como *incerto1*. É importante notar que esta designação deve ser diferenciada de outros registos que sejam também de valor desconhecido, para possibilitar que o conhecimento a eles associado possa evoluir de forma distinta;

- Introduzir-se a exceção aplicável nas situações em que existe, pelo menos, um parâmetro de valor desconhecido. A introdução da exceção visa criar a possibilidade de se adicionar o valor lógico *desconhecido* aos valores lógicos já existentes – *verdadeiro* e *falso*.

Assim, de um modo simples, algo é *verdadeiro* se for declarado positivamente; *falso* se não for declarado e não estiver associado a nenhuma exceção ou se tiver sido declarado negativamente e *desconhecido* se não for nem verdadeiro nem falso e tiver associado a uma exceção.

3.3.2. Conhecimento Imperfeito Impreciso (Tipo II)

O conhecimento imperfeito impreciso está associado aos valores desconhecidos e que só podem assumir um dos valores existentes de um conjunto de possíveis valores. Por exemplo, pode saber-se que um determinado utente pagou 30 ou 80 pelo cuidado recebido, e não se conhecer ao certo qual dos dois valores é o correto.

A forma de se considerarem estas situações segue as seguintes etapas:

- Introduzir-se a(s) exceção(ões), de forma concreta, para os registos cujo conhecimento seja impreciso.

3.3.3. Conhecimento Imperfeito Interdito (Tipo III)

O conhecimento imperfeito interdito retrata as situações em que, além de se desconhecer o valor de um determinado campo, também se impossibilita o conhecimento futuro desse valor. Dito de outro modo, além de desconhecido, também é interdito o conhecimento de uma determinada área da base de conhecimento.

- Inserir-se o conhecimento positivo detalhado e atribuir ao valor do campo em que o seu conhecimento é interdito um nome como *interdito1*.

- Introduzir-se a exceção aplicável nas situações em que existe, pelo menos, um parâmetro de valor interdito.

- Declarar que o valor *interdito1* é interdito, por exemplo, através de *nulo(interdito1)*. Esta afirmação possibilita a existência de um invariante que impeça, no futuro, o preenchimento deste parâmetro.

3.3.4. Predicados de Conhecimento Imperfeito

Predicado Utente – Tipo I

Para a simulação do conhecimento imperfeito incerto foi considerado que o utente u8 não tem a sua morada atualizada. Assim, é necessário registar-se esse utente, com o campo relativo à morada definido como sendo do tipo incerto. Além disso, é ainda preciso indicar-se a cláusula que declara que esse registo se trata de uma exceção, caso contenha

o parâmetro *incerto1* associado à morada. Esta cláusula permitirá que as moradas testadas neste caso em específico sejam retornadas como desconhecidas e não como falsas. Por último, é necessário introduzir-se também um invariante que possibilite a atualização futura do conhecimento imperfeito. Para tal, executaram-se as seguintes condições:

utente(u8, fernando_alves, 24, masculino, incerto1).

excecao(utente(IdU, N, I, G, M)):-utente(IdU, N, I, G, incerto1).

+(utente(u8, fernando_alves, 24, masculino, M))::(utente(u8, fernando_alves, 24, masculino, incerto1), remocao(utente(u8, fernando_alves, 24, masculino, incerto1))).

Justificação: Na primeira cláusula regista-se o utente com os dados conhecidos e define-se como "incerto1" o valor desconhecido. Trata-se da inserção simples de conhecimento positivo. Na segunda cláusula define-se a exceção para o termo "incerto1", isto é, define-se que, sempre que esse termo existir, existe também uma exceção que será utilizada na definição do valor "desconhecido" para além dos valores de "verdadeiro" e "falso". A última cláusula é um invariante que define as condições em que se poderá atualizar o conhecimento imperfeito relativo ao registo do utente u8. Assim, quando se pretender atualizar o utente com um registo (u8, fernando_alves, 24, masculino, M), em que M pode assumir qualquer valor, verifica-se se o registo existente tem o valor de "incerto1" na morada. Caso tenha esse valor, procede-se à remoção desse registo aceitando-se assim a inserção do novo registo que o irá substituir.

Simulação:

```
| ?- evolucao_imp(utente(u8, fernando_alves, 24, masculino, incerto1), utente(u
8, fernando_alves, 24, masculino, braga)).
yes
```

Predicado Utente – Tipo II

Na base utente também poderá existir conhecimento imperfeito impreciso. Considerou-se para esta situação que o utente u9 é menor de idade, mas não se sabe em concreto a sua idade real. Foram executados os seguintes predicados caracterizadores desta situação:

excecao(utente(u9, claudia_lopes, Menor, feminino, guimaraes)):-Menor>=0, Menor<18.

