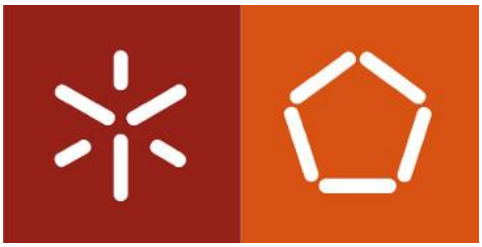
UNIVERSIDADE DO MINHO



Programação em Lógica

Mestrado Integrado em Engenharia Biomédica

Inteligência Artificial em Engenharia Biomédica

(1º Semestre/ Ano Letivo 2021/2022)

Grupo 10

Lara Alexandra Pereira Novo Martins Vaz (A88362)

Mariana Lindo Carvalho (A88360)

Tiago Miguel Parente Novais (A88397)

Braga

30 de dezembro de 2021

Resumo

Este trabalho tem como objetivo desenvolver um sistema de apoio ao diagnóstico médico, terapia e prescrição de tratamentos e/ou medicamentos, utilizando regras de produção para a representação de conhecimento e criação de mecanismos de raciocínio. Para isto utilizou-se a linguagem de programação em lógica PROLOG.

Neste sentido, procedeu-se à estruturação das regras de produção associadas a diagnósticos e respetivos sintomas com um grau de confiança. Por forma a tornar este sistema mais realista, foram também adicionados possíveis tratamentos juntamente com a respetiva medicação. Foi contruído um sistema de inferência com o mecanismo de raciocínio denominado *Backward Chaining*, bem como alguns predicados auxiliares, nomeadamente predicados para a inserção e remoção de factos na base de conhecimento e para o cálculo dos graus de confiança associados ao diagnóstico.

Índice

1. Introdução

Este trabalho tem como objetivo desenvolver um sistema de apoio ao diagnóstico médico, terapia e prescrição de tratamentos e/ou medicamentos, utilizando regras de produção para a representação de conhecimento e criação de mecanismos de raciocínio. Para isto utilizou-se a linguagem de programação em lógica PROLOG.

Com este intuito, a representação do conhecimento intencional e extensional (regras e evidências respetivamente), ocorre associada à definição de ocorrências que tem a si associadas um grau de confiança qualificativo. As evidências dizem respeito aos sintomas apresentados pelos doentes, enquanto que as regras são a aplicação dos sintomas no diagnóstico.

Com a verificação de determinadas condições, os sintomas, pretende-se que seja possível a identificação de um diagnóstico através das referidas regras, bem como o tratamento e a medicação a este associado. O sistema também é capaz de justificar a decisão tomada e calcular o respetivo grau de confiança

Posto isto, as seguintes funcionalidades deveram ser demonstradas ao longo da construção do caso prático apresentado:

- Caraterização de problemas através de conhecimento extensional;
- Representação de conhecimento intencional utilizando regras de produção;
- Representação qualitativa relativa à informação, tendo como perceção da mesma a ocorrência;
- Resolução da problemática associada à identificação e caraterização de problemas resultantes da elaboração de diagnóstico e prescrição;
- Desenvolvimento de sistemas de inferência capazes de implementar mecanismos de raciocínio associados a estes sistemas.

2. Preliminares

2.1. Regras de produção

Um sistema de regras de produção facilita a decisão relativamente a um diagnóstico, analisando a ocorrência de determinados sintomas que conjugados ou por si só podem ter diferentes resultados. Assim, pode haver a possibilidade de concluir vários diagnósticos, uso de medicamentos, respetivas dosagens, terapêuticas entre outros. Além disso, deve existir a capacidade por parte deste sistema de apresentar uma justificação para o resultado que este determina.

Com isto, surgem as regras de produção em lógica que são do tipo:

```
SE <situacao> ENTAO <acao>
SE <condicao> ENTAO <conclusao>
```

Sabe-se que no que toca a regras de produção, uma condição e uma conclusão de uma cláusula, podem ser condição de uma nova regra de base de conhecimento, havendo assim a possibilidade de se obter um diagnóstico.

