Universidade do Minho

Escola de Engenharia

Departamento de Informática

SRCR:TP2-Programação em lógica estendida

Author:
Carlos Gonçalves A77278
Jorge Oliveira A78660
José Ferreira A78452
Ricardo Peixoto A78587

April 20, 2018

.

Abstract

O presente documento aborda os temas da programação em lógica estendida e conhecimento imperfeito e pretende dar continuação aos temas de programação em lógica e invariantes, anteriormente abordados.

À semelhança do documento produzido no exercício anterior, este documento retrata o desenvolvimento de um sistema de representação de conhecimento e raciocínio que incide na área da prestação de cuidados de saúde. Sendo este exercício uma extensão ao exercício anterior, o documento mantém parte do que foi produzido anteriormente. Contudo, aborda adicionalmente lógica estendida, para uma melhor representação do conhecimento, mas também conhecimento imperfeito assim como decisões tomadas na manipulação desse conhecimento.

Contents

List of Figures

List of Tables

.

1 Introdução

Hoje em dia a saúde é uma preocupação para todas as pessoas, sendo que os cuidados médicos têm uma grande influência no bem-estar da população. Como este tema está bastante ativo na sociedade, foi-nos proposto, num modo simplista, desenvolver um sistema de representação de conhecimento e raciocínio capaz de qualificar a prestação de cuidados de saúde, recorrendo à programação em lógica estendida e à representação de conhecimento imperfeito.

Para o desenvolvimento desta base de conhecimento, decidimos considerar três predicados: o Utente, o Prestador e o Cuidado. Como extensão à lógica apresentada no projeto anterior, introduzimos o valor Desconhecido para não só permitir representar situações do dia a dia com mais realismo, mas também para nos possibilitar em certas situações evoluir o conhecimento do nosso sistema. Para representação de conhecimento imperfeito utilizamos três tipos de valores nulos: nulo incerto, nulo impreciso e nulo interdito.

Com o desenvolvimento deste projeto pretendemos criar uma base de conhecimento capaz de registar todos os cuidados prestados num dado local, garantindo integridade e consistência, que permitisse também a inserção de conhecimento parcial ou restrito.

Nas seguintes secções demonstramos como abordamos e tratamos os diversos tipos de conhecimento imperfeito, e também como evoluímos e involuímos a nossa base de conhecimento face às diversas formas de representação de conhecimento, explicando as decisões tomadas em cada situação. E ainda, de que forma se pode obter respostas da base de conhecimento.

2 Preliminares

De seguida apresentaremos os conceitos chave e pressupostos que permitiram a elaboração deste trabalho.

2.1 Programação em Lógica Estendida

A implementação deste trabalho foi feita numa linguagem de programação em lógica, logo, o conhecimento deste tipo de programação foi essencial. Este conhecimento foi obtido através da realização do trabalho prático transato e permitiu-nos perceber qual a melhor maneira de abordar o problema, ou seja, quais as cláusulas, os predicados e os invariantes a criar. Mais uma vez a ferramenta utilizada para manipular esta linguagem foi o *SicStus Prolog.*

2.2 Pressuposto do Mundo fechado

Assumimos que toda a informação que não está representada na base de conhecimento é falsa.

2.3 Pressuposto do Mundo aberto

Assumimos que uma informação só é falsa se estiver representada explicitamente na base de conhecimento como falsa.

2.4 Pressuposto dos nomes únicos

Neste pressuposto assumimos que cada nome identifica inequivocamente um objeto no universo, ou seja, dois nomes diferentes representam obrigatoriamente dois objetos diferentes.

Apesar dos pressupostos do mundo fechado/aberto serem contraditórios, ambos são utilizados simultaneamente na nossa base de conhecimento. Utilizamos o pressuposto do mundo aberto na generalidade do nosso conhecimento e aplicamos o pressuposto do mundo fechado individualmente a cada predicado (fecho transitivo).

2.5 Representação do conhecimento imperfeito

Ao contrário do que se passava no primeiro exercício, neste problema ocorrem várias situações nas quais o conhecimento não é perfeito, ou seja, não é conhecido na totalidade. Para podermos representar também esse conhecimento com falhas de informação tivemos de o categorizar:

- **Conhecimento incerto:** ocorre quando um certo conhecimento é totalmente desconhecido (exemplo: Não se sabe quem é a mãe da Maria)
- **Conhecimento impreciso:** ocorre quando existe um conjunto de possibilidades para um determinado facto (exemplo: O pai da Maria é o Manuel ou o José).
- **Conhecimento restrito:** ocorre quando um facto é e será sempre completamente desconhecido (exemplo: A Maria é órfã e nunca se saberá quem são os pais dela).

Com esta extensão de conhecimento aumentamos agora as possibilidades de resposta na interrogação da base de conhecimento e as respostas válidas agora são:

- **Verdadeiro:** Quando obtemos uma prova para a questão Q.
- **Falso:** Quando temos uma negação forte para a questão.
- **Desconhecido:** Quando não temos uma prova positiva(Q), nem uma negação forte para a questão(-Q).

Como a linguagem de programação lógica utilizada (*Prolog*) é baseada em lógica pura e tem apenas duas respostas(verdadeiro e falso), como esta extensão deixamos de poder utilizar o demonstrador original da linguagem e fomos obrigados a criar um de maneira a incluir a resposta Desconhecido, como especificaremos adiante.

3 Descrição do Trabalho e Análise de Resultados

3.1 Representação de Conhecimento

3.1.1 Utente

Na nossa base de conhecimento o predicado utente representa um indivíduo ao qual é prestado um determinado cuidado, podemos então deduzir, que no mundo real o utente será um cidadão comum. Portanto tem de possuir um nome, mas como um nome não identifica inequivocamente uma pessoa numa atividade é necessário um identificador. Precisamos ainda de um campo com a idade dos cidadãos, pois podemos querer identifica-los por idades, e também da sua morada para que possamos identificar os utentes de diferentes localidades. Para caraterizar a morada de um utente decidimos criar um predicado morada, sendo que esta contém um campo relativo à rua da morada do mesmo, outra relativo à localidade e outro relativo à cidade.

3.1.2 morada

No predicado utente está presente um campo morada. Como a morada de um cidadão é algo complexo decidimos então criar um predicado que nos permitisse ter um maior controlo sobre a morada de um individuo. Assim neste predicado decidimos incluir a rua, local onde reside uma pessoa (neste campo rua já está incluído o número da porta), localidade e cidade.

3.1.3 prestador

Na nossa base de conhecimento o predicado prestador representa um indivíduo que presta um determinado cuidado, podemos então deduzir, que no mundo real o prestador seria um profissional de saúde. Sendo assim tem de possuir um nome, e como o nome de uma pessoa não é suficiente para a identificar unicamente num serviço, foi necessário definir um identificador para cada prestador. Para além destes campos, um prestador deve de possuir uma especialidade e uma instituição onde pratica os cuidados. Todavia, um prestador pode trabalhar em várias instituições, caso assim deseje e seja permitido

3.1.4 cuidado

Na nossa base de conhecimento o predicado cuidado representa um serviço médico prestado a um determinado utente. Esse serviço poderá ir de um

simples curativo até uma intervenção cirúrgica. Para o caracterizarmos de uma forma completa precisamos de saber a quem foi prestado o serviço, quem foi o prestador do serviço, qual foi o tipo do serviço, quando foi prestado, em que instituição acorreu e qual foi o custo total que esse serviço implicou.

3.1.5 data

Para melhor manipularmos questões relacionadas com o tempo em que o cuidado foi realizado decidimos criar um predicado data. Este predicado contem todos os elementos essenciais de uma data como um ano, um mês, um dia. Adicionalmente decidimos acrecentar a hora, para que pudéssemos distinguir cuidados nas diferentes partes de um dia.

Nas tabelas seguintes apresentamos a manipulação do conhecimento.

