Sistemas de Representação de Conhecimento e Raciocínio - Trabalho Prático 1

Programação em Lógica

Grupo 17

Rita Gomes	João Torres	José Manso	José Reis
A87960	A85846	A87961	A87980

9 de Abril de 2021

Resumo
O presente documento representa o relatório do primeiro trabalho prático da Unidade Curricular de Sistemas de Representação de Conhecimento e Raciocínio, lecionada no terceiro ano do curso de Mestrado Integrado em Engenharia Informática. O trabalho é composto por duas fases, às quais a cada uma delas são propostas diversas funcionalidades a implementar no sistema.

Conteúdo

Lista de Figuras

1 Introdução

No âmbito da Unidade Curricular de Sistemas de Representação de Conhecimento e Raciocínio, foi-nos proposto o desenvolvimento, na linguagem PROLOG, de um sistema com capacidade para caracterizar um universo de discurso na área da vacinação global da população portuguesa, dentro do contexto COVID 19 que se vive atualmente.
Este projeto é composto por duas fases, sendo este relatório um ficheiro único representativo do trabalho efetuado.
Com isso, é necessário satisfazer um conjunto de funcionalidades para que o sistema funcione da melhor forma. Ao longo do documento, iremos apresentar os requerimentos necessários ao sucesso da resolução do projeto, seguidos dos procedimentos utilizados para o desenvolvimento do sistema e das explicações de cada funcionalidade desenvolvida. Acrescentaremos também as nossas funcionalidades extra que complementam a utilidade do nosso trabalho prático, finalizando com a tradicional conclusão.

2 Preliminares

De maneira a proceder de forma correta à resolução do enunciado proposto, é necessário:

- Compreender os conceitos da linguagem PROLOG: predicado, invariantes, factos, etc;
- Entrar "em sintonia"com a programação declarativa usada em PROLOG;
- Perceber de que forma o enunciado proposto se enquadra nos conhecimentos abordados nas aulas;
- Pesquisar sobre o tema do enunciado de modo a maximizar a utilidade do projeto no que toca ao conjunto de funcionalidades que este tem a capacidade de oferecer.

Com estes requerimentos, a compreensão e respetiva resolução do enunciado proposto torna-se num processo bastante mais simples.

3 Descrição do trabalho e Análise de resultados

3.1 Inserção do Conhecimento

De maneira a testar os diversos predicados, procedemos a uma inserção de conhecimento de modo a complementar a nossa base. Primeiramente, foram inseridos os utentes, constituídos pelo seu identificador, número de segurança social, nome, data de nascimento, email, telefone, morada, profissão, lista de doenças crónicas (caso haja), e identificador do centro de saúde no qual está registado.

```
% Povoamento do predicado utente: #Idutente, Nº Segurança_Social, Nome, Data_Nasc, Email, Telefone, Morada,
%                                  Profissão, [Doenças_Crónicas], #CentroSaúde -> {V,F}
utente(0 , '970530380', 'Joao Tadeu'          , '10-01-1999', 'joaoTadeu99@gmail.com'        , '939827435', 'Rua Perto de Braga'     , 'Estudante'  , [], 0).
utente(1 , '988877885', 'Jose Reis'           , '10-04-1999', 'josereis@hotmail.com'         , '910753465', 'Rua Santa Eugenia'      , 'Enfermeiro' , ['Asma'], 1).
utente(2 , '606028459', 'Jose Manso'          , '24-11-2000', 'josemansinho@yahoo.com'       , '935581435', 'Rua Mais Perto de Braga', 'Medico'     , ['Parkinson'], 1).
utente(3 , '722084637', 'Rita Gomes'          , '31-05-2000', 'ritagomees@gmail.com'         , '921293006', 'Rua em Braga'           , 'Estudante'  , [], 2).
utente(4 , '925775811', 'Timoteo Mata'        , '25-12-1970', 'titeomata@outlook.com'        , '990247466', 'Rua da Mata'            , 'Coveiro'    , ['Osteoporose'], 2).
utente(5 , '724866051', 'Luciana Abreu'       , '25-05-1985', 'floribela@sic.pt'             , '984310706', 'Rua do Professor'       , 'Atriz'      , [], 3).
utente(6 , '214583755', 'Alberto Pereira', '01-01-1945', 'betinho@yahoo.com', '979382414', 'Avenida Central', 'Reformado'  , ['Alzheimer', 'Colesterol', 'Hipertensao'], 3).
utente(7 , '868975797', 'Ricardo Quaresma'    , '26-09-1983', 'rq@vsc.pt'                    , '988370144', 'Avenida Sao Goncalo'    , 'Futebolista', [], 3).
```

Figura 1: Inserção de Conhecimento sobre Utentes

De seguida, a inserção de conhecimento sobre os centros de saúde, contendo o identificador respetivo, nome, morada, telefone e email:

```
% Povoamento do predicado centro_saude: #Idcentro, Nome, Morada, Telefone, Email -> {V,F}
centro_saude(0, 'Centro de Barceladas Catitas'    , 'Casa do Rafa nº 22'        , 252144785, 'cbc@dgs.gmail.com').
centro_saude(1, 'clinica dos Cowboys'             , 'Rua Cringe, nº 11'         , 253412238, 'cowboys@dgs.outlook.pt').
centro_saude(2, 'Centro de Saude Pires & Chávenas', 'Avenida do 9 e meio'       , 254961222, 'piresandchavenas@dgs.lol.pt').
centro_saude(3, 'Centro de Saude Mansos Lda.'     , 'Praca dos merges conflicts', 254786950, 'centromansos@dgs.mimimi.com').
```