Quanto às regras do tipo SE-ENTAO, estas determinam as relações lógicas entre conceitos, surgindo, assim, a necessidade de se acrescentar um grau de confiança às regras de produção. Este traduz a probabilidade de ocorrência da conclusão à qual está associado, podendo, assim, ser quantitativo ou qualitativo. Uma possível representação seria:

```
SE <condicao> ENTAO <conclusao> :: <grau de confianca>
```

2.2. *Backward Chaining*

Existem dois métodos capazes de avaliar as regras descritas na base de conhecimento, o *Backward Chaining* e o *Forward Chainig*, sendo a sua escolha determinada pelo objetivo que se pretende atingir [1]. Para este trabalho foi selecionado o *Backward Chaining*.

O *Backward Chaining* consiste num mecanismo regressivo que parte das conclusões obtidas até aos factos, sendo que neste caso estas conclusões são representadas pelo

diagnóstico. Quando este mecanismo é invocado, este percorre a base de conhecimento com o intuito de encontrar um facto ou regra cuja conclusão coincide com a questão associada no momento da invocação. De seguida, é realizada uma verificação em relação à sua veracidade.

3. Descrição do Trabalho e Análise de Resultados

3.1. Declarações Iniciais

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

```
% SWI PROLOG: Declaracoes iniciais
:- set_prolog_flag( discontiguous_warnings,off ).
:- set_prolog_flag( single_var_warnings,off ).
:- set_prolog_flag( unknown,fail ).
```

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

3.2. Definições Iniciais

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

```
% SWI PROLOG: Definições iniciais
:- op( 800,fx,se ).
:- op( 800,fx,facto ).
:- op( 700,xfx,entao ).
:- op( 300,xfy,ou ).
:- op( 200,xfy,e ).
:- op( 900,xfx,porque ).
:- op( 900,xfx,com ).
:- op( 900,xfx,:: ).
:- op( 900,xfx,::: ).
:- dynamic('::'/2).
:- dynamic(':::'/2).
:- dynamic((facto)/1).
```

O operador ('::') está relacionado com as invariantes e os operadores (e), (ou), (com), (se), (entao), (porque), entre outros, estão relacionados com o sistema de inferência desenvolvido no presente trabalho prático.

Relativamente ao predicado que vai representar o conhecimento (facto), este foi definido como dinâmico. Foi também indicado o número de argumentos para cada predicado, sendo 1 para o facto.

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

3.3. Regras de produção

Neste trabalho foram definidas regras de produção, que relacionam os sintomas inseridos com o diagnóstico médico proposto, para a representação do conhecimento intencional. A condição corresponde a um ou vários sintomas, sendo que, neste último caso, podem ser vistos como conjunções e/ou disjunções. A conclusão diz respeito ao diagnóstico proposto, o qual tem associado o grau de confiança, a medicação e respetivo tratamento. Neste mesmo âmbito, foi aplicada a seguinte estrutura nas regras de produção:

```
SE ‘sintoma’ ENTAO ‘diagnostico’ :: ( ‘grau de confianca’, ‘Lista de Medicamentos’, ‘Lista de Tratamentos’).
```

Posto isto, foram criadas as seguintes regras de produção, cujos factos estão definidos no Anexo A. Em baixo estão apresentados alguns exemplos:

```
SE stress('ligeiro') ENTAO 'ansiedade'::('certo','Lorazepam', '2 a 3 comprimidos de 1 mg por dia').

SE 'ansiedade' OU stress('cronico') ENTAO anorexia('mental')::('provavel','Psicoterapia', '2 sessoes por semana').

SE 'nauseas' E 'vomitos' E 'fadiga' E anorexia('ligeira') ENTAO 'nefrite'::('muito provavel','Prednisona','5mg a 60mg por dia').

SE 'nefrite' E 'insonia' E 'hipertensao' ENTAO 'glomerulonefrite'::('provavel', 'Indapamida Eulex', '1 comprimido por dia').

SE 'edema' E 'oliguria' E 'glomerulonefrite' ENTAO 'sindrome nefritica'::('pouco provavel' ,'Aldactone' , '25 mg a 200 mg por dia').
```