Legenda:

Interdito - @AAA; **Incerto** - #AAA; **Impreciso** - a,b - a ou b; [a,b] - de a até b;

Table 1: Utente

ID	Nome	Idade	Morada
1	Ana	20	morada(‘Rua do Louro’ , ‘Caldelas’ , ‘Guimaraes’)
2	Bruno	[25,35]	morada(‘Rua do Louro’ , ‘Caldelas’ , ‘Guimaraes’)
3	Carlos	#497	@336
4	Duarte	35	morada(‘Rua dos Loiros’ , ‘Caldelas’ , ‘Guimaraes’)
5	Elisabete	26	morada(‘Rua da Ajuda’ , ‘Vila Nova’ , ‘Guimaraes’)
6	Filipa	#001	morada(‘Rua do Emigrante’ , ‘Azurem’ , ‘Braga’)
7	Gisela	33	#002
8	Helder	[15,18]	morada(‘Rua do Azevinho’ , ‘Braga’ , ‘Braga’)
9	Irene	{ 30,50 }	morada(‘Rua do Tolos’ , { ‘Briteiros’,’S.Clemente’ } , ‘Guimaraes’)
10	Jacinto	48	morada(‘Rua da Rua’ , ‘Braga’ , ‘Braga’)
11	{ Marta,Mara }	34	morada(‘Rua do Pinheiro’ , ‘São Lourenço’ , {‘Braga’,’Guimaraes’})
12	Joaquim	[60,80]	morada(‘Rua do Limoeiro’ , ‘Amais’ , ‘Viana do Castelo’)
13	Marcelo	45	@115
14	Diana	[8,12]	morada(‘Rua dos Loiros’ , ‘Caldelas’ , ‘Guimaraes’)

Table 2: Prestador

ID	Nome	Especialidade	Instituição
1	Antonio	urologia	Hospital de Guimarães
2	Bernardo	ortopedia	Hospital Privado de Guimarães
3	Carla	#789	Hospital de Guimarães
4	Dalila	neurologia	@123
5	Ermelinda	enfermeira	Hospital de Braga
6	Fausto	neurologia	{ Hospital Privado de Braga, Hospital de Braga }
7	Gabriel	@456	{Hospital de Braga, Hospital de Guimarães}
8	Henriqueta	cardiologia	Hospital de Braga
9	Iglesias	{urologia,patologia}	Hospital Privado de Braga
10	Josefina	Patologia	#423
11	Núria	Dermatologia	Hospital do Porto
12	João	@171	Hospital do Porto
13	Júlia	{ neurocirurgia, neurologia }	{Hospital do Porto, Hospital Guimaraes}
14	Renato	#145	#167

Todo o povoamento da base de conhecimento apresenta-se em anexo.

Table 3: Cuidado

Data	ID Utente	ID Prestador	Descrição	Custo	Instituição
[data(1,1,2018),data(5,1,2018)]	1	5	'curativo'	10	Hospital de Braga
data(2,1,2018)	2	6	'investigação'	@444	{Hospital Privado de Braga, Hospital de Braga}
data(1,2,2018)	3	7	#908	50	{Hospital de Braga, Hospital de Guimaraes}
data(2,2,2018)	4	8	@007	15489	Hospital de Braga
data(3,3,2018)	5	9	{'rotina','exame'}	25	Hospital Privado de Braga
data(3,4,2018)	6	10	'medição'	70	#423
{data(4,4,2018),data(5,4,2018)}	7	1	'exame'	69	Hospital de Guimarães
data(1,1,2018)	8	2	'cirurgia'	[100,500]	Hospital Privado de Guimarães
data(28,5,2018)	9	3	#111	#222	Hospital de Guimarães
@400	10	4	'rotina'	333	@123
data(20,5,2018)	11	13	'cirurgia'	100	#113
data(1,6,2018)	12	14	'exame'	50	#168
data(1,7,2018)	13	12	'rotina'	10	Hospital do Porto
data(4,7,2018)	14	11	@112	#999	Hospital do Porto

3.2 Evolução da Base de Conhecimento

A evolução do conhecimento apresentou-se como o maior fator de discussão no seio do grupo na realização do trabalho. Para evoluir a base de conhecimento tínhamos várias possibilidades, desde uma mais simples até a mais complexa de todas. De seguida iremos abordar sobre cada uma das abordagens disponíveis.

3.2.1 Adição de conhecimento negativo quando existe conhecimento positivo contraditório

Um dos objetivos no controlo da evolução da base conhecimento é garantir que esta não fique com conhecimento que a torne inconsistente. Desta forma não deve ser possível, manter na mesma base de conhecimento, um conhecimento positivo e outro negativo que são opostos. Por exemplo: Não podemos permitir que existam, em simultâneo, o seguinte conhecimento: prestador(1,jose,'ortopedia',['Hospital de Braga']) e -prestador(1,jose,'ortopedia',['Hospital de Braga']). Para controlar esta situação discutimos duas abordagens possíveis.

1ª Abordagem

A primeira abordagem não permite que tal conhecimento contraditório seja adicionado. Ou seja, caso se queira adicionar conhecimento negativo quando já se possui conhecimento positivo que contradiz o que se quer adicionar, então tal conhecimento não deve ser possível acrescentar, continuando a base, com o mesmo conhecimento.

2ª Abordagem

A segunda abordagem passa por realizar uma evolução da base de conhecimento. Isto é, quando se pretende adicionar conhecimento negativo quando

a base já possui conhecimento positivo a contradizer o que se pretende adicionar, então o conhecimento que se esta a adicionar é posto em evidência com a alteração do conhecimento presente na base de conhecimento do tipo positivo para o tipo Desconhecido. Caso se voltasse a colocar a mesma informação negativa então a base de conhecimento já deixaria adicionar tal conhecimento removendo o que possui do tipo Desconhecido.

Veredicto:

As duas abordagens são corretas e cada uma tem os seus prós e contras. A segunda abordagem permitiria uma evolução da base de conhecimento, porém é muito mais complexa para implementar, utilizando uma ferramenta como o PROLOG, e ainda era necessário acrescentar um novo tipo de conhecimento Desconhecido (por factos que serão abordados de seguida). No mundo real alguém conhecimento que hoje é verdadeiro, amanhã pode não o ser. No entanto é necessário ter uma relação entre o que é verdade e o que não é, como por exemplo pesos para o que conhecemos e o que é o novo conhecimento, sem nunca descartar o que de novo há para aprender nem esquecer o que antes se conhecia. É necessário "misturar tudo" o que se sabe, e uma forma de o fazer é com análises estatísticas. Isso não é implementável em PROLOG ou é bastante complexo.

A nossa escolha para tratar desta situação foi a primeira abordagem. Para a implementar apenas tivemos de verificar se a base de conhecimento já possuía um conhecimento positivo que fizesse match com o conhecimento negativo que se pretendesse adicionar. Decidimos implementar por ser mais simples que a segunda, mas não só. Analisando o conhecimento que queremos representar, atos médicos, é um "conhecimento fixo", ou seja, que em princípio quando a possuímos conhecimento positivo ou negativo, este não se vai alterar nunca mais, já que é um historial médico, ou então a informação de um utente. Assim sendo, decidimos optar pela primeira abordagem que nos pareceu ser a mais indicada para a evolução sobre o conhecimento que pretendemos representar.

De notar que poderia ainda se ter discutido sobre uma terceira abordagem em que quando se inserir conhecimento negativo, se retirava o conhecimento positivo contraditório. Mas esta abordagem não teve qualquer apoio por qualquer elemento do grupo, pelo que não foi motivo de discussão.

A inserção de conhecimento positivo quando já existe conhecimento negativo segue um raciocínio análogo, pelo que não será exposto.

3.2.2 Adição de conhecimento quando existe um conhecimento do tipo Desconhecido

De salientar que caso o conhecimento Desconhecido seja do tipo Interdito não é permitido adicionar qualquer tipo de conhecimento. Todas as abordagens que iremos abordar em seguida são relativas à existência de conhecimento Desconhecido do tipo impreciso e incerto.

Aqui apenas iremos apresentar algumas abordagens possiveis sobre a adição de conhecimento positivo ou negativo. Sobre a adição de conhecimento do tipo Desconhecido abordaremos posteriormente.

1ª Abordagem

Uma possível abordagem seria "à la Funcionário Público" onde qualquer conhecimento que tentássemos inserir, tal não seria possível caso a base de conhecimento já possuísse conhecimento relativo ao facto que queremos inserir. Por exemplo, estamos a tentar inserir o seguinte conhecimento positivo: "cuidado(data(2018,04,18,10),1,1,'rotina',20,'Hospital')" mas já existe, na base de conhecimento, uma exceção para este mesmo cuidado onde não se sabe se o custo do mesmo foi de 20 ou 30. Com a abordagem referida a cima, como então já existe um conhecimento "análogo" não deve ser permitido acrescentar o conhecimento positivo. Desta forma seria bastante fácil tratar da evolução da base de conhecimento, já que apenas evoluía para conhecimento que ainda não possuía. Apesar da sua simplicidade não retrata de forma ideal o conhecimento que queremos representar, já que, recuando ao exemplo dado em cima, a qualquer momento podia-se ter certeza de qual o valor do custo do cuidado, pelo que poderá ser vantajoso inserir o novo conhecimento.

2ª Abordagem

Outra abordagem, que seria muito mais complexa, era analisar não só se podíamos inserir ou não tal conhecimento como também fazer uma análise do conhecimento na base, onde o conhecimento verdadeiro estaria sempre no intervalo do conhecimento do tipo Desconhecido, e tudo o que não fosse verdadeiro ou desconhecido, seria totalmente falso.

Quando temos conhecimento do tipo Incerto, caso acrescentássemos conhecimento negativo nenhuma alteração ia ser efetuada, porém caso acrescentássemos conhecimento positivo, então todo o conhecimento do tipo desconhecido Incerto e, eventual, conhecimento negativo seriam apagados da base de conhecimento.