Figura 2: Inserção de Conhecimento sobre Centros de Saúde

Depois, a inserção de conhecimento sobre o staff dos centros de saúde, caracterizados pelos respetivos identificadores (seus e dos seus centros), nome e email.

```
% Povoamento do predicado staff: #IdStaff, #Idcentro, Nome, Email, Fase -> {V,F}
staff(0 , 0, 'Lewis Hamilton', 'lewishamilton@dgs.cbc.com').
staff(1 , 0, 'Batmene'       , 'batmene@dgs.cbc.com').
staff(2 , 0, 'Andre Andre'   , 'andretwice@dgs.cbc.com').
staff(3 , 0, 'Marcus Edwards', 'edwards.marcus@dgs.cbc.com').

staff(4 , 1, 'Bruno Varela'  , 'brunovarela@dgs.cowboys.pt').
staff(5 , 1, 'Rochinha'      , 'rochinhaM@dgs.cowboys.pt').
staff(6 , 1, 'Pedro Henrique', 'pedrohenrique@dgs.cowboys.pt').
staff(7 , 1, 'Lyle Foster'   , 'lylefoster@dgs.cowboys.pt').

staff(8 , 2, 'Joao Carlos Teixeira', 'jct@dgs.pc.pt').
staff(9 , 2, 'Max Verstappen'      , 'verstappen@dgs.pc.pt').
staff(10, 2, 'Daniel Ricciardo'    , 'danielric@dgs.pc.pt').
staff(11, 2, 'Cristina Ferreira'   , 'criscris@dgs.pc.pt').

staff(12, 3, 'Claire Williams', 'williams.claire@dgs.mansos.com').
staff(13, 3, 'Serena Williams', 'serenawilliams@dgs.mansos.com').
staff(14, 3, 'Maria Sharapova', 'sharapova.m@dgs.mansos.com').
staff(15, 3, 'Dulce Felix'    , 'dulcefelix@dgs.mansos.com').
```

Figura 3: Inserção de Conhecimento sobre membros do Staff

Segue-se a inserção sobre a vacinação, que contém o identificador do membro do staff que vacinou, o identificador do utente, a data, o tipo de vacina e o número da toma (primeira ou segunda).

```
% Povoamento do predicado vacinacao_Covid: #Staff, #utente, Data, Vacina, Toma, Fase -> {V,F}
vacinacao_Covid(0 , 1, '20-01-2021', 'Oxford'     , 1, 1).
vacinacao_Covid(2 , 1, '20-01-2021', 'Oxford'     , 2, 1).
vacinacao_Covid(4 , 2, '20-01-2021', 'Pfizer'     , 1, 3).
vacinacao_Covid(6 , 2, '20-01-2021', 'Pfizer'     , 2, 3).
vacinacao_Covid(8 , 3, '08-02-2021', 'BioNYech'   , 1, 3).
vacinacao_Covid(12, 7, '01-02-2021', 'AstraZeneca', 1, 2).
```

Figura 4: Inserção de Conhecimento sobre registos de Vacinação

Para complementar informação, decidimos acrescentar conhecimento sobre registos de consultas, tratamentos e receitas médicas. Ambos contém o seu identificador, o identificador do staff e do utente, a data do registo e o respetivo tipo de registo.

```
% Povoamento do predicado consulta: #Id, #Staff, #utente, Data, Tipo de Consulta -> {V,F}
consulta(0, 0 , 0, '12-03-2021', 'Pedologia').
consulta(1, 4 , 1, '14-02-2021', 'Clinica Geral').
consulta(2, 4 , 1, '15-02-2021', 'Exames medicos').
consulta(3, 12, 6, '11-01-2021', 'Cardiologia').
consulta(4, 12, 6, '25-01-2021', 'Ginecologia').
consulta(5, 13, 6, '29-01-2021', 'Pedologia').
consulta(6, 15, 7, '06-03-2021', 'Exames medicos').
```

Figura 5: Inserção de Conhecimento sobre Consultas

```
% Povoamento do predicado tratamento: #Id, #Staff, #utente, Data, Tipo de tratamento -> {V,F}
tratamento(0,  0, 0, '22-03-2021', 'Tratamento pedologico').
tratamento(1,  5, 2, '20-02-2021', 'Exames desportivos').
tratamento(2,  4, 1, '17-03-2021', 'Exames desportivos').
tratamento(3, 12, 6, '14-01-2021', 'Exame de resistencia').
tratamento(4, 13, 6, '12-02-2021', 'Tratamento pedologico').
tratamento(5, 10, 4, '23-01-2021', 'Raio X').
```

Figura 6: Inserção de Conhecimento sobre Tratamentos

```
% Povoamento do predicado receita: #Id, #Staff, #utente, Data, Nome -> {V,F}
receita(0, 4 , 1, '14-02-2021', 'Be-nu-ron').
receita(1, 12, 6, '11-01-2021', 'Brufen').
receita(2, 13, 6, '12-02-2021', 'Montelucaste').
receita(3, 0,  0, '22-03-2021', 'Kestine').
```