A qualidade da informação supramencionada é caracterizada por um grau de confiança associado a cada ocorrência. Estes valores são caraterizados em termos qualitativos e podem tomar qualquer valor dentro do conjunto:

```
{certo, muito provável, provável, pouco provável, improvável}
```

3.4. Sistema de Inferência

De forma a inquirir a base do conhecimento, foi desenvolvido um sistema de inferência capaz de retornar um diagnóstico associado a um grau de confiança, bem como uma lista de medicamentos e uma lista com o respetivo tratamento, específicos de cada situação.

```
% Extensao do meta-predicado rp:Diagnostico,Grau,Explicacao,Lista
de Medicamentos, Lista de Tratamento->{V,F}

rp( D, G, (D com G) porque Jc, [M|LM],[T|LT]):-
            (se C entao D) :: (Gr,M,T),
            rp(C,Gc,Jc,LM,LT),
            prob(Gr,Gc,G).
rp((D1 e D2),G,(J1 e J2),LM,LT):-
            nonvar(D1),
            nonvar(D2),
            rp(D1,G1,J1,LM1,LT1),
            rp(D2,G2,J2,LM2,LT2),
            menor(G1,G2,G),
            concatenar(LM1,LM2,LM),
            concatenar(LT1,LT2,LT).
rp(D1 ou D2,G1,J1, LM1, LT1):-
            nonvar(D1),
            nonvar(D2),
            rp(D1,G1,J1,LM1,LT1),
            nao(rp(D2,G2,J2,LM2,LT2)).
rp(D1 ou D2,G2,J2, LM2, LT2):-
            nonvar(D1),
            nonvar(D2),
            rp(D2,G2,J2,LM2,LT2),
            nao(rp(D1,G1,J1,LM1,LT1)).
rp(D1 ou D2,G,J1 e J2, LM, LT):-
            nonvar(D1),
            nonvar(D2),
```

```
            rp(D1,G1,J1,LM1,LT1),
            rp(D2,G2,J2,LM2,LT2),
            maior(G1,G2,G),
            concatenar(LM1,LM2,LM),
            concatenar(LT1,LT2,LT).
rp(D, G , D com G,[],[]):-
            (facto (D))::G.
```

O mecanismo utilizado foi o de *Backward Chaining*, que parte de determinadas hipóteses e verifica, através de diagnósticos possíveis, se os sintomas apresentados se aplicam aos mesmos.

Este sistema de inferência é composto por cinco argumentos: o Diagnóstico (D) que se pretende verificar, o grau de confiança (G) associado ao diagnóstico, a justificação (J) que demonstra como se deduziu o diagnóstico com base nas regras e factos inseridos, uma lista de medicamentos ([M|LM]) recomendados, juntamente com uma lista os tratamentos associados ([T|LT]).

Na primeira instância, o diagnóstico resulta da conclusão de uma regra, ou conjunto de regras, ou seja, um conjunto de sintomas que resulta num determinado diagnóstico. Note-se que a condição poderá ser uma conclusão de uma outra regra, uma disjunção ou conjunção de sintomas ou de apenas um sintoma. Assim sendo, a explicação do diagnóstico resulta da justificação da condição e o grau de confiança neste caso é obtido pelo predicado *prob* (Secção 3.5.6.). A lista de medicamentos tem como cabeça o medicamento associado à regra e como cauda os medicamentos provenientes das condições. O mesmo ocorre com a lista dos tratamentos.

A instância seguinte representa o caso em que a condição corresponde a uma conjunção de termos, sendo que estes podem ser sintomas ou conclusão de outras regras. É utilizado o predicado *nonvar*, responsável por indicar que as variáveis devem ser instanciadas, evitando assim a ocorrência de um ciclo infinito. O grau de confiança do diagnóstico, que resulta neste caso da conjunção dos graus de confiança de ambos os termos, foi obtido através do predicado *menor (Secção 3.5.6.)*. Relativamente à

justificação da condição, esta é constituída pela explicação de ambos os termos separados pelo operador ‘e’. A lista de medicamentos associada à conjunção resulta da concatenação das listas de medicamentos resultantes dos termos. O mesmo acontece para a lista de tratamentos.