+(utente(u9, claudia_lopes, M, feminino, guimaraes))::(excecao(utente(u9, claudia_lopes, M, feminino, guimaraes))).

Justificação: Nesta situação basta definir a exceção ao registo do utente u9, inserindo os dados conhecidos do utente e colocando o valor "Menor" no campo referente à sua idade. Esta exceção só é validada se o valor "Menor" estiver entre 0 e 17 anos, restringindo-se assim os valores possíveis da idade. Por sua vez, a segunda cláusula permite a atualização do valor M, relativo à

idade do utente u9, na condição da existência de uma exceção válida. Note-se que a exceção já limita o valor da idade e, por isso, só serão aceites valores que se enquadrem em situações de menoridade. Note-se ainda que, o valor "Menor" inicia-se por maiúscula e, por isso, só será exceção nas situações em que for inserido algum valor dentro dos limites definidos.

Simulação:

```
| ?- evolucao_imp(utente(u9, claudia_lopes, impreciso1, feminino, guimaraes), u
tente(u9, claudia_lopes, 10, feminino, guimaraes)).
```

Nesta situação, como não é declarado positivamente o utente u9, quando se efetua o comando *listing*, este utente não é listado. Uma forma de se ultrapassar isso, é executar as condições do conhecimento impreciso do seguinte modo:

utente(u9, claudia_lopes, impreciso1, feminino, guimaraes).

excecao(utente(u9, claudia_lopes, Menor, feminino, guimaraes)):-Menor>=0, Menor<18.

+(utente(u9, claudia_lopes, M, feminino, guimaraes))::(utente(u9, claudia_lopes, impreciso1, feminino, guimaraes), excecao(utente(u9, claudia_lopes, M, feminino, guimaraes)), remocao(utente(u9, claudia_lopes, impreciso1, feminino, guimaraes))).

Esta forma de se inserir conhecimento aproxima-se mais das condições do Tipo I e permite uma melhor visualização da realidade da base de conhecimento. Esta alternativa será justificada nas situações em que se simulará a existência simultânea de conhecimento imperfeito do Tipo I e do Tipo II.

Predicado Utente – Tipo III

Nas situações em que algum dos dados de um registo está impedido de ser conhecido, terá de se definir um conjunto de cláusulas que permitam o registo com este tipo de conhecimento imperfeito. Além disso, é necessária a inclusão de uma exceção para efeitos de atribuição do valor lógico "desconhecido" e a inserção de um invariante que impeça, no futuro, a atualização do conhecimento em falta. Uma forma de se efetuar essas condições é a que se discrimina de seguida:

utente(u10, embaixada1, 48, masculino, lisboa).

excecao(utente(IdU, N, I, G, M)):-utente(IdU, embaixada1, I, G, M).

interdito(embaixada1).

+utente(IdU, N, I, G, M)::(solucoes(N, (utente(u10, N, 48, masculino, lisboa), nao(interdito(N))),S), comprimento(S, R), R==0).

Justificação: Inicialmente insere-se o conhecimento positivo relativo ao utente u10 em que, no nome se identifica como "embaixada1". De seguida, na segunda cláusula, cria-se a exceção genérica que considera como exceção todos os registos que contenham "embaixada1" no nome. Adicionalmente, é colocado o predicado interdito(embaixada1), inserindo assim um novo parâmetro associado a "embaixada1". Este parâmetro será utilizado para impedir, no futuro, qualquer atualização deste valor. Na última cláusula é colocado o invariante que impossibilita a inserção de novos utentes cujos valores sejam os que estão associados ao utente u10. Isto é conseguido pesquisando-se todos os registos que contenham os dados do utente u10, qualquer que seja o seu nome e, em simultâneo, verifica a negação do valor lógico de interdito(nome). Se o nome for "embaixada1", o valor lógico de interdito(N) é verdadeiro pelo que, a sua negação é falso, logo o conjunto de soluções S é vazio. Se, por outro lado, o nome for diferente de "embaixada1", o valor lógico de não(interdito(N)) é verdade e S tem um registo. Neste caso, a parte final do invariante "comprimento(S, R), R==0" obriga a que S não contenha elementos, ou seja não permite inserir nenhum nome diferente de "embaixada1" verificando-se assim o que se pretende: ter um conhecimento imperfeito interdito.