No caso de possuirmos conhecimento do tipo Impreciso, na adição de conhecimento positivo, que estivesse dentro do intervalo delimitado (caso estivesse fora do intervalo que o Desconhecido abrange, não seria permitida a sua adição) a ação seria análoga ao do caso referido anteriormente. O caso especial seria se pretendêssemos adicionar conhecimento negativo. Mais uma vez, caso este conhecimento estivesse fora do intervalo abrangido pelo conhecimento Desconhecido, tal conhecimento não seria possível ser adicionado. Porém, se estivesse dentro do intervalo, então deixaria acrescentar. Mas, utilizando esta abordagem, caso houvesse apenas um e um único facto do tipo desconhecido (após a inserção de um conhecimento do tipo negativo) então esse facto desconhecido deveria ser tornado verdadeiro e todos os restantes conhecimentos negativos apagados da base de conhecimento já que seria conhecimento redundante. Pegando mais uma vez no exemplo referido a cima em que na base de conhecimento temos: "exceção(cuidado(data(2018,04,18,10),1,1,'rotina',20,'Hospital'))" e "exceção(cuidado(data(2018,04,18,10),1,1,'rotina',30,'Hospital'))" e queremos inserir "-cuidado(data(2018,04,18,10),1,1,'rotina',20,'Hospital')". Neste caso, ficaríamos apenas com uma exceção, pelo que o único conhecimento que deveria de estar presente na base de conhecimento era "cuidado(data(2018,04,18,10),1,1,'rotina',20,'Hospital')". Isto, deve-se ao facto de se considerar que o conhecimento verdadeiro está dentro dos limites do conhecimento do tipo Desconhecido e tudo o resto ser falso.

Esta abordagem seria bastante diferente da inicial, e muito difícil de conseguir implementar usando a ferramenta pedida PROLOG, tornando-se um desafio bastante interessante.

3ª Abordagem

Por fim, uma terceira abordagem, onde apesar de poder existir um conhecimento do tipo Desconhecido, não há certezas absolutas de que o conhecimento verdadeiro esteja dentro do intervalo do Desconhecido e tudo o que está fora é falso, mas não é uma negação forte, mas sim antes, uma negação por falha na prova de que o conhecimento é verdadeiro ou desconhecido.

Nesta abordagem, a inserção tanto de conhecimento verdadeiro como de conhecimento falso ia ser feito de forma semelhante. Ou seja, caso inseríssemos estes tipos de conhecimento, este iria coexistir com o conhecimento do tipo Desconhecido, quer estivesse dentro ou fora dos limites, já que o facto de ser Desconhecido não garante com certeza absoluta de que o conhecimento verdadeiro esteja definido no intervalo do Desconhecido. É uma abordagem

menos intuitiva que as outras, e até nos leva a pensar, se não temos a certeza de que o valor verdadeiro esteja dentro do Desconhecido então para que queremos deste tipo de conhecimento Desconhecido? Ora, com certeza uma boa questão, que pode ser respondida pelo seguinte exemplo: Dois colegas estão a discutir a classificação de uma equipa de futebol, neste caso o Vitória SC, na temporada transata, onde chegam à conclusão de que como o Vitória é uma equipa mediana em Portugal (no que toca ao valor financeiro e não ao da equipa) dizem que muito provavelmente ficou entre 5º e 8º, já que os 4 primeiros lugares estariam dedicados ao Benfica, Porto, Sporting e Braga. Porém, a verdadeira classificação não se encontra no intervalo que os amigos determinaram, já que o Vitória ficou em 4º lugar. Tínhamos um conhecimento do tipo Desconhecido onde parecia que quase de certeza a informação verdadeira estava dentro do intervalo, mas na verdade estava fora, porém, aquele conhecimento do tipo Desconhecido parecia fazer sentido.

Quando inserimos um conhecimento negativo, tanto este como o conhecimento Desconhecido, seriam mantidos na base de conhecimento, porém nunca ficaria inconsistente. Apesar de poder dar duas respostas para a mesma pergunta (caso se pergunte diretamente à base de conhecimento), para nós a base só fica inconsistente caso tenha conhecimento positivo e ao mesmo tempo negativo a contradizer o positivo. Mas porquê manter os dois? Porque caso eliminássemos o conhecimento do tipo Negativo, este ficaria a ser Desconhecido e não falso por negação na prova. O mesmo raciocínio é feito para a inserção de conhecimento verdadeiro.

Veredicto:

Pelo facto de a última abordagem ser menos complexa ao nível de implementação usando a ferramenta PROLOG e ser bastante completa na representação do conhecimento, que no fundo é o mais importante a considerar, decidimos usá-la como estratégia na evolução de conhecimento no caso de já existir conhecimento do tipo Desconhecido.

3.2.3 Adição de conhecimento desconhecido

No que toca à adição de conhecimento do tipo Desconhecido, optámos por não adicionar, devido ao seu grau de complexidade em PROLOG, mas no entanto, deixámos qual deveria de ser um tipo de abordagem a utilizar.

Caso o conhecimento Desconhecido fosse do tipo Interdito, seria necessário eliminar da base de conhecimento todo o conhecimento positivo, negativo e

Desconhecido, já que, para qualquer pergunta para o predicado a inserir, a resposta a dar teria de ser sempre "desconhecido". Caso pretendêssemos adicionar conhecimento Desconhecido, mas no entanto, já existisse um conhecimento positivo ou negativo, que fizesse match com o predicado a inserir, íamos permitir que tal acontecesse, já que, segundo abordagens anteriores, deixamos que exista em simultâneo conhecimento Desconhecido e negativo ou Desconhecido e positivo.

Porém quando existisse conhecimento Desconhecido do tipo Impreciso ou Incerto era necessário efetuar uma análise ao conhecimento que já existe o que se pretende inserir para que o intervalo de desconhecido fosse sempre coerente. Ou seja, um exemplo seria, efetuar uma disjunção, onde se ficava com o maior dos dois intervalos, ou então, uma conjunção disjunta (este último poderia seria o mais ideal a aplicar, mas também seria o mais difícil para efetuar).

Ora para que tal fosse possível, era necessário fazer um parse do que se estava a inserir (para poder identificar qual o tipo de Desconhecido), posteriormente analisar a base de conhecimento, e por ventura, era necessário tomar decisões de qual o intervalo a manter sobre o desconhecido e ainda criar um novo predicado (diferente do que queríamos inserir), caso tivéssemos de alterar a variação dos valores do desconhecido.

3.2.4 Aplicação Prática

Para auxiliar no processo de adição de conhecimento construímos um meta-predicado denominado de "evolução". Este tinha como principal objetivo fazer uma avaliação (testando invariantes) de que se o conhecimento poderia ser adicionado ou não. Caso o conhecimento fosse adicionado com sucesso o resultado seria *true*, caso não fosse adicionado o conhecimento deveria dar como resposta *false*.

```
% Extensao do predicado evolucao: Termo -> {V,F}
evolucao( Termo ) :- solucoes( Inv,+Termo::Inv,S ),
                     inserir( Termo ),
                     testar( S ).

% Extensao do predicado inserir: Predicado -> {V,F}
inserir( P ) :- assert( P ).
inserir( P ) :- retract( P ), !, fail.
```

```
% Extensao do predicado testar: ListaPredicado -> {V,F}
testar( [] ).
testar( [X|R] ) :- X,
                   testar( R ).
```

Para que não fosse possível adicionar, por exemplo, um cuidado "negativo" quando já existia um cuidado igual, criamos um invariante que procurava na base de conhecimento se já existia um cuidado com todos os parâmetros iguais ao conhecimento negativo que estamos a inserir.

```
% ------ Nao pode haver conhecimento negativo igual aquele cuidado
se queremos adicionar negativo ----------------

+( -cuidado( Id,Data,U,P,D,C,I ) ) :: (
                   solucoes( A,-cuidado( Id,Data,U,P,D,C,I ),L ),
                   comprimento( L,N ),
                   N == 0).
```

Já no caso de existir conhecimento positivo, é necessário verificar se não existe já um predicado positivo com id igual ao que queremos inserir (neste caso para o predicado "utente").

```
%---------- Nao podem haver ids repetidos ----------------
+cuidado( IdC,_,_,_,_,_,_,_ ) :: (
solucoes( IdP,( cuidado( IdC,A,B,C,D,E,F ),
                   nao( nuloD( A ) ),nao( nuloD( B ) ),
                   nao( nuloD( C ) ),nao( nuloD( D ) ),
                   nao( nuloD( E ) ),nao( nuloD( F ) ) ),L ),
                   comprimento( L,X ),
                   X =< 1).
```

Os predicados apresentados anteriormente, são muito idênticos aos construídos para os predicados utente/4 e prestador/4.

Para além destes invariantes construímos outros que controlam, não só a nível estrutural, como a nível referencial, como por exemplo, o facto de um prestador não puder ter mais do que 3 serviços à mesma hora, ou então, não prestar um serviço numa instituição da qual não faz parte, entre outros. De salientar, que estes invariantes, já forma utilizados no trabalho prático

anterior, mas era importante, voltar a referir e colocar neste trabalho, já que implicitamente tratam da evolução do conhecimento.