As três instâncias seguintes representam o caso em que a condição corresponde à disjunção de termos: a verificação do primeiro e a não verificação do segundo, a verificação do segundo e a não verificação do primeiro e a verificação de ambos. No caso de apenas se verificar um os termos então o grau de confiança, justificação, lista de medicamentos e lista de tratamentos resultarão do próprio termo. No caso de se verificarem ambos os termos então o processo ocorre de forma semelhante à conjunção, sendo o cálculo do grau de confiança auxiliado pelo predicado *maior* (Secção 3.5.6.).

A última instância representa o caso em que o diagnóstico pertence a um facto da base de conhecimento. Assim, tem-se como explicação o próprio facto, sendo que o grau de confiança corresponde ao grau que se encontra associado a esse mesmo facto. Assim, quer a lista de medicamentos quer a lista de tratamento serão listas vazias.

O funcionamento deste predicado encontra-se demonstrado no Anexo A.

3.5. Predicados Auxiliares

Os predicados desta secção são denominados Predicados Auxiliares, uma vez que a sua função é ser a base de outros predicados necessários ao trabalho proposto.

3.5.1. Meta-predicado *nao*

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

```
% Extensao do meta-predicado nao: QUESTAO -> { V,F }

nao( QUESTAO ) :-

    QUESTAO, !, fail.

nao( QUESTAO ).
```

O meta-predicado *nao* é responsável por implementar a negação por falha na prova e está divido em duas instâncias.

Na primeira instância, o predicado recebe como parâmetro uma *Questao* e, caso seja encontrada uma prova positiva para essa mesma *Questao*, então esta sucede e é aplicado o mecanismo que impede o retrocesso (!). Desta forma, o processo falha, sendo devolvida uma resposta negativa por falha na prova. Caso não seja encontrada uma prova positiva para a *Questao*, então a negação dessa mesma *Questao* vai suceder. Isto é, se um determinado facto existir na base de conhecimento, então não se pode dizer que não existe e é retornado *false*. Por outro lado, se a *Questao* em causa não puder ser provada, então pode-se dizer que a mesma não existe e é retornado *true*.

3.5.2. Predicado *concatenar*

Este predicado retorna duas listas concatenadas. Na primeira instância, uma vez que o primeiro argumento é uma lista vazia, a concatenação entre uma lista vazia e L2 será a própria L2, ou seja, a lista definida como segundo argumento. Relativamente à segunda instância, ambos os argumentos são listas não vazias. Por isso, a concatenação de uma lista com cabeça X e cauda L1 com uma lista L2 sucede numa lista de cabeça X e cauda R, em que, R consiste na concatenação de L1 com L2.

```
% Extensao do predicado concatenar: L1,L2,R -> {V,F}

concatenar( [],L,L ).

concatenar( [X|L1],L2,[X|L3]) :-

      concatenar( L1,L2,L3 ).
```

3.5.3. Predicado *comprimento*

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

recursividade para somar o número de elementos de uma lista só termina quando atinge o critério de paragem.

```
% Extensão do predicado comprimento: Lista, Resultado -> {V,F}

comprimento( [],0 ).

comprimento( [X|L],R ) :-

      comprimento( L,N ),

      R is N+1.
```

3.5.4. Predicado *solucoes*

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

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

solucoes( X,Y,Z ) :-

      findall( X,Y,Z ).
```

3.5.5. Predicados *menor, maior e prob*

Para os casos em que existiam mais do que uma questão, foi necessário construir alguns predicados, tendo estes como objetivo auxiliar o sistema de inferência no cálculo do grau de confiança associado ao diagnóstico.

Posto isto, foram adicionados à base de conhecimento os predicados *menor*, *maior e prob*, os quais são aplicados na conjunção de condições, na disjunção de condições e na aplicação de uma regra, respetivamente.

3.5.6.1. Predicado *menor*

O predicado *menor* recebe como argumentos os graus de confiança associados a duas questões. Como resultado retorna o menor valor entre os graus de confiança. Posto isto, recorreu-se a este predicado para a representação da conjunção(“e”) no predicado *rp*.