Simulação:

```
| ?- evolucao_imp(utente(u10, embaixada1, 48, masculino, lisboa), utente(u10, j
ohn, 48, masculino, lisboa)).
no
_
```

Predicado Prestador – Tipo I

De uma forma semelhante ao conhecimento perfeito incerto do predicado utente foi criado este tipo de conhecimento para a base prestador. Neste caso, foi considerado que, além de não se saber o nome do prestador p15, sabe-se que não se trata do nome john_smith. Para impor-se esta restrição é preciso declarar-se esta cláusula na forma de conhecimento negativo. O conjunto de cláusulas que caracterizam este conhecimento imperfeito são as que se enumeram de seguida:

-prestador(p15, john_smith, medicina_dentaria, hospital_luz, enfermeiro).

prestador(p15, incerto2, medicina_dentaria, hospital_luz, enfermeiro).

excecao(prestador(IdP, N, E, I, T)):-prestador(IdP, incerto2, E, I, T).

+(prestador(p15, N, medicina_dentaria, hospital_luz, enfermeiro))::(prestador(p15, incerto2, medicina_dentaria, hospital_luz, enfermeiro), nao(-prestador(p15, N, medicina_dentaria, hospital_luz, enfermeiro)), remocao(prestador(p15, incerto2, medicina_dentaria, hospital_luz, enfermeiro))).

Justificação: Estas cláusulas são semelhantes às justificadas na base utente com conhecimento imperfeito do Tipo I. A principal diferença reside na primeira cláusula que impõe o conhecimento negativo, iniciado pelo símbolo "-", que indica que esse registo é falso, ou seja, que o prestador

p15 não se chama john_smith. Outras das diferenças está no invariante utilizado que, além de restringir que o prestador p15 com o nome "incerto2" exista, impõe ainda que o novo registo não tenha o valor igual ao do conhecimento negativo declarado, removendo-se, neste caso, o conhecimento imperfeito inicial.

Predicado Prestador – Tipo II

Para o predicado prestador foi também definido conhecimento imperfeito do Tipo II. A diferença relativamente ao do predicado utente é que será considerado um conjunto de apenas dois valores possíveis para o tipo de profissional que, neste caso, está associado ao prestador p16. Uma forma de se proceder para criar este tipo de conhecimento é:

```
excecao(prestador(p16, rui_fonseca, ortopedia, hospital_sta_luzia, Tipo)):-Tipo=enfermeiro.

excecao(prestador(p16, rui_fonseca, ortopedia, hospital_sta_luzia, Tipo)):-Tipo=tecnico.

+(prestador(p16, rui_fonseca, ortopedia, hospital_sta_luzia, T))::(excecao(prestador(p16,
rui_fonseca, ortopedia, hospital_sta_luzia, T))).
```

Justificação: A forma de se proceder é semelhante à do predicado utente com conhecimento imperfeito impreciso e cuja principal diferença reside no facto de se necessitar de duas exceções. Assim, serão criadas uma exceção para cada possível valor que pode assumir o campo "Tipo". A forma de permitir atualizações também é análoga à que foi descrita no predicado utente.

Predicado Cuidado – Tipo I + Tipo II

Este tipo de linguagem de programação permite a inserção de conhecimento imperfeito de diversos tipos, necessitando, para isso, de cláusulas mais complexas. No caso do predicado do cuidado foram construídas cláusulas que permitissem em simultâneo o conhecimento imperfeito do Tipo I e do Tipo II, tal como se representa de seguida:

```
cuidado(data(30, 04, 18), u1, impreciso2, rinectomia, incerto3).

excecao(cuidado(data(D, M, A), IdU, IdP, Desc, C)):-
cuidado(data(D, M, A), IdU, impreciso2, Desc, incerto3), IdP==p2.

excecao(cuidado(data(D, M, A), IdU, IdP, Desc, C)):-
cuidado(data(D, M, A), IdU, impreciso1, Desc, incerto3), IdP==p11.

+cuidado(data(30, 04, 18), u1, IdP, rinectomia, C)::(cuidado(data(30, 04, 18), u1,
impreciso1, rinectomia, incerto3), excecao(cuidado(data(30, 04, 18), u1, IdP, rinectomia,
C)), remocao(cuidado(data(30, 04, 18), u1, impreciso1, rinectomia, incerto3))).
```

Justificação: Inicialmente regista-se o conhecimento positivo relativo ao cuidado com o id do prestador "impreciso2" e o custo como "incerto3". Posteriormente, criam-se duas exceções das duas possibilidades de valores de "impreciso2", mantendo-se o custo como "incerto3". O invariante permite a atualização destes valores desconhecidos desde que exista o registo declarado inicialmente, além de uma exceção associada à inserção dos novos valores para o id do prestador e para o custo. Caso se verifiquem estas restrições, o registo inicial é removido.