3.2.5 Apresentação de Resultados

```
| ?- evolucao( cuidado( 3,data( 2,1,2018 ),3,6,'investigacao',10,'Hospital Privado de Braga' )).
yes
| ?-
| ?-
| ?-
| ?- listing( cuidado ).
cuidado(2, data(2,1,2018), 2, 6, investigacao, xpto444, 'Hospital Privado de Braga').
cuidado(2, data(2,1,2018), 2, 6, investigacao, xpto444, 'Hospital de Braga').
cuidado(3, data(1,2,2018), 3, 7, xpto908, 50, 'Hospital de Braga').
cuidado(3, data(1,2,2018), 3, 7, xpto908, 50, 'Hospital de Guimaraes').
cuidado(4, data(2,2,2018), 4, 8, xpto007, 15489, 'Hospital de Braga').
cuidado(6, data(3,4,2018), 6, 10, medicao, 70, xpto424).
cuidado(9, data(28,5,2018), 0, 3, xpto11, xpto222, 'Hospital de Guimares').
cuidado(10, xpto400, 10, 4, rotina, 333, xpto123).
cuidado(11, data(20,5,2018), 11, 12, cirurgia, 100, xpto113).
cuidado(11, data(20,5,2018), 11, 12, cirurgia, 110, xpto113).
cuidado(12, data(1,6,2018), 12, 14, exame, 50, xpto168).
cuidado(13, data(1,7,2018), 13, 12, rotina, 10, 'Hospital do Porto').
cuidado(14, data(4,7,2018), 14, 11, xpto112, xpto999, 'Hospital do Porto').
cuidado(3, data(2,1,2018), 3, 6, investigacao, 10, 'Hospital Privado de Braga').
```

Figure 1: Evolucao de conhecimento positivo

```
| ?- evolucao( -cuidado( 3,data( 2,1,2018 ),3,6,'investigacao',10,'Hospital Privado de Braga' )).
no
| ?-
| ?-
| ?- evolucao( cuidado( 3,data( 2,1,2018 ),4,6,'investigacao',10,'Hospital Privado de Braga' )).
no
| ?- evolucao( -cuidado( 3,data( 2,1,2018 ),6,6,'investigacao',10,'Hospital Privado de Braga' )).
yes
| ?- listing( - ).
-utente(1,maria,20,morada('Rua do Louro','Caldelas','Guimaraes')).
-utente(A,B,C,D) :-
        nao(utente(A,B,C,D)),
        nao(excecao(utente(A,B,C,D))).
-prestador(1,peixe,'medicina interna','Hospital de Guimaraes').
-prestador(A,B,C,D) :-
        nao(prestador(A,B,C,D)),
        nao(excecao(prestador(A,B,C,D))).
-cuidado(1,data(2,1,2018),1,5,curativo,10,'Hospital de Braga').
-cuidado(A,B,C,D,E,F,G) :-
        nao(cuidado(A,B,C,D,E,F,G)),
        nao(excecao(cuidado(A,B,C,D,E,F,G))).
-cuidado(3,data(2,1,2018),6,6,investigacao,10,'Hospital Privado de Braga').

yes
```

Figure 2: Evolução de conhecimento negativo e repetido

3.3 "Inevolução" da Base de Conhecimento

A "Inevolução" consiste na remoção do conhecimento da base. Para realizar esta ação, tivemos de construir um predicado "inevolução", muito semelhante ao predicado evolução, mas com o intuito de avaliar se era permitido remover tal conhecimento da base.

No caso de querer remover conhecimento negativo, permitimos sempre, já que não afeta em nada a base de conhecimento. Já a remoção de conhecimento positivo, já não é assim tão trivial. Decidimos que um cuidado, pode ser removido, porém caso se pretenda remover um prestador ou então um utente, tal só é permitido se o mesmo não possuir registos de cuidados. Para

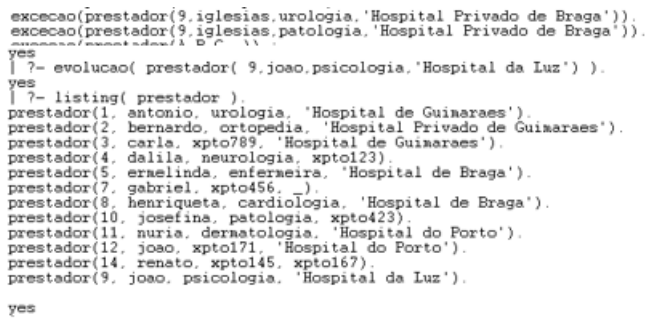

Figure 3: Evolução de conhecimento postivo, com desconhecido (não interdito) existente

este caso apenas tivemos de construir um invariante que verificava se o utente ou prestador em causa tinha algum registo de um cuidado, quer esse registo seja positivo, negativo ou do tipo Desconhecido.

No caso de conhecimento de tipo desconhecido, não deixamos retirar, por ser, mais uma vez, uma ação algo complexa para ser tratada com uma ferramenta como o PROLOG. Por exemplo, caso existisse um conhecimento Desconhecido do tipo impreciso no parâmetro do "custo", entre 10 e 50, caso se pretendesse remover que era desconhecido para o valor de 15, teríamos de dividir o predicado presente em dois, para que pudesse criar duas áreas de imprecisão, nos intervalos [10, 15[e]15,50].

3.4 Demonstrador

O *ProLog*, linguagem utilizada, não nos responde às questões da forma pretendida pois apenas responde com verdadeiro e falso e nós queremos que responda com verdadeiro, falso e desconhecido, visto termos a partir de agora diversos tipos de conhecimento, como referido anterirormente. Por esta razão decidimos criar um **demonstrador** que se apoia na linguagem utilizada para dar as respostas pretendidas.

Para realizar o **demonstrador** podíamos ter efetuado um que apenas nos dá uma resposta a uma questão, no entanto decidimos que faria sentido criar um mecanismo de demonstração que desse resposta a um conjunto de questões, separadas por operadores lógicos. Decidimos que apenas teríamos 2 tipos de operadores: relativos à conjunção e à disjunção de termos, visto

que a maioria das questões são do tipo e/ou e que praticamente todas as fórmulas lógicas podem ser reduzidas a conjunções ou disjunções.

Para podermos utilizar o mecanismo referido e como temos 3 tipos de conhecimento diferentes (verdadeiro, falso e desconhecido), não pudemos utilizar as tabelas de verdade da lógica clássica relativas aos operadores referidos. Tivemos então de criar as nossas tabelas, que são apresentadas de seguida.

3.4.1 Tabela da conjunção

Para os casos que apenas existem na lógia clássica não efetuamos qualquer alteração. No entanto, tivemos de definir qual seria o resultado quando a conjunção envolvesse conhecimento imperfeito, ou seja, desconhecido. Aqui definimos o mesmo raciocínio da lógica clássica, no qual pusemos o conhecimento desconhecido entre o verdadeiro e o falso, logo este seria absorvente com conhecimento verdadeiro e o conhecimento falso seria absorvente com conhecimento desconhecido.

e	V	F	D
V	V	F	D
F	F	F	F
D	D	F	D

Table 4: Tabela da conjuncao

```
%---------- e: Termo, Termo, Resposta -> {V,F}
e( falso ,_, falso ).
e( _, falso , falso ).
e( desconhecido , desconhecido , desconhecido ).
e( desconhecido , verdadeiro , desconhecido ).
e( verdadeiro , desconhecido , desconhecido ).
e( verdadeiro , verdadeiro , verdadeiro ).
```

3.4.2 Tabela da disjunção

Para os casos que apenas existem na lógia clássica, mais uma vez, não efetuamos qualquer alteração. No entanto, tivemos de definir qual seria o resultado quando a disjunção envolvesse conhecimento imperfeito, ou seja, desconhecido. Aqui definimos o mesmo raciocínio aplicado para a conjunção relativamente à lógica clássica, com o Desconhecido a ser absorvente no caso

de existir Falso ou Desconhecido, e o Verdadeiro a ser absorvente no caso de existir Verdadeiro ou Desconhecido.

ou	V	F	D
V	V	V	V
F	V	F	D
D	V	D	D

Table 5: Tabela da disjuncao

```
%---------- ou: Termo,Termo,Resposta -> {V,F}
ou( verdadeiro,_,verdadeiro ).
ou( _,verdadeiro,verdadeiro ).
ou( desconhecido,desconhecido,desconhecido ).
ou( desconhecido,falso,desconhecido ).
ou( falso,desconhecido,desconhecido ).
ou( falso,falso,falso ).
```

3.4.3 Implementação do demonstrador

Para a implementação do **demonstrador** tivemos de decidir de qua forma é que íamos representar o conhecimento colocado em argumento no predicado do **demonstrador** . Após alguma discussão chegamos a duas abordagens possíveis, podia ser representado por:

1. e(A,B); ou(A,B)
2. A^B; A+B

Decidimos utilizar a segunda abordagem, pelo facto da dificuldade de representar a primeira ser superior em *ProLog*. Em relação à segunda decidimos utilizar o símbolo ^ como o símbolo da conjunção, visto ser uma abordagem universal e caso usássemos o símbolo e poderia ser complicado nos casos em que os predicados contivessem o carater "e". Para representar o símbolo correspondente à conjunção poderíamos ter optado por ou, no entanto, pelo mesmo motivo referido acima, não optamos por utilizar o ou. Podíamos também ter utilizado o símbolo v, no entanto não usamos pela mesma razão e decidimos utilizar o símbolo + visto também ser um símbolo universal para a disjunção.