```
% Extensão de predicado menor: GrauQ1, GrauQ2, Menor ->{V,F}
menor(X,X,X).
menor('improvavel',X,'improvavel').
menor(X,'improvavel','improvavel').
menor('certo',X,X).
menor(X,'certo',X).
menor('muito provavel','provavel','provavel').
menor('provavel','muito provavel','provavel').
menor('muito provavel','pouco provavel','pouco provavel').
menor('pouco provavel','muito provavel','pouco provavel').
menor('pouco provavel','provavel','pouco provavel').
menor('provavel','pouco provavel','pouco provavel').
```

3.5.6.2. Predicado *maior*

Este predicado recebe também como argumentos os graus de confiança associados a duas questões. No final, o resultado será o maior valor entres esses graus de confiança. O predicado *maior* é utilizado no predicado *rp*, mais precisamente, na disjunção (“ou”).

```
% Extensão de predicado maior: GrauQ1, GrauQ2, Maior ->{V,F}
maior(X,X,X).
maior('certo',X,'certo').
maior(X,'certo','certo').
maior('improvavel',X,X).
maior(X,'improvavel',X).
maior('muito provavel','provavel','muito provavel').
```

```
maior('provavel','muito provavel','muito provavel').
maior('muito provavel','pouco provavel','muito provavel').
maior('pouco provavel','muito provavel','muito provavel').
maior('pouco provavel','provavel','provavel').
maior('provavel','pouco provavel','provavel').
```

3.5.6.3. Predicado *prob*

O predicado *prob* apesar de ser semelhante aos predicados apresentados anteriormente, apresenta um raciocínio diferente. Quando se está perante uma conjunção ou disjunção de condições, para cada questão o predicado *rp* recorre à sua primeira instância, onde é retornado o grau de confiança acordado entre o grau relativo à condição e o grau referente à regra. Se os graus de confiança fossem quantificadores, o resultado da primeira instância seria a multiplicação entre os graus. Contudo, como os graus, neste caso, se tratam de qualificadores e não quantificadores, foi preciso decidir quais os valores obtidos para cada combinação (Tabela 1).

Tabela 1. Resultados das combinações entre argumentos para o predicado *prob*.

Argumentos	Certo	Muito Provável	Provável	Pouco Provável	Improvável
Certo	Certo	Muito Provável	Muito Provável	Provável	Provável
Muito Provável	Muito Provável	Muito Provável	Muito Provável	Provável	Pouco Provável
Provável	Muito Provável	Muito Provável	Provável	Pouco Provável	Pouco Provável
Pouco Provável	Provável	Provável	Pouco Provável	Pouco Provável	Improvável
Improvável	Provável	Pouco Provável	Pouco Provável	Improvável	Improvável

Na base de conhecimento o predicado *prob* apresenta-se do seguinte modo:

```
% Extensão de predicado prob: Gr, Gc, G ->{V,F}
prob('certo','certo','certo').
prob('muito provavel','muito provavel','muito provavel').
prob('provavel','provavel','provavel').
prob('pouco provavel','pouco provavel','pouco provavel').
prob('improvavel','improvavel','improvavel').
prob('provavel','certo','muito provavel').
prob('certo','provavel','muito provavel').
prob('certo','improvavel','provavel').
prob('improvavel','certo','provavel').
prob('muito provavel','certo','muito provavel').
prob('certo','muito provavel','muito provavel').
prob('certo','pouco provavel','provavel').
prob('pouco provavel','certo','provavel').
prob('muito provavel','provavel','muito provavel').
prob('provavel','muito provavel','muito provavel').
prob('muito provavel','improvavel','pouco provavel').
prob('improvavel','muito provavel','pouco provavel').
prob('muito provavel','pouco provavel','provavel').
prob('pouco provavel','muito provavel','provavel').
prob('pouco provavel','provavel','pouco provavel').
prob('provavel','pouco provavel','pouco provavel').
prob('provavel','improvavel','pouco provavel').
prob('improvavel','provavel','pouco provavel').
prob('improvavel','pouco provavel','improvavel').
prob('pouco provavel','improvavel','improvavel').
```

3.6. Evolução de Conhecimento

Para que o responsável pelo diagnóstico possa inserir na base de conhecimento os sintomas que um determinado paciente apresente de forma a determinar o diagnóstico e tratamento, criou-se o predicado *evolucao_factos*. Este predicado possibilita a inserção

de um sintoma e aceita dois parâmetros, o facto a inserir e o seu grau de confiança qualitativo que lhe está associado.