3.4. Sistema de Inferência

O sistema de inferência permite incluir o valor lógico *desconhecido* para caracterizar as situações de conhecimento imperfeito, sejam elas incertas, imprecisas, interditas ou uma combinação de vários tipos distintos. Desta forma, o resultado a uma questão pode ser expresso na forma de *verdadeiro*, *falso* ou *desconhecido*. Assim, foi definido o meta-predicado si do seguinte modo:

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

si(Questao,verdadeiro):- Questao.

si(Questao,falso):- -Questao.

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

Numa primeira cláusula afirma-se que uma questão é verdadeira se foi declarada positivamente, isto é, se o conhecimento estiver contido, de forma afirmativa, na base de conhecimento, essa questão retornará *verdadeiro*. Na cláusula seguinte, é considerado *falso* um facto que tenha sido declarado negativamente, ou seja, é falso se se afirmou que esse facto é negativo. O valor desconhecido surge na terceira condição como um facto que não foi considerado nem como verdadeiro nem como falso, ou seja, é *desconhecido* quando não é nem verdade nem falso.

Com o intuito de se testar o sistema de inferência desenvolvido, efetuou-se um conjunto de questões que suportam as simulações desenvolvidas. Este conjunto de questões têm como base a idade do utente u9, que é imprecisa, em que apenas se sabe que está contida no intervalo dos 0 aos 17 anos.

- **Questão 1:** O registo do utente u9 tem no campo idade o valor de *impreciso1*?

```
| ?- si(utente(u9, claudia_lopes, impreciso1, feminino, guimaraes),S).
S = verdadeiro ?
yes
```

- **Questão 2:** O utente u9 tem 30 anos?

```
| ?- si(utente(u9, claudia_lopes, 30, feminino, guimaraes),S).
S = falso ?
yes
```

- **Questão 3:** O utente u9 tem 16 anos?

```
| ?- si(utente(u9, claudia_lopes, 16, feminino, guimaraes),S).
S = desconhecido ?
yes
```

Nos três casos testados, constata-se que os valores devolvidos são os esperados, tendo em conta que apenas se sabe que este utente é menor de idade.

Uma das limitações do sistema de inferência desenvolvido reside no facto de apenas poder ser testada uma questão de cada vez. De forma a supri-la e para se incluir também a possibilidade de se responder a conjunções ou disjunções de questões, é necessário implementar-se um predicado adicional. Inicialmente, e de modo a auxiliar na sua construção, elaboraram-se as tabelas lógicas (Tabela 1) que se aplicam a cada uma das situações – conjunções e disjunções de questões.

Tabela 1 - Valores lógicos associados a conjunções e disjunções

Conjunção (e)	V	D	F
V	V	D	F
D	D	D	F
F	F	F	F

Disjunção (ou)	V	D	F
V	V	V	V
D	V	D	D
F	V	D	F

De uma forma resumida, tem-se que:

- Numa conjunção, *Falso* é um elemento absorvente, isto é, basta que, pelo menos, uma das questões seja falsa para que a resposta seja do tipo falso;
- Numa conjunção, no caso de uma das questões ser verdadeira e a outra desconhecida, o resultado é desconhecido;
- Numa disjunção, *Verdadeiro* é um elemento absorvente, isto é, independentemente do valor de verdade da outra questão, a resposta é verdadeira;
- Numa disjunção, quando uma das questões é falsa e outra desconhecida, o resultado é desconhecido;
- No caso de serem duas questões desconhecidas, o resultado é desconhecido em ambas as situações;
- Perante um cenário com duas questões verdadeiras ou duas questões falsas, o resultado é verdadeiro e falso, respetivamente, quer se trate de uma conjunção, quer seja uma disjunção;

Tendo em conta a Tabela 1, para se realizar uma extensão do sistema de inferência para a inclusão em simultâneo de duas questões, foi criado o predicado si_2Q. Este predicado foi construído do seguinte modo:

```
Extensao do meta-predicado si_2Q: Questao, Resposta -> {V,F}

% ---------- CONJUNCAO ----------

si_2Q(Q1 e Q2,verdadeiro):- si(Q1,verdadeiro), si(Q2,verdadeiro).

si_2Q(Q1 e Q2,falso):-si(Q1,falso).

si_2Q(Q1 e Q2,falso):-si(Q2,falso).

si_2Q(Q1 e Q2,desconhecido):-

(nao(si_2Q(Q1 e Q2,verdadeiro)), nao(si_2Q(Q1 e Q2,falso))).

% ---------- DISJUNCAO ----------

si_2Q(Q1 ou Q2,falso):-si(Q1,falso), si(Q2,falso).

si_2Q(Q1 ou Q2,verdadeiro):-si(Q1,verdadeiro).

si_2Q(Q1 ou Q2,verdadeiro):-si(Q2,verdadeiro).

si_2Q(Q1 ou Q2,desconhecido):-

(nao(si_2Q(Q1 ou Q2,verdadeiro)), nao(si_2Q(Q1 ou Q2,falso))).
```