Com a decisão da representação do conhecimento efetuada passamos para a decisão de como seria realizado o *parsing* do mesmo, pois este seria necessário por forma a utilizar os predicados criados (predicados referidos nas tabelas). Para isto, apenas utilizamos a abordagem referida anteriormente, ou seja o predicado *demonstrador* divide-se em:

- demonstrador(A^B,R).
- demonstrador(A+B,R).

e cada um auxilia-se no predicado lógico correspondente. Depois teríamos também de decidir o que seria realizado aos termos A e B, e como estes poderiam também ser conjunções e disjunções aplicamos recursivamente o mesmo predicado até que o termo seja um simples termo. Chegado a este caso, chamamos um predicado auxiliar **demo** criado para responder a termos simples.

```
demonstrador( A+B,R ) :-
        demonstrador( A,T ),
        demonstrador( B,Q ),
        ou( T,Q,R ).

demonstrador( A^B,R ) :-
        demonstrador( A,T ),
        demonstrador( B,Q ),
        e( T,Q,R ).

demonstrador( A,R ) :-
        demo( A,R ).
```

3.4.4 Implementação do demo

O predicado **demo** manipula como conhecimento um termo simples que queremos demonstrar e o resultado da demonstração, que pode ser:

- verdadeiro
- falso
- desconhecido

A demonstração do termo é verdadeira caso exista uma prova para o mesmo. É falsa caso exista uma prova para a negação (forte) do termo. Por exclusão de partes é desconhecida caso não exista uma prova para o termo nem para a negação (forte) do mesmo.

```
demo( Questao, verdadeiro ) :-
    Questao.

demo( Questao, falso ) :-
    -Questao.

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

3.5 Apresentação de Resultados

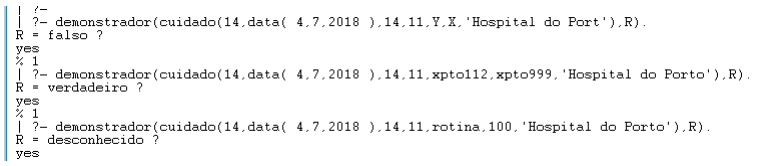

Figure 4: Teste de perguntas sobre o predicado do cuidado com id 14

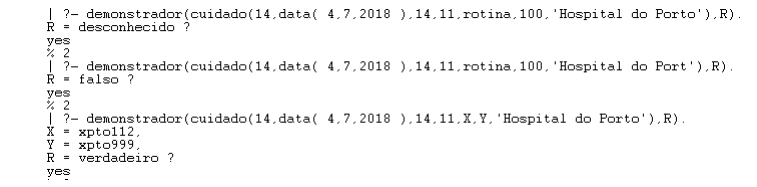

Figure 5: Teste de multi-perguntas sobre o predicado do cuidado com id 14

4 Conclusões e Sugestões

Conseguimos, sem grandes problemas, implementar o mecanismo de evolução da base de conhecimento do modo que planeamos inicialmente, não abdicando da complexidade pretendida. O mecanismo que implementamos permite que seja possível adicionar conhecimento positivo e negativo desde que este conhecimento não seja contraditório ou repetido ao conhecimento que já está a ser representado e respeite as restrições impostas pelo conhecimento de tipo Desconhecido que já lá estava presente.

Implementamos também com sucesso um predicado *demonstrador* capaz de dar resposta a não só um predicado mas também de múltiplos predicados em simultâneo.Isto permitiu que as queries feitas à nossa base de conhecimento fossem mais complexas e práticas.

Pensamos na possibilidade de permitir também a inserção de conhecimento do tipo Desconhecido na nossa base de conhecimento, mas, como já foi abordado anteriormente, optamos por não o fazer pois esta abordagem seria de elevado grau de complexidade para ser implementada em *PROLOG*, pelo que apenas registamos teoricamente a abordagem a tomar.

Como trabalho futuro poderemos aproximar este sistema da realidade permitindo que conhecimento positivo ou negativo na nossa base de conhecimento possa ser alterado após a sua veracidade ser posta em causa (após várias tentativas de inserir conhecimento contraditório por exemplo). Poderemos também permitir a inserção de conhecimento do tipo desconhecido, novamente no sentido de aproximar o sistema à realidade.

O projeto desenvolvido gerou um resultado bastante satisfatório uma vez que alcançamos todos os objetivos a que nos propusemos inicialmente.

5 Anexos

```
%---------------------------------- - - - - - - - - - -  -  -  -
-   -

% SIST. REPR. CONHECIMENTO E RACIOCINIO - MiEI/3


%---------------------------------- - - - - - - - - - -  -  -  -
-   -

% TP1 - Programacao em logica e invariantes

%---------------------------------- - - - - - - - - - -  -  -  -
-   -

% SICStus PROLOG: Declaracoes iniciais


:- set_prolog_flag( discontiguous_warnings,off ).

:- set_prolog_flag( single_var_warnings,off ).

:- set_prolog_flag( unknown,fail ).



%---------------------------------- - - - - - - - - - -  -  -  -
-   -

% SICStus PROLOG: definicoes iniciais


:- op( 900,xfy,'::' ).

:- dynamic utente/4.

:- dynamic prestador/4.

:- dynamic cuidado/7.

:- dynamic '-'/1.


%---------------------------------- - - - - - - - - - -  -  -  -
-   -

% Extensao do predicado utente: IdUt, Nome, Idade, Morada ->
{V,F}


-utente(1,maria,20,morada('Rua do
Louro','Caldelas','Guimaraes')).


utente( 1,ana,20,morada( 'Rua do Louro','Caldelas','Guimaraes' )
).
```

```
excecao( utente( 2,bruno,A,morada( 'Rua do Louro' , 'Caldelas' ,
'Guimaraes' ) ) ) :- A >= 25,

                                                    A =< 35.


utente( 3,carlos,xpto497,xpto336 ).

nuloD( xpto497 ).

excecao( utente( A,C,_,_ ) ) :- utente( A,C,xpto497,xpto336 ).

nuloI( xpto336 ).


+utente( O,B,C,D ) :: ( solucoes( N,( utente( 3,N,X,T ),nao(
nuloI( T ) ) ),L ),
                          comprimento( L,Aux ),
                          Aux == 0
                          ).


+(-utente( O,B,C,D )) :: ( solucoes( N,( -utente( 3,N,X,T ),nao(
nuloI( T ) ) ),L ),
                          comprimento( L,Aux ),
                          Aux == 0
                          ).



utente( 4,duarte,35,morada( 'Rua dos Loiros' , 'Caldelas' ,
'Guimaraes' ) ).



utente( 5,elisabete,26,morada( 'Rua da Ajuda' , 'Vila Nova' ,
'Guimaraes' ) ).


utente( 6,filipa,xpto001,morada('Rua do
Emigrante','Azurem','Braga') ).

nuloD( xpto001 ).
```

```
excecao( utente(A,B,_,D) ) :- utente( A,B,xpto001,D ).



utente( 7,gisela,33,xpto002 ).

nuloD( xpto002 ).

excecao( utente(A,B,C,_) ) :- utente(A,B,C,xpto002).



excecao( utente( 8,helder,Idade,morada('Rua do
Azevinho','Braga','Braga') ) ) :-

      Idade >= 15,

      Idade =< 18.


excecao(utente(9,irene,30,morada('Rua dos
tolos','Briteiros','Guimaraes'))).

excecao(utente(9,irene,30,morada('Rua dos
tolos','S.Clemente','Guimaraes'))).

excecao(utente(9,irene,50,morada('Rua dos
tolos','Briteiros','Guimaraes'))).

excecao(utente(9,irene,50,morada('Rua dos
tolos','S.Clemente','Guimaraes'))).


utente(10,jacinto,48,morada('Rua da Rua','Braga','Braga')).


excecao(utente(11,marta,34,morada('Rua do Pinheiro','Sao
Lourenco','Braga'))).

excecao(utente(11,marta,34,morada('Rua do Pinheiro','Sao
Lourenco','Guimaraes'))).

excecao(utente(11,mara,34,morada('Rua do Pinheiro','Sao
Lourenco','Braga'))).

excecao(utente(11,mara,34,morada('Rua do Pinheiro','Sao
Lourenco','Guimaraes'))).


excecao( utente( 12,joaquim,I,morada( 'Rua do Limoeiro' ,
'Amais' , 'Viana do Castelo' ) ) ) :-

      I >= 60,
```

```
      I =< 80.


utente( 13,marcelo,45,xpto115 ).

excecao( utente( ID,N,I,L ) ) :- utente( ID,N,I,xpto115 ).

nuloI(xpto115).