```
% Extensão do predicado evolucao_factos: Termo, Certeza -> {V,F}
evolucao_factos(Termo, Certeza) :-
      solucoes(Invariante,+(facto(Termo)::Certeza):::Invariante,L
ista),
      insercao(Termo, Certeza),
      grau(Certeza),
      teste(Lista).
```

A partir da análise do predicado *evolucao_factos*, verifica-se que este predicado procura todos os invariantes que dizem respeito ao facto que se pretende inserir com o grau em concordância, guardando estes numa lista. Com o auxílio de outros três predicados, é possível fazer a inserção deste novo conhecimento. Através do predicado *insercao*, abaixo representado, o facto é inserido na base do conhecimento e através do predicado *teste* confirma-se se esta inserção pode realmente acontecer. No caso de se tentar inserir conhecimento repetido, o predicado *insercao* não vai inserir nada na base de conhecimento, sendo invocada a segunda instância deste predicado. Neste sentido remove-se o facto que tinha sido inserido, mas não respeita os invariantes existentes para validar a base do conhecimento. Por último, através do predicado *grau*, confirma-se que o grau de confiança que se quer inserir existe e pode ser inserido.

No Anexo A, encontra-se exemplificado o funcionamento do predicado *evolucao_factos.*

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

```
% Extensão do predicado teste: Lista -> {V,F}

teste([]).

teste([I|L]) :-

    I, teste(L).

% Extensão do predicado grau: Grau -> {V, F}

grau('improvavel').

grau('pouco provavel').

grau('provavel').

grau('muito provavel').

grau('certo').
```

3.7. Involução de Conhecimento

De forma a ser possível remover conhecimento da base de conhecimento, foi definido o predicado *involucao_factos*. A funcionalidade deste predicado obedece a algumas condições, ou seja, depende dos invariantes, do predicado *remover* e do predicado *teste*.

Este predicado começa com a procura de todos os invariantes associados a um determinado termo T, termo este que corresponde ao facto que se pretende remover. O predicado *remover* é auxiliado pelas funcionalidades do *retract*. Após esta ação, o predicado *teste* é invocado de modo a verificar se cada um dos invariantes é válido ou não. Se todos os invariantes forem válidos, este processo termina e o facto foi certamente removido.

Caso algum dos invariantes não seja válido, retrocede-se à seguinte invocação válida, isto é, invoca-se a segunda instância do predicado *remover*, onde este irá inserir (através das funções do *assert*) de novo o facto que se tinha previamente removido.

No Anexo A, encontra-se exemplificado o funcionamento do predicado *involucao_factos.*

```
% Extensão do predicado involucao_factos: ID, Termo, Certeza ->
{V, F}
involucao_factos(ID,Termo,Certeza) :-
      solucoes(Invariante,
      -(facto (ID,Termo)::Certeza):::Invariante,Lista),
      remover(ID, Termo,Certeza),grau(Certeza),
    teste( Lista ).

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

3.8. Invariantes

Para que a base de conhecimento possa ser manipulada, mas, ao mesmo tempo mantenha a sua consistência é necessária a utilização de invariantes.