Para a sua validação funcional, foram incluídos os operadores lógicos “e” e “ou”, de modo a interligar as duas questões e indicar se se trata de uma conjunção (e) ou de uma disjunção (ou).

No caso da conjunção, o resultado será verdadeiro se ambas as questões forem verdadeiras, tal como indicado na primeira cláusula. Nos segundo e terceiro predicados considera-se que Q1 e Q2 é falso se uma das questões (Q1 ou Q2) for falsa, ou seja, basta que uma delas seja falsa para o resultado ser falso. Define-se no quarto predicado o resultado desconhecido para Q1 e Q2 sempre que o resultado não é verdadeiro nem é falso.

Para a disjunção procede-se de uma forma semelhante, em que Q1 e Q2 é verdadeiro se, pelo menos, uma das questões for verdadeira; é falso se ambas forem falsas e é desconhecido nos restantes casos.

A título de exemplo, simularam-se as seguintes combinações de questões de forma a testar o funcionamento deste sistema de inferência em estudo.

- **Questão 1:** O prestador p15 chama-se joao_silva e o utente u1 é a rita_pinto?

```
| ?- si_2Q(prestador(p15, joao_silva, medicina_dentaria, hospital_luz, enfermeiro)
 e utente(u1, rita_pinto, 41, feminino, braga),S).
S = desconhecido ?
yes
```

Como todos os dados, além dos ids e dos nomes, estão corretos, é expectável que a primeira parte da pergunta seja desconhecida e a segunda parte verdadeira. Note-se que o registo de p15 tem um nome de valor desconhecido. Desta forma, ao utilizar-se uma conjunção associada a uma questão desconhecida e a outra verdadeira, é previsto um

resultado desconhecido. Tal como se pode observar, o programa retornou o valor *desconhecido* e, por isso, o resultado é o esperado.

- **Questão 2:** O prestador p15 chama-se joao_silva ou o utente u1 é a rita_pinto?

```
| ?- si_2Q(prestador(p15, joao_silva, medicina_dentaria, hospital_luz, enfermeiro)
 ou utente(u1, rita_pinto, 41, feminino, braga),S).
S = verdadeiro ?
yes
```

Tendo em conta o valor *desconhecido* da primeira questão relativa ao prestador e o valor de *verdadeiro* da questão associada ao utente, o resultado retornado é *verdadeiro*, uma vez que, na disjunção, basta que uma das afirmações seja verdade para que o resultado também seja também verdadeiro.

- **Questão 3:** Quais os utentes que realizaram um raio-x no dia 20 de maio de 2018 e quais o prestadores associados e custos incorridos?

```
| ?- si_2Q(utente(R,N,I,S,M) e cuidado(data(20,05,18),R,P,raio_x,C),verdadeiro)

R = u1,
N = rita_pinto,
I = 41,
S = feminino,
M = braga,
P = p1,
C = 30 ?
```

Do resultado, conclui-se que, nesse dia, a utente rita_pinto foi atendida pelo prestador p1 e efetuou um raio-x, tendo tido um custo associado de 30.

- **Questão 4:** Quais os utentes que tiveram um cuidado no dia 15 de março de 2011 e quais os prestadores associados?

```
| ?- si_2Q((cuidado(data(15,03,11),U,P,_,_) e prestador(P,I,_,_,_)),R).
U = u7,
P = p11,
I = americo_abreu,
R = verdadeiro ?
yes
```

O resultado mostra que o utente u7 teve um cuidado realizado nessa data pelo prestador p11, chamado americo_abreu.