+utente( ID,N,I,L ) :: ( solucoes( ID,( utente( 13,N,I,M ),nao(
nuloI(M) ) ),R ),
                                  comprimento( R,Aux ),
                                  Aux==0
                                  ).


+(-utente( ID,N,I,L )) :: ( solucoes( ID,( -utente( 13,N,I,M
),nao( nuloI(M) ) ),R ),
                                  comprimento( R,Aux ),
                                  Aux==0
                                  ).


excecao( utente( 14,diana,I,morada( 'Rua dos Loiros' ,
'Caldelas' , 'Guimaraes' ) ) ) :-
      I >= 8,
      I =< 12.




-utente(ID,N,I,M) :-
      nao( utente(ID,N,I,M) ),
      nao( excecao( utente( ID,N,I,M ) ) ).



%--------------------------------- - - - - - - - - - -  -  -  -
-   -

% Extensao do predicado prestador: IdPrest, Nome, Especialidade,
Instituicao -> {V,F}
```

```
-prestador( 1,peixe,'medicina interna','Hospital de Guimaraes'
).



prestador( 1,antonio,urologia,'Hospital de Guimaraes' ).



prestador( 2,bernardo,ortopedia,'Hospital Privado de Guimaraes'
).



prestador( 3,carla,xpto789,'Hospital de Guimaraes' ).

nuloD( xpto789 ).

excecao( prestador( A,B,_,D ) ) :- prestador( A,B,xpto789,D ).



prestador( 4,dalila,neurologia,xpto123 ).

nuloI( xpto123 ).

excecao( prestador( A,B,C,_ ) ) :- prestador( A,B,C,xpto123 ).



+prestador( O,B,C,D ) :: ( solucoes( N,( prestador( 4,N,X,T
),nao( nuloI( T ) ) ),L ),
                           comprimento( L,Aux ),
                           Aux == 0
                           ).


+(-prestador( O,B,C,D )) :: ( solucoes( N,( -prestador( 4,N,X,T
),nao( nuloI( T ) ) ),L ),
                           comprimento( L,Aux ),
                           Aux == 0
                           ).
```

```
prestador( 5,ermelinda,enfermeira,'Hospital de Braga' ).



excecao( prestador( 6,fausto,neurologia,'Hospital Privado de
Braga' ) ).
excecao( prestador( 6,fausto,neurologia,'Hospital de Braga' ) ).



excecao( prestador( 7,gabriel,xpto456,'Hospital de Braga' ) ).
excecao( prestador( 7,gabriel,xpto456,'Hospital de Guimaraes' )
).
nuloI(xpto456).
prestador(7,gabriel,xpto456,'Hospital de Braga').
prestador(7,gabriel,xpto456,'Hospital de Guimaraes').
excecao( prestador( A,B,_,D ) ) :-   prestador( A,B,xpto456,D).


+prestador( O,B,C,D ) :: ( solucoes( N,( prestador( 7,N,X,T
),nao( nuloI( X ) ) ),L ),
                           comprimento( L,Aux ),
                           Aux == 0
                           ).


+(-prestador( O,B,C,D )) :: ( solucoes( N,( -prestador( 7,N,X,T
),nao( nuloI( X ) ) ),L ),
                           comprimento( L,Aux ),
                           Aux == 0
                           ).



prestador( 8,henriqueta,cardiologia,'Hospital de Braga' ).
```

```
excecao(prestador(9,iglesias,urologia,'Hospital Privado de
Braga')).

excecao(prestador(9,iglesias,patologia,'Hospital Privado de
Braga')).


prestador(10,josefina,patologia,xpto423).

excecao(prestador(A,B,C,_)) :- prestador(A,B,C,xpto423).


prestador(11,nuria,dermatologia,'Hospital do Porto').



prestador( 12,joao,xpto171,'Hospital do Porto' ).

excecao( prestador( ID,N,_,L ) ) :- prestador( ID,N,xpto171,L ).

nuloI(xpto171).


+prestador( ID,N,E,L ) :: ( solucoes( ID,( prestador( 12,N,I,L
),nao( nuloI(I) ) ),R ),
                                    comprimento( R,Aux ),
                                    Aux==0
                                ).



excecao( prestador( 13,julia,neurocirurgia,'Hospital do Porto' )
).

excecao( prestador( 13,julia,neurocirurgia,'Hospital de
Guimaraes' ) ).

excecao( prestador( 13,julia,neurologia,'Hospital do Porto' ) ).

excecao( prestador( 13,julia,neurologia,'Hospital de Guimaraes'
) ).



prestador( 14,renato,xpto145,xpto167 ).