Figura 7: Inserção de Conhecimento sobre Receitas Médicas

3.2 Resolução das Funcionalidades propostas

Para as funcionalidades mínimas, o grupo desenvolveu da seguinte forma:

- **Permitir a definição de fases de vacinação, definindo critérios de inclusão de utentes nas diferentes fases**:

  Os critérios de vacinação funcionam, de certa forma, através um sistema de pontuação, em que quanto maior esta for, mais prioritário será o utente em questão. Por isso, a pontuação funciona da seguinte forma:

  - **Profissão de risco**: 2 pontos
  - **Idade maior do que 65 anos**: 1 ponto
  - **Doenças crónicas**: 2 pontos por cada uma

  As profissões de risco são: médico(a), enfermeiro(a), dentista, professor(a), educador(a) de infância e bombeiro(a).

  Para além disso, pensamos num sistema de distribuição de vacinas, pois como sabemos, nem sempre existe os recursos necessários para que haja um equilíbrio entre oferta e procura (neste caso, a procura corresponde ao número de utentes). Estendemos então o predicado "fasesVacinacao".

  A *query* recebe como argumentos quantas fases vai haver e o número de vacinas por fase. Esta query pode ser usada de duas maneiras: para definir fases de vacinação para a primeira toma da vacina ou para a segunda. No caso da primeira, é utilizado o predicado -vacinados"para obter a lista dos utentes não vacinados, depois percorre a lista e associa a cada um destes uma pontuação correspondente ao risco. Sendo assim, a lista dos pares (Risco, Utente) é ordenada decrescentemente pelos riscos de cada utente e, para finalizar, basta distribuir os utentes pelas fases, tendo em conta o número de fases e o número de vacinas por fase. Desta maneira, quanto maior o risco de um utente, mais cedo vai tomar a vacina. No caso da segunda toma o processo é o mesmo mas, em vez de começar com o predicado -vacinados", começa com o predicado "segundaToma", para obter os utentes que já receberam a primeira toma mas não receberam a segunda.

  Apesar de assumirmos que o número máximo de fases é 3, neste predicado é dada a liberdade para que esse número tome qualquer valor.

```
% Primeira query para a primeira toma
fasesVacinacao(Fases, Vacinas, X) :- -vacinados(NaoVacinados),
                                     associaRiscoUtente(NaoVacinados, [], L),
                                     distribui(Fases, Vacinas, Vacinas, L, [], [],X),!.

distribui(0,_,_,_,_,R,R).
distribui(_,_,_,[],L2,LF,R) :- append([L2],LF,R).
distribui(F,V,0,L1,L2,LF,R) :- F2 is F - 1, distribui(F2,V,V,L1,[],[L2|LF],R).
distribui(F,V1,V2,[(_,N)|T],L,LF,R) :- V3 is V2 - 1, distribui(F,V1,V3,T,[N|L],LF,R).
```

Figura 8: Fases de Vacinação

- **Identificar pessoas vacinadas**:

  Em relativamente pouco tempo, esta funcionalidade não nos deixou dúvidas quanto àquele que seria o procedimento a ter em conta para obter o resultado pretendido: primeiramente, necessitamos de desenvolver um predicado (tal como na figura seguinte) "vacinado"que apenas retorna os utentes que foram vacinados. De seguida, através do predicado "solucoes"(resultante do predicado original "findall") que pesquisa todas as ocorrências de um determinado predicado, armazena-os numa lista e, apenas por questões estéticas, ordena-os alfabeticamente.

```
% Extensao do predicado vacinados: [Utente] -> {V,F}
vacinados(X) :- solucoes(N , vacinado(N), L),
                ordena(L,X).

% Extensao do predicado vacinado: {NomeUtente} -> {V,F}
vacinado(X) :- utente(N,_,X,_,_,_,_,_,_,_),
               vacinacao_Covid(_,N,_,_,_,_).
```

Figura 9: Identificar pessoas vacinadas

- **Identificar pessoas não vacinadas**:

  Esta funcionalidade é resultante da anterior, pelo que, a partir dos utentes todos, simplesmente elimina os que já foram vacinados, restando apenas os utentes não vacinados.

```
% Identificar pessoas não vacinadas
-vacinados(X) :- solucoes(N, -vacinado(N), L),
                 ordena(L,X).

-vacinado(X) :- utente(N,_,X,_,_,_,_,_,_,_),
                nao(vacinacao_Covid(_,N,_,_,_,_)).
```

Figura 10: Identificar pessoas não vacinadas

- **Identificar pessoas vacinadas indevidamente**:

  Tendo como base o sistema de pontuações mencionado anteriormente, esta funcionalidade foi pensada fase a fase. Por isso, o predicado recebe dois argumentos: o número da fase em questão e a lista a devolver. Sendo assim, as pontuações definidas para as três fases (número máximo definido) são:

  - **1ª Fase**: apenas utentes com pelo menos 2 pontos
  - **2ª Fase**: apenas utentes com 1 ponto
  - **3ª Fase**: utentes com 0 pontos

```
% Identificar pessoas vacinadas indevidamente
vacInd(Lista, Fase) :- vacinadosFase(L2, Fase),
                       ordena(L2, L),
                       vacIndAux(L, Fase, [], Lista).

vacinadoFase(Nome, Fase) :- utente(Id,_,Nome,_,_,_,_,_,_),
                            vacinacao_Covid(_,Id,_,_,_,Fase).
vacinadosFase(L, Fase) :- solucoes(N, vacinadoFase(N, Fase), X),
                          ordena(X, L).

vacIndAux([],_, R, R).
vacIndAux([H|T], 1, L2, R) :- calculaRiscoUtente(H, Risco),
                              Risco < 2, !,
                              vacIndAux(T, 1, [H|L2], R).
vacIndAux([_|T], 1, L2, R) :- vacIndAux(T, 1, L2, R).
vacIndAux([H|T], 2, L2, R) :- calculaRiscoUtente(H, Risco),
                              Risco \= 1, !,
                              vacIndAux(T, 2, [H|L2], R).
vacIndAux([_|T], 2, L2, R) :- vacIndAux(T, 2, L2, R).
vacIndAux([H|T], 3, L2, R) :- calculaRiscoUtente(H, Risco),
                              Risco \= 0, !,
                              vacIndAux(T, 3, [H|L2], R).
vacIndAux([_|T], 3, L2, R) :- vacIndAux(T, 3, L2, R).
vacIndAux(_,_,[],[]).
```

Figura 11: Identificar pessoas vacinadas indevidamente

A decisão de limitar o número de fases vem de não considerarmos que existe motivo para exceder a complexidade pedida neste sistema.

- **Identificar pessoas não vacinadas e que são candidatas à vacinação**:

  Ao invés de tratar esta funcionalidade por fases, abordar uma estratégia diferente. Como sabemos, qualquer utente tem o direito de ser vacinado. No entanto, no nosso entendimento, basta que um utente possua uma doença ou profissão de risco para se considerar candidata a vacinação.

```
% Identificar pessoas não vacinadas e que são candidatas a vacinação
candidatos(X) :- -vacinados(Lista), candidatosAux(Lista, [], X).

candidatosAux([],L,L).
candidatosAux([H|T], L, R) :- calculaRiscoUtente(H, Risco), Risco > 0, candidatosAux(T,[H|L],R),!.
candidatosAux([_|T], L, R) :- candidatosAux(T,L,R).
```

Figura 12: Identificar pessoas não vacinadas e candidatas à vacinação

- **Identificar pessoas a quem falta a segunda toma da vacina**:

  Apenas foi necessária a criação de um predicado "segundaToma", que começa por procurar todos os utentes e, através do seu nome, verifica se já tomou a primeira dose da vacina. Em caso afirmativo, verifica se já tomou também a segunda dose, ao qual, em caso negativo, adiciona à lista pretendida.

```
% Identificar pessoas a quem falta a segunda toma da vac
segundaToma(X) :- solucoes(U, segundaTomaAux(U), X).

segundaTomaAux(X) :- utente(N,_,X,_,_,_,_,_,_,_),
                     vacinacao_Covid(_,N,_,_,1,_),
                     nao(vacinacao_Covid(_,N,_,_,2,_)).
```

Figura 13: Pessoas a quem falta a segunda toma

- **Representar conhecimento positivo e negativo**:

  De modo a identificar conhecimento positivo de negativo, necessitamos de identificar e definir o conhecimento positivo. Desta forma, criámos um predicado que afirma que uma entidade só é realmente um predicado é verdadeiro se existir um registo do mesmo e não se tratar de uma exceção. Em termos práticos, podemos demonstrar o caso do utente:

```
utente(I,S,N,D,E,Nr,M,P,D,C) :- utente(I,S,N,D,E,Nr,M,P,D,C),
                                 nao(excecao(utente(I,S,N,D,E,Nr,M,P,D,C))).
```

Figura 14: Conhecimento positivo do Utente

Desta forma, qualquer entidade pode admitir exceções. Por exemplo, existem aves que não têm a capacidade de voar, tratando-se portanto de exceções. O conhecimento negativo é realizado de forma inversa. No

```
% Conhecimento negativo de utente
-utente(I,S,N,D,E,Nr,M,P,D,C) :- nao(utente(I,S,N,D,E,Nr,M,P,D,C)),
                                 nao(excecao(utente(I,S,N,D,E,Nr,M,P,D,C))).
```

Figura 15: Conhecimento negativo do Utente

caso do utente, começamos por verificar que não se trata de um utente nem de uma exceção, como podemos observar na figura seguinte:

Este procedimento foi aplicado a outros registos existentes: membros de staff, centros de saúde, membros de staff, registos de vacinação, consultas e receitas médicas e tratamentos. Por esse motivo, considerámos que seria suficientemente útil apresentar sucintamente exemplos que apenas demonstram de que forma são criados.

- **Representar casos de conhecimento imperfeito, pela utilização de valores nulos de todos os tipos estudados**:

  Existem três tipos de conhecimento imperfeito: incerto, impreciso e interdito. Quanto ao primeiro tipo, acontece quando um predicado está definido corretamente, no entanto simplesmente não se conhece o seu valor verdadeiro (exemplo: dados estatísticos). O conhecimento impreciso forma-se com predicados que são vagos por si. Já o interdito trata-se de informação que é impedida de se aceder.