No caso do predicado si_2Q pode-se aumentar ainda mais a complexidade das questões, fazendo-se associações de grupos de duas questões como, por exemplo:

- **Questão 5:** O prestador p15 é o joao_silva e a utente u1 é a rita_pinto, ou os utentes u7 e u10 são, respetivamente, o fernando_alves e o john_smith?

```
| ?- si_2Q((prestador(p15, joao_silva, medicina_dentaria, hospital_luz, enfermeiro)
 e utente(u1, rita_pinto, 41, feminino, braga)) ou (utente(u7, fernando_alves, 64,
masculino, leiria) e utente(u10, jonh_smith, 48, masculino, lisboa)),S).
S = desconhecido ?
yes
```

Decompondo-se esta questão, tem-se uma situação do género:

- p15 é o joao_silva **e** u1 é a rita_ pinto
  ou
- u7 é o fernando_alves **e** u10 é o john_smith

que se traduz nos valores lógicos (desconhecido e verdadeiro) ou (falso e desconhecido), ou seja, desconhecido ou falso, que corresponde ao valor final de desconhecido, tal como se comprova na simulação.

3.5. Predicados Auxiliares

A construção dos diversos predicados recorre, para a sua implementação, a predicados mais simples e que auxiliam na implementação de predicados mais complexos. Assim, foram criados os seguintes predicados auxiliares:

- **Comprimento,** que calcula o número total de elementos de uma lista;
- **Soluções,** que devolve os valores que respeitam as regras de uma pesquisa efetuada;
- **Não,** que devolve o valor oposto de uma afirmação;
- **Inserir,** que insere um novo registo;
- **Remoção,** que exclui do conhecimento um determinado registo;
- **Testar,** que define a forma de se efetuarem os testes aos invariantes;
- **Registar,** que insere um novo registo que respeite as restrições dos invariantes que lhe são aplicáveis;
- **Remover,** que exclui um registo e que respeita as regras definidas pelos invariantes;

Para além destes predicados, que já foram devidamente detalhados no *Trabalho Prático 1*, foram também adicionados novos predicados que permitem a evolução do conhecimento, seja ele perfeito positivo, perfeito negativo ou imperfeito.

3.5.1 Predicado Auxiliar evolucao_pos

Para a evolução do conhecimento positivo elaborou-se o predicado:

Extensao do predicado evolucao_pos: Rant, Rnovo -> {V,F}

evolucao_pos(Rant, Rnovo):-si(Rant, verdadeiro), remocao(Rant), registar(Rnovo).

Note-se que se optou por permitir alterar um conhecimento positivo para outro conhecimento positivo ou para um conhecimento negativo. A introdução do novo conhecimento remove o conhecimento antigo.

Justificação: A introdução do registo antigo e do novo registo proporciona a verificação, através do sistema de inferência, se o registo antigo é verdadeiro, e, caso seja, remove-o e insere o novo registo. Esta inserção far-se-à de forma a serem testados os invariantes e, só após isso é que se regista o novo conhecimento. Ou seja, ao colocar-se um novo conhecimento não se podem ignorar as restrições como, por exemplo, a colocação de Ids não únicos.

Efetuaram-se as simulações seguintes para se aferir a correta implementação do predicado evolucao_pos. A listagem parcial dos utentes é a que se apresenta:

```
utente(u6, matilde_neves, 50, feminino, porto).
utente(u7, andre_fernandes, 64, masculino, leiria).
utente(u8, fernando_alves, 24, masculino, incerto1).
utente(u9, claudia_lopes, impreciso1, feminino, guimaraes).
```

Simulação 1: substituir o registo u7 de leiria pelo id u8 de braga. Verifica-se que não é concretizada esta evolução porque o registo com id u8 já existe e não se permite a existência de dois ids iguais.

```
| ?- evolucao_pos(utente(u7, andre_fernandes, 64, masculino, leiria),utente(u8,
 andre_fernandes, 64, masculino, braga)).
no
```

Simulação 2: substituir o registo u7 de leiria pelo id u7 de braga. Neste caso a alteração é efetuada.

```
| ?- evolucao_pos(utente(u7, andre_fernandes, 64, masculino, leiria),utente(u7,
 andre_fernandes, 64, masculino, braga)).
yes
```

Comprova-se a correção da alteração pela execução de *listing(utente).*

```
utente(u6, matilde_neves, 50, feminino, porto).
utente(u8, fernando_alves, 24, masculino, incerto1).
utente(u9, claudia_lopes, impreciso1, feminino, guimaraes).
utente(u10, embaixada1, 48, masculino, lisboa).
utente(u7, andre_fernandes, 64, masculino, braga).
```

Simulação 3: substituir o registo de conhecimento positivo u2 de vilareal pelo registo negativo u2 de lisboa. Nota-se que passa a existir um novo conhecimento negativo e que o anterior registo foi eliminado.

```
| ?- evolucao_pos(utente(u2, carlos_moreira, 12, masculino, vilareal),-utente(u
2, carlos_moreira, 12, masculino, lisboa)).
yes
```