excecao( prestador( ID,N,_,_ ) ) :- prestador(
ID,N,xpto145,xpto167 ).
```

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


%-------------------------------- - - - - - - - - - -  -  -  -
-   -

% Extensao do predicado cuidado: Data, IdUt, IdPrest, Descricao,
Custo, Instituicao -> {V,F}


-cuidado(1,data( 2,1,2018,20 ),1,5,'curativo',10,'Hospital de
Braga' ).


excecao( cuidado( 1,D,1,5,'curativo',10,'Hospital de Braga' ) )
:-

      comparaData( >=,D,data( 1,1,2018,20 ) ),

      comparaData( <=,D,data( 5,1,2018,20 ) ).



cuidado( 2,data( 2,1,2018,20
),2,6,'investigacao',xpto444,'Hospital Privado de Braga' ).

cuidado( 2,data( 2,1,2018,20
),2,6,'investigacao',xpto444,'Hospital de Braga' ).

nuloI( xpto444 ).

excecao( cuidado( 2,data( 2,1,2018,20
),2,6,'investigacao',xpto444,'Hospital Privado de Braga' ) ).

excecao( cuidado( 2,data( 2,1,2018,20
),2,6,'investigacao',xpto444,'Hospital de Braga') ).

excecao( cuidado( A,B,C,D,E,_,G ) ) :- cuidado(
A,B,C,D,E,xpto444,G ).


%-------- Invariante de nulo interdito ---------------

+cuidado( O,A,B,C,D,E,F ) :: ( solucoes( N,( cuidado(
2,N,_,T,_,X,_ ),nao( nuloI( X ) ) ),L ),
                              comprimento( L,Aux ),
```

```
                            Aux == 0

                            ).


%-------- Invariante de nulo interdito ---------------

+(-cuidado( O,A,B,C,D,E,F )) :: ( solucoes( N,( -cuidado(
2,N,_,T,_,X,_ ),nao( nuloI( X ) ) ),L ),

                            comprimento( L,Aux ),

                            Aux == 0

                            ).



cuidado( 3,data( 1,2,2018,20 ),3,7,xpto908,50,'Hospital de
Braga' ).

cuidado( 3,data( 1,2,2018,20 ),3,7,xpto908,50,'Hospital de
Guimaraes' ).

nuloD( xpto908 ).

excecao( cuidado( A,L,C,D,E,F,G ) ) :- cuidado(
A,L,C,D,xpto908,F,G ).

excecao( cuidado( 3,data( 1,2,2018,20 ),3,7,xpto908,50,'Hospital
de Braga' ) ).

excecao( cuidado( 3,data( 1,2,2018,20 ),3,7,xpto908,50,'Hospital
de Guimaraes') ).



cuidado( 4,data( 2,2,2018,20 ),4,8,xpto007,15489,'Hospital de
Braga' ).

nuloI( xpto007 ).

excecao( cuidado( A,B,C,D,_,F,G ) ) :- cuidado(
A,B,C,D,xpto007,F,G ).


%-------- Invariante de nulo interdito ---------------

+cuidado( O,A,B,C,D,E,F ) :: ( solucoes( N,( cuidado(
4,N,_,T,X,_,_ ),nao( nuloI( X ) ) ),L ),

                            comprimento( L,Aux ),

                            Aux == 0

                            ).
```

```
+(-cuidado( O,A,B,C,D,E,F )) :: ( solucoes( N,( -cuidado(
4,N,_,T,X,_,_ ),nao( nuloI( X ) ) ),L ),
                                comprimento( L,Aux ),
                                Aux == 0
                                ).



excecao( cuidado( 5,data( 3,3,2018,20
),5,9,'rotina',25,'Hospital Privado de Braga' ) ).

excecao( cuidado( 5,data( 3,3,2018,20 ),5,9,'exame',25,'Hospital
Privado de Braga' ) ).


cuidado( 6,data(3,4,2018,20),6,10,'medicao',70,xpto424 ).

nuloD( xpto424 ).

excecao( cuidado( A,B,C,D,E,F,_ ) ) :- cuidado(
A,B,C,D,E,F,xpto424 ).



excecao( cuidado( 7,data(4,4,2018,20),7,1,'exame',69,'Hospital
de Guimaraes' ) ).

excecao( cuidado( 7,data(5,4,2018,20),7,1,'exame',69,'Hospital
de Guimaraes' ) ).



excecao( cuidado(
8,data(1,1,2018,20),8,2,'cirurgia',Custo,'Hospital Privado de
Guimaraes' ) ) :-
     Custo >= 100,
     Custo =< 500.



nuloD(xpto111).

nuloD(xpto222).

cuidado(9,data(28,5,2018,20),0,3,xpto111,xpto222,'Hospital de
Guimares').
```

```
excecao(cuidado(A,B,C,D,_,_,G)) :-
cuidado(A,B,C,D,xpto111,xpto222,G).



nuloI(xpto400).

nuloI(xpto123).

cuidado(10,xpto400,10,4,rotina,333,xpto123).

excecao(cuidado(A,_,C,D,E,F,_)) :-
cuidado(A,xpto400,C,D,E,F,xpto123).


+cuidado( O,A,B,C,D,E,F ) :: ( solucoes( N,( cuidado(
10,X,_,_,_,_,Y ),nao( nuloI( X ) ),nao(nuloI(Y) ) ),L ),
                            comprimento(L,Aux),
                            Aux == 0
                            ).


+(-cuidado( O,A,B,C,D,E,F )) :: ( solucoes( N,( -cuidado(
10,X,_,_,_,_,Y ),nao( nuloI( X ) ),nao(nuloI(Y) ) ),L ),
                            comprimento(L,Aux),
                            Aux == 0
                            ).


nuloD(xpto113).

cuidado(11,data(20,5,2018,20),11,12,cirurgia,100,xpto113).

exececao(cuidado(A,B,C,D,E,F,_)) :-
cuidado(A,B,C,D,E,F,xpto113).



cuidado( 12,data( 1,6,2018,20 ),12,14,'exame',50,xpto168 ).

excecao( cuidado( ID,D,U,P,Desc,C,_ ) ) :- cuidado(
ID,D,U,P,Desc,C,xpto168 ).

nuloD(xpto168).
```

```
cuidado( 13,data( 1,7,2018,20 ),13,12,'rotina',10,'Hospital do
Porto' ).


cuidado( 14,data( 4,7,2018,20 ),14,11,xpto112,xpto999,'Hospital
do Porto' ).

excecao( cuidado( ID,D,U,P,_,_,I ) ) :- cuidado(
ID,D,U,P,xpto112,xpto999,I ).

nuloI(xpto112).

nuloD(xpto999).


+cuidado( ID,D,U,P,Desc,C,I ) :: ( solucoes( ID,( cuidado(
14,D,U,P,Des,C,I ),nao( nuloI(Des) ) ),R ),
                                                comprimento( R,N
),
                                                N==0
                                            ).



-cuidado(ID,D,U,P,Desc,C,I) :-
      nao( cuidado(ID,D,U,P,Desc,C,I) ),
      nao( excecao( cuidado(ID,D,U,P,Desc,C,I) ) ).



%----------------------------------Data do cuidado é válida-----
----------------------------------


+cuidado( Data,U,P,Descricao,C,I ) :: (dataValida(Data)).



%----------------- ----- --- -- -
% Extensao do predicado dataValida: Data -> {V,F}
dataValida( data(A,M,D,H) ) :-  natural(A),
                                        natural(M),
                                        natural(D),
                                        natural(H),
```

```
                                        H < 25,
                                        M < 13,
                                        diasValidos( A,M,D ).

diasValidos( A,M,D ) :- R is M mod 2,
                                        R \= 0,
                                        D < 32,
                                        D > 0.
diasValidos( A,M,D ) :- R is M mod 2,
                                        R == 0,
                                        M \= 2,
                                        D < 30,
                                        D > 0.
diasValidos( A,2,D ) :- R is A mod 4,
                                        R \= 0,
                                        D < 29,
                                        D > 0.
diasValidos( A,2,D ) :- R is A mod 4,
                                        R == 0,
                                        D < 30,
                                        D > 0.

%----------------------------------Nao pode haver mais do que 3
cuidados à mesma hora tanto para o utente como o profissional---
--------------

+cuidado(Id,D,U,P,Des,C,I) ::
(solucoes((D,P),cuidado(Id,D,Ut,P,Descr,Cus,Ins),L),
                                        comprimento(L,X),
                                        X=<3
                                        ).

+cuidado(Id,D,U,P,Des,C,I) ::
(solucoes((D,U),cuidado(Id,D,U,Pr,Descr,Cus,Ins),L),
```

```
                                        comprimento(L,X),
                                        X=<3
                                        ).


%-------- Nao podem haver ids repetidos ------------


+utente( IdU,_,_,_ ) :: (solucoes( IdU,( utente( IdU,A,B,C
),nao( nuloD( A ) ),nao( nuloD( B ) ),nao( nuloD( C ) ) ),L ),
                                        comprimento( L,X ),
                                        X =< 1).


+prestador( IdP,_,_,_ ) :: (solucoes( IdP,( prestador( IdP,A,B,C
),nao( nuloD( A ) ),nao( nuloD( B ) ),nao( nuloD( C ) ) ),L ),
                                        comprimento( L,X ),
                                        X =< 1).



+cuidado( IdC,_,_,_,_,_,_ ) :: (solucoes( IdP,( cuidado(
IdC,A,B,C,D,E,F ),nao( nuloD( A ) ),nao( nuloD( B ) ),nao(
nuloD( C ) ),nao( nuloD( D ) ),nao( nuloD( E ) ),nao( nuloD( F )
) ),L ),
                                        comprimento( L,X ),
                                        X =< 1).



%--------- Nao pode ter uma idade inválida ----------
+utente( IdU,_,I,_ ) :: ( validaIdade( I ) ).


%--------------- --- -- -
% Extensao do predicado validaIdade: Idade -> {V,F}
validaIdade( A ) :- R is A+1,
                            natural(R).
```

```
%--------------- Nao se podem adicionar cuidados para os quais
nao existam utentes/prestadores -------------------


+cuidado( _,_,IdU,_,_,_,_ ) :: (solucoes( IdU,demonstrador(
utente( IdU,A,_,_ )+excecao( utente( IdU,_,_,_ ) ),verdadeiro
),L ),
                                comprimento( L,X ),
                                X >= 1).


+cuidado( _,_,_,IdP,_,_,_ ) :: (solucoes( IdP,demonstrador(
prestador( IdP,A,_,_ )+excecao( prestador( IdP,_,_,_ )
),verdadeiro ),L ),
                                comprimento( L,X ),
                                X >= 1).


%--------------- Nao se pode adicionar um cuidado cujo custo
seja negativo ------------------------


+cuidado( _,_,_,_,_,C,_ ) :: (C >= 0).


%----------------- Nao se pode adicionar um cuidado com os
campos todos iguais --------------------

+cuidado( Id,D,IdU,IdP,Desc,C,I ) :: (solucoes( LI,cuidado(
Id,D,IdU,IdP,Desc,C,I ),L ),
                                comprimento( L,X ),
                                X == 1).


%--------------- Nao se pode remover um utente/prestador para o
qual existam cuidados relativos ---------------


-utente( IdU,_,_,_ ) :: (solucoes( D,cuidado(_,D,IdU,_,_,_,_ ),L
),
                                comprimento( L,X ),
                                X == 0).
```

```
-prestador( IdP,_,_,_ ) :: (solucoes( D,cuidado(_,D,_,IdP,_,_,_
),L ),
                                comprimento( L,X ),
                                X == 0).


%-------------------- Invariantes de conhecimento imperfeito ---