  Decidimos atribuir o valor *null* sempre que algum parâmetro seja fruto de conhecimento incerto e *interdito* para os casos de conhecimento interdito.

  Seguem-se exemplos de cada um dos tipos de conhecimento imperfeito aplicados no sistema.

  - **Incerto**: o utente 'Luciana Abreu' não tem número de segurança social, estando esse campo preenchido com o valor *null*.
  - **Interdito**: ainda na mesma figura, o utente 'Alberto Pereira' não permite que possamos aceder ao campo da morada.

```
%-------------------------------- - - - - - - - - - -  -  -  -  -   -   -   -    -    -    -     -
% Povoamento do predicado utente: #Idutente, Nº Segurança_Social, Nome, Data_Nasc, Email, Telefone, Morada,
%                                            Profissão, [Doenças_Crónicas], #CentroSaúde -> {V,F}
utente(0 , '970530380', 'Joao Tadeu'      , '10-01-1999', 'joaoTadeu99@gmail.com' , '939827435', 'Rua Perto de Braga'      ,
utente(1 , '988877885', 'Jose Reis'       , '10-04-1999', 'josereis@hotmail.com'  , '910753465', 'Rua Santa Eugenia'       ,
utente(2 , '606028459', 'Jose Manso'      , '24-11-2000', 'josemansinho@yahoo.com', '935581435', 'Rua Mais Perto de Braga',
utente(3 , '722084637', 'Rita Gomes'      , '31-05-2000', 'ritagomees@gmail.com'  , '921293006', 'Rua em Braga'            ,
utente(5 ,  null      , 'Luciana Abreu'   , '25-05-1985', 'floribela@sic.pt'      , '984310706', 'Rua do Professor'        ,
utente(6 , '214583755', 'Alberto Pereira' , '01-01-1945', 'betinho@yahoo.com'     , '979382414',  interdito                ,
utente(7 , '868975797', 'Ricardo Quaresma', '26-09-1983', 'rq@vsc.pt'             , '988370144', 'Avenida Sao Goncalo'     ,
```

Figura 16: Conhecimento incerto e interdito

  - **Impreciso**: o utente Timóteo Mata está associado a dois centros de saúde.

```
excecao(utente(4 ,'925775811','Timoteo Mata','25-12-1970','titeomata@outlook.com','990247466','Rua da Mata', 'Coveiro', ['Osteoporose'], 2)).
excecao(utente(4 ,'925775811','Timoteo Mata','25-12-1970','titeomata@outlook.com','990247466','Rua da Mata', 'Coveiro', ['Osteoporose'], 3)).
```

Figura 17: Conhecimento impreciso

- **Manipular invariantes que designem restrições à inserção e à remoção de conhecimento do sistema**:

  Invariante Estrutural para a inserção de Utentes/Centros de Saúde/Staff

  Apenas é possível adicionar informação à base de conhecimento se não existir informação repetida. Tanto para o utente como para o centro e o staff, a informação é considerada repetida se tiver o mesmo identificador. No utente também existe o caso de ter o mesmo número de segurança social, o mesmo acontece no centro de saúde para o número de telemóvel.

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

Figura 18: Invariantes Estruturais para inserção de Utentes

```
+centro_saude(Id,_,_,_,_) :: (solucoes(Id, centro_saude(Id,_,_,_,_), S),
                              comprimento(S,N),
                              N == 1).
+centro_saude(_,_,_,Nr,_) :: (solucoes(Nr, centro_saude(_,_,_,Nr,_), S),
                              comprimento(S,N),
                              N == 1).
```

Figura 19: Invariantes Estruturais para inserção de Centros de Saúde

```
+staff(Id,_,_,_) :: (solucoes(Id, staff(Id,_,_,_), S),
                     comprimento(S,N), N==1).
```

Figura 20: Invariante Estrutural para inserção de Staff

Invariante Referencial para a inserção de Utentes/Staff/Vacinação

Para a inserção de informação ser válida, é necessário verificar se todos os fatores para essa informação existem. Por outras palavras, sempre que se insere um identificador de outra informação, por exemplo no utente, esse identificador refere-se ao do Centro de Saúde, e é necessário verificar se existe alguma informação sobre esse identificador.

```
+utente(_,_,_,_,_,_,_,_,_,Centro) :: (existeCentroId(Centro)).
```

Figura 21: Invariante Referencial para a inserção de Utentes

```
+staff(_,Centro,_,_) :: (existeCentroId(Centro)).
```

Figura 22: Invariante Referencial para a inserção de Staff

```
+vacinacao_Covid(Staff,_,_,_,_,_) :: (existeStaffId(Staff)).
+vacinacao_Covid(_,Utente,_,_,_,_) :: (existeUtenteID(Utente)).
```

Figura 23: Invariantes Referenciais para a inserção de Vacinação

Invariante Referencial para a remoção de Centros de Saúde

Só é possível remover um centro de saúde se não existir nenhum staff na base de conhecimento que pertença a esse centro.

```
-centro_saude(Centro,_,_,_,_) :: (solucoes(Centro, staff(_,Centro,_,_), S),
                                  comprimento(S,N),
                                  N == 0).
```

Figura 24: Invariante Referencial para a remoção de Centros de Saúde

Invariantes Referenciais para a inserção de Vacinação

Para a inserção da vacinação, há mais três regras para além da referida acima.