A lista dos utentes passa, assim, a ser:

```
| ?- listing(utente).
utente(u3, rui_sousa, 12, masculino, leiria).
utente(u4, maria_tavares, 84, feminino, braga).
utente(u5, ivone_lopes, 36, feminino, lisboa).
utente(u6, matilde_neves, 50, feminino, porto).
utente(u8, fernando_alves, 24, masculino, incerto1).
utente(u9, claudia_lopes, impreciso1, feminino, guimaraes).
utente(u10, embaixada1, 48, masculino, lisboa).
utente(u7, andre_fernandes, 64, masculino, braga).
```

Comprova-se a existência do conhecimento negativo através do sistema de inferência:

```
| ?- si(-utente(u2, carlos_moreira, 12, masculino, lisboa),S).
S = verdadeiro ?
yes
```

Como o registo negativo foi corretamente inserido não é possível registar um utente com os mesmos valores:

```
| ?- registar(utente(u2, carlos_moreira, 12, masculino, lisboa)).
no
```

Esta situação altera-se se se modificar a morada para Braga:

```
| ?- registar(utente(u2, carlos_moreira, 12, masculino, braga)).
yes
```

3.5.2 Predicado Auxiliar evolucao_imp

Foi também construído um predicado que permite a alteração de conhecimento imperfeito de modo a que este passe a conhecimento perfeito. O predicado evolucao_imp que permite esta atualização foi construído através das cláusulas:

Extensao do predicado evolucao_neg: Rant, Rnovo -> {V,F}

evolucao_imp(Rant, Rnovo):-si(Rant, verdadeiro), registar(Rnovo).

Justificação: Este predicado é semelhante, mas não necessita que seja removido o registo antigo porque, nos invariantes, já se considerou a remoção dos registos antigos.

Simulação 1: substituir o desconhecimento incerto1, do utente u8, para braga. Na primeira situação colocou-se como IdU o valor u9 e, como já é um IdU existente não se conseguiu evoluir o conhecimento. registo negativo do prestador p15 para negativo com outro nome de prestador.

```
| ?- evolucao_imp(utente(u8, fernando_alves, 24, masculino, incerto1),
                  utente(u9, fernando_alves, 24, masculino, braga)).
no
```

Na segunda situação manteve-se o Id em u8 e já foi possível inserir a atualização do conhecimento.

```
| ?- evolucao_imp(utente(u8, fernando_alves, 24, masculino, incerto1),
                  utente(u8, fernando_alves, 24, masculino, braga)).
yes
```

A lista dos utentes fica assim do seguinte modo:

```
| ?- listing(utente).
utente(u1, rita_pinto, 41, feminino, braga).
utente(u2, carlos_moreira, 12, masculino, vilareal).
utente(u3, rui_sousa, 12, masculino, leiria).
utente(u4, maria_tavares, 84, feminino, braga).
utente(u5, ivone_lopes, 36, feminino, lisboa).
utente(u6, matilde_neves, 50, feminino, porto).
utente(u7, andre_fernandes, 64, masculino, leiria).
utente(u9, claudia_lopes, impreciso1, feminino, guimaraes).
utente(u10, embaixada1, 48, masculino, lisboa).
utente(u8, fernando_alves, 24, masculino, braga).
```

3.6. Invariantes

No presente trabalho prático mantiveram-se os invariantes que restringem a inserção dos novos registos e que contemplam os aspectos discriminados de seguida.

3.6.1. Predicado Utente

- Não permite inserir utentes com ids que já pertençam à base de conhecimento;
- Não permite inserir utentes com nomes que já pertençam à base de conhecimento;
- Não permite inserir utentes com idades negativas ou superiores a 130 anos;
- Não permite inserir utentes com género diferente de "feminino" ou "masculino";
- Não permite remover utentes que já tenham algum registo em cuidado médico.

3.6.2. Predicado Prestador

- Não permite inserir prestador com ids que já pertençam à base de conhecimento;
- Não permite inserir prestador com nomes que já pertençam à base de conhecimento;
- Não permite inserir prestadores cuja especialidade não seja disponibilizada pela instituição;
- Não permite inserir prestadores de tipo diferente de médico, enfermeiro ou técnico;
- Não permite remover prestadores que já tenham realizado cuidados médicos.

3.6.3. Predicado Especialidade

- Não permite inserir especialidades já existentes;
- Não permite remover especialidades que esteja associada a um prestador.