-------------------------------------


% ----- Nao pode haver conhecimento negativo igual aquele
cuidado se queremos adicionar positivo -------------

+utente( Id,No,I,M ) :: (solucoes( A,-utente( Id,No,I,M ),L ),
                                                comprimento( L,N
),
                                                N == 0).


% ----- Nao pode haver conhecimento positivo igual aquele utente
se queremos adicionar negativo -------------

+( -utente( Id,No,I,M ) ) :: (solucoes( A,utente( Id,No,I,M ),L
),
                                                comprimento( L,N
),
                                                N == 0).


% ----- Nao pode haver conhecimento negativo igual aquele utente
se queremos adicionar negativo -------------

+( -utente( Id,No,I,M ) ) :: (solucoes( A,-utente( Id,No,I,M ),L
),
                                                comprimento( L,N
),
                                                N =< 2). % -----
aqui é menor ou igual a 2 pois ele encontra sempre o do fecho
transitivo
```

```
% ----- Nao pode haver conhecimento negativo igual aquele
cuidado se queremos adicionar positivo -------------

+prestador( Id,No,I,M ) :: (solucoes( A,-prestador( Id,No,I,M
),L ),
                                                    comprimento( L,N
),
                                                    N == 0).


% ----- Nao pode haver conhecimento positivo igual aquele
prestador se queremos adicionar negativo -------------

+( -prestador( Id,No,I,M ) ) :: (solucoes( A,prestador(
Id,No,I,M ),L ),
                                                    comprimento( L,N
),
                                                    N == 0).


% ----- Nao pode haver conhecimento negativo igual aquele
prestador se queremos adicionar negativo -------------

+( -prestador( Id,No,I,M ) ) :: (solucoes( A,-prestador(
Id,No,I,M ),L ),
                                                    comprimento( L,N
),
                                                    N =< 2).



% ----- Nao pode haver conhecimento negativo igual aquele
cuidado se queremos adicionar positivo -------------

+cuidado( Id,Data,U,P,D,C,I ) :: (solucoes( A,-cuidado(
Id,Data,U,P,D,C,I ),L ),
                                                    comprimento( L,N
),
                                                    N == 0).


% ----- Nao pode haver conhecimento positivo igual aquele
cuidado se queremos adicionar negativo -------------

+( -cuidado( Id,Data,U,P,D,C,I ) ) :: (solucoes( A,cuidado(
Id,Data,U,P,D,C,I ),L ),
```

```
                                                    comprimento( L,N
),
                                                    N == 0).


% ----- Nao pode haver conhecimento negativo igual aquele
cuidado se queremos adicionar negativo -------------

+( -cuidado( Id,Data,U,P,D,C,I ) ) :: (solucoes( A,-cuidado(
Id,Data,U,P,D,C,I ),L ),
                                                    comprimento( L,N
),
                                                    N =< 2).


% --------------- Penso que na remocao de prestadores e utentes
temos de ter atencao em eles estarem também em cuidados mas n se
saber se sao mm eles------------------------

-utente( A,N,I,M ) :: ( solucoes( D,( excecao( cuidado(
ID,Data,A,P,D,C,In ) ) ),L ),
                                    comprimento( L,Aux ),
                                    Aux == 0
                                    ).



% --------------- Penso que na remocao de prestadores e utentes
temos de ter atencao em eles estarem também em cuidados mas n se
saber se sao mm eles------------------------

-prestador( A,N,I,M ) :: ( solucoes( D,( excecao( cuidado(
ID,Data,U,A,D,C,In ) ) ),L ),
                                    comprimento( L,Aux ),
                                    Aux == 0
                                    ).


%------------------------- Predicados de insercao --------------
----------------------


%---------------- Adicionar utente ----------------------------
-------
```

```
% Extensao do predicado adicionarUtente: Id,Nome,Idade,Morada ->
{V,F}


adicionarUtente( IdUtente,Nome,Idade,Morada ) :-

     evolucao( utente( IdUtente,Nome,Idade,Morada ) ).



%---------------- Adicionar prestador -------------------------
----------

% Extensao do predicado adicionarPrestador:
Id,Nome,Especialidade,ListaInstituicao -> {V,F}


adicionarPrestador( IdPrestador,Nome,Especialidade,ListaI ) :-

     evolucao( prestador( IdPrestador,Nome,Especialidade,ListaI
) ).


%---------------- Adicionar cuidado ---------------------------
--------

% Extensao do predicado adicionarCuidado:
Data,IdUtente,IdPrestador,Descricao,Custo,Instituicao -> {V,F}


adicionarCuidado(
Id,Data,IdUtente,IdPrestador,Descricao,Custo,Instituicao ) :-

     evolucao( cuidado(
Id,Data,IdUtente,IdPrestador,Descricao,Custo,Instituicao ) ).


%------------------------- Predicados de remocao ---------------
--------------------


%---------------- Remover utente ------------------------------
-----

% Extensao do predicado retirarUtente: Id -> {V,F}


retirarUtente( IdUtente ) :-
```

```
                            inevolucao( utente( IdUtente,_,_,_ )
).



%---------------- Remover prestador ----------------------------
--------

% Extensao do predicado retirarPrestador: Id -> {V,F}


retirarPrestador( IdPrestador ) :-
                                        inevolucao( prestador(
IdPrestador,_,_,_ ) ).


%---------------- Remover cuidado ------------------------------
------

% Extensao do predicado retirarCuidado:
Data,IdUtente,IdPrestador,Descricao,Custo,Instituiao -> {V,F}

retirarCuidado(
Id,Data,IdUtente,IdPrestador,Descricao,Custo,Instituiao ) :-

     inevolucao( cuidado(
Id,Data,IdUtente,IdPrestador,Descricao,Custo,Instituiao ) ).




%--------------------------------- - - - - - - - - - -  -  -  -
-   -

% Extensao do meta-predicado demo: Questao,Resposta -> {V,F}


demo( Questao,verdadeiro ) :-

    Questao.

demo( Questao,falso ) :-

    -Questao.

demo( Questao,desconhecido ) :-

    nao( Questao ),

    nao( -Questao ).
```

```
demonstrador( A+B,R ) :- demonstrador( A,T ),
                                 demonstrador( B,Q ),
                                 ou( T,Q,R ).
demonstrador( A^B,R ) :- demonstrador( A,T ),
                                 demonstrador( B,Q ),
                                 e( T,Q,R ).
demonstrador( A,R ) :- demo( A,R ).

%----------- ou: Termo,Termo,Resposta -> {V,F}
ou( verdadeiro,_,verdadeiro ).
ou( _,verdadeiro,verdadeiro ).
ou( desconhecido,desconhecido,desconhecido ).
ou( desconhecido,falso,desconhecido ).
ou( falso,desconhecido,desconhecido ).
ou( falso,falso,falso ).

%----------- e: Termo,Termo,Resposta -> {V,F}
e( falso,_,falso ).
e( _,falso,falso ).
e( desconhecido,desconhecido,desconhecido ).
e( desconhecido,verdadeiro,desconhecido ).
e( verdadeiro,desconhecido,desconhecido ).
e( verdadeiro,verdadeiro,verdadeiro ).



%---------------------------- Predicados de Evolucao -----------
--------

%-------------------------------------------------
% Extensao do predicado evolucao: Termo -> {V,F}
evolucao( Termo ) :- solucoes( Inv,+Termo::Inv,S ),
                           inserir( Termo ),
```

```
                        testar( S ).

%-----------------------------------------------------
% Extensao do predicado inserir: Predicado -> {V,F}
inserir( P ) :- assert( P ).
inserir( P ) :- retract( P ), !, fail.


%-----------------------------------------------------
% Extensao do predicado involucao: Termo -> {V,F}
inevolucao( Termo ) :- solucoes(Inv,-Termo::Inv,S),
                       remover(Termo),
                       testar(S).


%-----------------------------------------------------
% Extensao do predicado remover: Predicado -> {V,F}
remover( P ) :- retract( P ).
remover( P ) :- assert( P ), !, fail.

%-----------------------------------------------------
% Extensao do predicado testar: ListaPredicado -> {V,F}
testar( [] ).
testar( [X|R] ) :- X,
                   testar( R ).

%------------------------- Predicado nao - negacao por falha na
prova ----------------
% Extensao do predicado nao: Termo -> {V,F}

nao( T ) :- T, !, fail.
nao( T ).
```

```
%-----------------------------------------------------

% Extensao do predicado solucoes:
Termo,Questao,Solucao(ListaQuestao) -> {V,F}

solucoes(X,Y,Z) :- findall(X,Y,Z).



comprimento(R,S) :- length(R,S).



%-----------------------------------------------------

% Extensao do predicado natural: Numero -> {V,F}

natural( 1 ).

natural( N ) :- R is N-1,

                R > 0,

            natural(R).



%-------------------------------------------------------------

% Extensao do predicado comparaData: Criterio(>=,<=), Data, Data
-> {V,F}


comparaData(igual,data(A1, M1, D1, H1), data(A2, M2, D2, H2)) :-

     A1 == A2,

     M1 == M2,

     D1 == D2,

     H1 == H2.

comparaData(>=,data(A1, M1, D1, H1), data(A2, M2, D2, H2)) :-

     A1 > A2.

comparaData(>=,data(A1, M1, D1, H1), data(A2, M2, D2, H2)) :-

     A1 == A2,

     M1 > M2.

comparaData(>=,data(A1, M1, D1, H1), data(A2, M2, D2, H2)) :-

     A1 == A2,
```

```
      M1 == M2,
      D1 > D2.
comparaData(>=,data(A1, M1, D1, H1), data(A2, M2, D2, H2)) :-
      A1 == A2,
      M1 == M2,
      D1 == D2,
      H1 >= H2.
comparaData(<=,data(A1, M1, D1, H1), data(A2, M2, D2, H2)) :-
      A1 < A2.
comparaData(<=,data(A1, M1, D1, H1), data(A2, M2, D2, H2)) :-
      A1 == A2,
      M1 < M2.
comparaData(<=,data(A1, M1, D1, H1), data(A2, M2, D2, H2)) :-
      A1 == A2,
      M1 == M2,
      D1 < D2.
comparaData(<=,data(A1, M1, D1, H1), data(A2, M2, D2, H2)) :-
      A1 == A2,
      M1 == M2,
      D1 == D2,
      H1 =< H2.
```