1. um utente só pode levar a segunda toma se já levou a primeira.
2. um utente não pode levar a mesma toma duas vezes;
3. a fase de vacinação tem de ser inferior a 4;

```
+vacinacao_Covid(_,Utente,_,_,Toma,_) :: (solucoes((Utente, Toma), vacinacao_Covid(_,Utente,_,_,Toma,_), S),
                                          comprimento(S,N), N==1).
+vacinacao_Covid(_,_,_,_,_,Fase) :: (Fase < 4).
+vacinacao_Covid(_,Utente,_,_,2,_) :: (vacinacao_Covid(_,Utente,_,_,1,_)).
```

Figura 25: Invariantes Referenciais para a inserção de Vacinação

Invariantes para as funcionalidades extra

A adição de novos predicados (explicados no capítulo 3.1) motivou ao desenvolvimento de novos invariantes (referenciais e estruturais) presentes nas figuras seguintes.

```
% Invariante Estrutural: não permite a inserção de um identificador já existente
+consulta(Id,_,_,_,_) :: (solucoes(Id, consulta(Id,_,_,_,_), S),
                          comprimento(S,N),
                          N==1).
% Invariante Referencial: garante que o identificador do Staff existe
+consulta(_,Staff,_,_,_) :: (existeStaffId(Staff)).
% Invariante Referencial: garante que o identificador do Utente existe
+consulta(_,_,Utente,_,_) :: (existeUtenteID(Utente)).
% Invariante Referencial: não permite a inserção de conhecimento interdito
+consulta(I,S,Utente,D,T) :- (solucoes(Utente, (consulta(I,S,Utente,D,T), nao(nulo(Utente))), S),
                             comprimento(S,N),
                             N == 0).
```

Figura 26: Invariantes para a inserção de Consultas

```
% Invariante Estrutural: garante que não existem tratamentos com o mesmo identificador
+tratamento(Id,_,_,_,_) :: (solucoes(Id, tratamento(Id,_,_,_,_), L),
                            comprimento(L,N),
                            N == 1).
% Invariante Referencial: garante que o membro do staff está presente na base de conhecimento
+tratamento(_,Id,_,_,_) :: existeStaffId(Id).
% Invariante Referencial: garante que o utente está presente na base de conhecimento
+tratamento(_,_,Id,_,_) :: existeUtenteID(Id).
```

Figura 27: Invariantes para a inserção de Tratamentos

```
% Invariante Estrutural: não permite a inserção de um identificador já existente
+receita(Id,_,_,_,_) :: (solucoes(Id, receita(Id,_,_,_,_), S),
                         comprimento(S, N),
                         N == 1).
% Invariante Referencial: garante que o identificador do Staff existe
+receita(_,Staff,_,_,_) :: (existeStaffId(Staff)).
% Invariante Referencial: garante que o identificador do Utente existe
+receita(_,_,Utente,_,_) :: (existeUtenteID(Utente)).
```

Figura 28: Invariantes para a inserção de Receitas

- **Lidar com a problemática da evolução do conhecimento, criando os procedimentos adequados**:

  A evolução do conhecimento tem de ser realizada com o predicado *evolucao* de modo a inserir conhecimento. O predicado começa por receber um termo para adiciona-o à base de conhecimento. De seguida, procura todos os invariantes do predicado para esse termo e testa-os. Caso pelo menos um deles seja falso, o termo é resolvido da base por *backtracking* do predicado *insercao*. Caso contrário, o termo mantém-se na base de conhecimento.

```
% Extensão do predicado que permite a evolucao do conhecimento
evolucao(Termo) :- solucoes(Invariante, +Termo::Invariante, Lista),
                   insercao(Termo),
                   teste(Lista).

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

teste([]).
teste([R|LR]) :- R, teste(LR).

%------------------------------ - - - - - - - - - - - - - - - - - -
% Extensão do predicado que permite a involucao do conhecimento
involucao(Termo) :- solucoes(Invariante, -Termo::Invariante, Lista),
                    remocao(Termo),
                    teste(Lista).

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

Figura 29: Evolução de conhecimento

Já o predicado *involucao* funciona da maneira oposta, sendo que primeiramente, remove o termo da base e, caso não cumpra pelo menos um dos invariantes, insere-o novamente. Em caso contrário, o termo permanece removido.

- **Desenvolver um sistema de inferência capaz de implementar os mecanismos de raciocínio inerentes a estes sistemas**:

  Esta funcionalidade verifica se um dado predicado é verdadeiro (existe na base de conhecimento), falso (não existe) ou desconhecido. Este último aspeto corresponde aos casos em que a base de conhecimento não sabe o valor lógico de um determinado predicado, sendo portanto desconhecido. Para isso, utilizamos o predicado "demo", presente na figura 22. Seguem-se também alguns exemplos de utilização deste predicado.