3.6.4. Predicado Instituição

- Não permite inserir instituições já existentes;
- Não permite remover instituição que esteja associada a um prestador.

3.6.5. Predicado Tipo de Cuidado

- Não permite inserir tipos de cuidado já existentes;
- Não permite remover tipos de cuidado que estejam associados a um cuidado.

3.6.6. Predicado Cuidado

- Não permite inserir cuidados de prestadores nem de utentes que não pertençam à base de conhecimento;
- Não permite inserir cuidados com, simultaneamente, os mesmos Data, IdU, IdP e Descrição;
- Não permite inserir custos negativos;
- O custo dos cuidados prestados a maiores de 65 anos é zero (idosos isentos de pagamento de análises clínicas);
- Os cuidados de rinectomia só podem ser prestados pela especialidade de otorrinolaringologia;
- Os cuidados de densitometria óssea só podem ser prestados pela especialidade de ortopedia.

3.6.7. Predicado Instituição-Especialidade

- Não permite inserir instituições nem especialidades que não pertençam à base de conhecimento;
- Não permite inserir um par instituição-especialidade repetido;
- Não permite remover registos que já tenham cuidados médicos associados-

3.6.8. Predicado Instituição-Cuidado

- Não permite inserir instituições nem tipos de cuidados que não pertençam à base de conhecimento;
- Não permite inserir um par instituição-cuidado repetido;
- Só aceita inserir um cuidado de rinectomia se a instituição correspondente tiver a especialidade de otorrinolaringologia;
- Só aceita inserir um cuidado de densitometria óssea se a instituição correspondente tiver a especialidade de ortopedia;
- Não permite remover registos que já tenham cuidados médicos associados.

3.6.9. Predicado Data

Além dos invariantes anteriores, que já foram discriminados no Trabalho Prático 1 foram considerados os seguintes invariantes associados ao predicado data, de forma a permitir apenas a inserção de datas válidas.

+cuidado(data(D, M, _), _, _, _, _) :: (cuidado(data(D, M, _), _, _, _, _), ((M==1, D<32));

((M==3, D<32));((M==5, D<32));((M==7, D<32));((M==8, D<32));((M==10, D<32));((M==12, D<32)); ((M==2, D<29));((M==4, D<31));((M==6, D<31));((M==9, D<31));((M==11, D<31))).

+cuidado(data(_, _, A), _, _, _, _) :: (cuidado(data(_, _, A), _, _, _, _), A>=0, A<100).

Simulação:

```
| ?- registar(cuidado(data(32, 04, 12), u1, p2, raio_x, 30)).
no
| ?- registar(cuidado(data(28, 04, 12), u1, p2, raio_x, 30)).
yes
```

Quando insere o dia 32 no mês de abril, o registo não pode ser inserido. Por outro lado, esta inserção é válida caso o dia seja o 28.

```
| ?- registar(cuidado(data(28, 04, 01), u1, p2, raio_x, 30)).
yes
| ?- registar(cuidado(data(28, 04, 100), u1, p2, raio_x, 30)).
no
```

Quando insere o ano 01, o registo pode ser inserido. Por outro lado, esta inserção não é caso seja colocado o ano 100.

4. Conclusão

Com a realização deste trabalho constatou-se que os sistemas baseados em lógica matemática possibilitam a inserção de conhecimento perfeito e imperfeito em bases de conhecimento. Além disso, permitem ainda a evolução do conhecimento, de forma a atualizar valores antigos. Esta evolução garante que se possa evoluir o conhecimento positivo e passar o conhecimento imperfeito para perfeito. Verificou-se ainda que é possível construir modelos de conhecimento imperfeito mais complexos e mais próximos da realidade, incluindo-se no mesmo registo dois tipos de conhecimento imperfeito diferentes. Também se observou que se pode implementar um sistema de inferência que permite, de uma forma fácil, solicitar respostas a questões simples ou a várias questões, através de conjunções e/ou disjunções entre elas.

Para um trabalho futuro poderiam ser analisadas cláusulas mais complexas com a inclusão dos três tipos de conhecimento estudados e poderia prever-se também a possibilidade de se evoluir parcialmente conhecimento incerto para impreciso, ou um conhecimento duplo (incerto e/ou impreciso) para uma atualização de apenas um dos valores desconhecidos.

Em suma, observou-se que, com um pequeno conjunto de instruções e um bom domínio da lógica matemática, é possível implementar uma base de conhecimento adequada às situações reais garantir o seu correto funcionamento.

5. Bibliografia

[Analide] Analide, César

"Representação de Informação Completa"

ISLab, Universidade do Minho