```
% Extensao do meta-predicado demo (Sistema de inferencia)
demo(Questao, verdadeiro) :- Questao.
demo(Questao, falso) :- -Questao.
demo(Questao, desconhecido) :- nao(Questao), nao(-Questao).
```

Figura 30: Predicado Demo

```
?- demo(vacinado('Jose Manso'),X).
X = verdadeiro .
```

Figura 31: Predicado Demo com retorno verdadeiro

```
?- demo(vacinado('Luciana Abreu'),X).
X = falso .
```

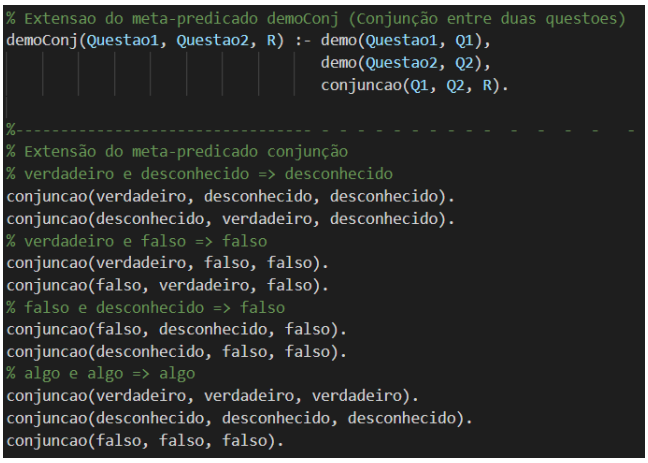
Figura 32: Predicado Demo com retorno falso

```
?- demo(vacinado('Pato Donald'),X).
X = desconhecido.
```

Figura 33: Predicado Demo com retorno desconhecido

De seguida, desenvolvemos os predicados *demoConj* e *demoDisj* que visam comparar dois predicados. No entanto, enquanto o primeiro apenas é verdadeiro se ambos os predicados forem verdadeiros, o segundo necessita que apenas um deles o seja.

Por isso, foi necessário definir uma conjunção e disjunções, bem como todos os casos possíveis, sendo que que os resultados obtidos estão de acordo com as tabelas de verdade abordadas em anos anteriores, tanto para a disjunção como para a conjunção de duas questões. Segue-se a extensão predicado *demoConj*:

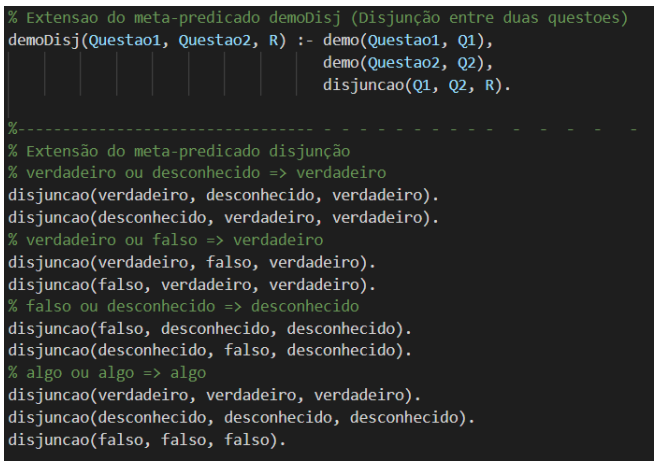


```
% Extensao do meta-predicado demoConj (Conjunção entre duas questoes)
demoConj(Questao1, Questao2, R) :- demo(Questao1, Q1),
                                   demo(Questao2, Q2),
                                   conjuncao(Q1, Q2, R).

%--------------------------------- - - - - - - - - - - - - - - - - -
% Extensão do meta-predicado conjunção
% verdadeiro e desconhecido => desconhecido
conjuncao(verdadeiro, desconhecido, desconhecido).
conjuncao(desconhecido, verdadeiro, desconhecido).
% verdadeiro e falso => falso
conjuncao(verdadeiro, falso, falso).
conjuncao(falso, verdadeiro, falso).
% falso e desconhecido => falso
conjuncao(falso, desconhecido, falso).
conjuncao(desconhecido, falso, falso).
% algo e algo => algo
conjuncao(verdadeiro, verdadeiro, verdadeiro).
conjuncao(desconhecido, desconhecido, desconhecido).
conjuncao(falso, falso, falso).
```

Figura 34: Conjunção entre duas questões (*demoConj*)

Por fim, o predicado *demoDisj*, bem como o predicado da disjunção:



```
% Extensao do meta-predicado demoDisj (Disjunção entre duas questoes)
demoDisj(Questao1, Questao2, R) :- demo(Questao1, Q1),
                                   demo(Questao2, Q2),
                                   disjuncao(Q1, Q2, R).

%--------------------------------- - - - - - - - - - - - - - - - - -
% Extensão do meta-predicado disjunção
% verdadeiro ou desconhecido => verdadeiro
disjuncao(verdadeiro, desconhecido, verdadeiro).
disjuncao(desconhecido, verdadeiro, verdadeiro).
% verdadeiro ou falso => verdadeiro
disjuncao(verdadeiro, falso, verdadeiro).
disjuncao(falso, verdadeiro, verdadeiro).
% falso ou desconhecido => desconhecido
disjuncao(falso, desconhecido, desconhecido).
disjuncao(desconhecido, falso, desconhecido).
% algo ou algo => algo
disjuncao(verdadeiro, verdadeiro, verdadeiro).
disjuncao(desconhecido, desconhecido, desconhecido).
disjuncao(falso, falso, falso).
```

Figura 35: Disjunção entre duas questões (*demoDisj*)

É de notar que o conhecimento desconhecido tem prioridade sobre o verdadeiro na conjunçaõ de duas questões, enquanto na disjunção é sobre a mentira.

3.3 Funcionalidades Extra

Com base no trabalho desenvolvido, acrescentamos algumas funcionalidades para melhorar a qualidade do projeto. Todas elas foram criadas com o objetivo de estender a base de conhecimento e mostrar a variabilidade de predicados que podem ser construídos para fortalecer a viabilidade e complexidade do nosso sistema.

- **Listagem do Staff de um Centro de Saúde**:

  Apresenta uma lista dos nomes do staff que opera num determinado centro de saúde.

```
% Listagem do staff de um centro de saude [(Staff,Vacinado)] -> {V,F}
listagemStaff(Centro, L) :- solucoes(Nome, staff(_,Centro,Nome,_), Staff),
                            staffVacinado(Staff, [], L).

staffVacinado([],L,L).
staffVacinado([H|T],L,R) :- vacinado(H), staffVacinado(T, [[H,1]|L],R), !.
staffVacinado([H|T],L,R) :- staffVacinado(T, [[H,0]|L],R).
```

Figura 36: Listagem Staff

- **Listagem de Utentes por idade**:

  Coloca uma lista ordenada por idade de forma decrescente de todos os utentes conhecidos.

```
% Listagem da idade dos utentes por ordem decrescente
listagemDecrescente(X) :- solucoes((Nome, DataNascimento),
                          utente(_,_,Nome,DataNascimento,_,_,_,_,_,_), S),
                          ld(S, [], L1),
                          ordenaDec(L1,X).

ld([],L,L).
ld([(N,D)|T],L,R) :- calculaIdade(D,I), ld(T,[(I,N)|L],R).
```

Figura 37: Listagem Utentes por idade

- **Registo das receitas médicas de um utente**:

  A partir do nome de um utente, apresenta o histórico das suas receitas médicas.

```
% Ver o registo das receitas médicas de um utente
registo(Utente, X) :- solucoes(Medicamento, receita(_,_,Utente,_,Medicamento), X).
```

Figura 38: Histórico de receitas médicas de um Utente

- **Tratamentos sem consulta**:

  Lista todos os utentes que efetuaram um determinado tratamento sem que tenha havido um registo de uma consulta prévia.

```
% Extensão do predicado que verifica os utentes que efetuaram um tratamento sem consulta prévia
tratamentoSemConsulta(L) :- solucoes(IdU , tratamento(_,_,IdU,_,_), L2),
                            tscAux(L2, [], L),!.

tscAux([], L, L).
tscAux([Id|T], L, R) :- solucoes(Id, consulta(_,_,Id,_,_), L2),
                        comprimento(L2,N), N == 0,        José Reis, 40 minutes ago • Tudo documen
                        utente(Id,_,Nome,_,_,_,_,_,_,_),
                        tscAux(T, [Nome|L], R).
tscAux([Id|T], L, R) :- solucoes(Id, consulta(_,_,Id,_,_), L2),
                        comprimento(L2,N), N > 0,
                        tscAux(T, L, R).
```

Figura 39: Tratamentos sem Consulta

- **Doentes com uma determinada doença**:

  A partir de uma doença crónica, determina todos os utentes que a possuem.

```
doenca(Doenca, L) :- solucoes(Nome, (utente(_,_,Nome,_,_,_,_,_,Doencas,_), membro(Doenca, Doencas)), L).
```

Figura 40: Doentes com uma determinada doença

4 Conclusões e Sugestões

Um dos maiores desafios do projeto esteve relacionado com a atribuição de critérios de vacinação, sendo que o nosso objetivo era criar um sistema relativamente simples e justo, em simultâneo, para que a ordem em que os utentes fossem vacinados seja organizada conforme o grau de risco, por ordem decrescente. Para além disso, como um dos parâmetros do utente é a data de nascimento, sentimos necessidade de estender um predicado que calculasse a idade de um utente, tendo em base esse parâmetro.

Apesar de ainda sentir algum desconforto com a linguagem em casos particulares (ainda proveniente da adaptação com a programação declarativa) por se tratar de uma linguagem com a qual temos pouca experiência, o grupo teve facilidade em resolver os diversos predicados devido à prática obtida nas aulas.

O conhecimento imperfeito e os seus diferentes tipos revelaram-se enriquecedores do ponto de vista realístico do trabalho, pois existem casos reais de conhecimento incerto, interdito e impreciso. O grupo conseguiu compreender e aplicar estes conceitos de uma forma simples e clara.

No geral, tendo em conta o conjunto de funcionalidades e estratégias desenvolvidas, consideramos que realizamos um trabalho adequado ao que foi pedido. A nossa preocupação, para além das funcionalidades propostas, teve lugar na potencial utilidade do projeto, através da adição de conhecimento e *queries* extra. Também foram desenvolvidos certos invariantes que permitem/impedem a adição/remoção de conhecimento devidamente correto (enquanto o programa é executado), tal como foi demonstrado. Certamente, seria possível, no futuro, melhorar a qualidade do projeto, adicionando mais critérios de vacinação para além dos existentes ou inserindo conhecimento sobre novos predicados, tais como meios de transporte de vacinas ou estagiários de um centro de saúde.