3.8.1. Invariante de Inserção

A inserção de conhecimento deve respeitar algumas regras de acordo com a sua real aplicação. Tendo por base a realidade a que o projeto foi submetido, foi criado um invariante que não permite introduzir factos repetidos na base de conhecimento. Para isso foi criado o invariante abaixo apresentado.

```
%Invariante que não permite a inserção de conhecimento repetido
na base de conhecimento.
+(facto(Termo) :: Certeza) ::: (solucoes((Termo), (facto(Termo)
:: Certeza), S), comprimento(S,N), N==1).
```

3.8.2. Invariante de Remoção

Para a remoção de conhecimento, foi criado um invariante de remoção para permitir remover um facto, apenas quando este existe na base de conhecimento.

```
%Invariante que permite remover um facto caso este exista na base de conhecimento.
-(facto(Termo) :: Certeza) ::: (solucoes((Termo), (facto(Termo) :: Certeza), S), comprimento(S,N), N==0).
```

4. Conclusão

Os sistemas de apoio à decisão e diagnóstico médico são sistemas que hoje em dia se começam a tornar essenciais, dado os avanços tecnológicos e a evolução da Inteligência Artificial. Um sistema de produção ou de regras de produção é importante no apoio à decisão, sendo considerado uma forma de Inteligência Artificial. É possível considerar que ao se representar o conhecimento através de regras de produção é possível uma aproximação ao mundo real.

O projeto desenvolvido teve como objetivo a criação de um protótipo de um sistema de apoio à decisão, ou seja, de um sistema capaz de auxiliar um médico na decisão de um diagnóstico. Este sistema permitiu efetuar diagnósticos tendo por base os sintomas apresentados, permitindo ainda receitar medicamentos e definir tratamentos.

Neste sentido, foi idealizado um sistema de inferência que potencia a obtenção dos diversos diagnósticos, bem como o grau de confiança que estes apresentam, o tratamento necessário e respetiva medicação.

Concluiu-se que este tipo de representação tem como vantagens a naturalidade, no sentido em que o médico pode expressar o conhecimento através de meras regras condição-ação e a uniformidade da representação, o que facilita a compreensão do conhecimento

Futuramente, num ambiente clínico real, este tipo de sistemas seria muito vantajoso tanto para os utentes como para os profissionais de saúde.

Bibliografia

[1] Bratko, I. *Prolog programming for artificial intelligence.* Pearson education, 2001.

[2] Russell, S. and Norvig, P. *Artificial intelligence*.

Anexo A

Abaixo estão representados os factos utilizados nas regras de produção:

```
facto stress('ligeiro') :: ('provavel').

facto ('ansiedade') :: ('provavel').

facto stress('cronico') :: ('provavel').

facto ('nauseas') :: ('pouco provavel').

facto ('vomitos') :: ('muito provavel').

facto ('fadiga') :: ('certo').

facto anorexia('ligeira') :: ('provavel').

facto ('insonia') :: ('provavel').

facto ('hipertensao') :: ('muito provavel').

facto ('edema') :: ('pouco provavel').

facto ('oliguria') :: ('certo').
```

De seguida, está exemplificado na Figura 1 o funcionamento do sistema de inferência *rp*, tendo como base as regras de produção criadas anteriormente.

```
?- rp(ansiedade,G,Exp,LM,LT).
G = 'muito provavel',
Exp =  ((ansiedade com 'muito provavel')porque(stress(ligeiro)com provavel)),
LM = ['Lorazepam'],
LT = ['2 a 3 comprimidos de 1 mg por dia'] .

?- rp(anorexia(mental),G,Exp,LM,LT).
G = 'muito provavel',
Exp =  ((anorexia(mental)com'muito provavel')porque((ansiedade com 'muito provavel')porque(stress(ligeiro)com provavel
))e(stress(cronico)com provavel)),
LM = ['Psicoterapia', 'Lorazepam'],
LT = ['2 sessoes por semana', '2 a 3 comprimidos de 1 mg por dia'] .

?- rp(nefrite,G,Exp,LM,LT).
G = provavel,
Exp =  ((nefrite com provavel)porque(nauseas com 'pouco provavel')e(vomitos com 'muito provavel')e(fadiga com certo)e(
anorexia(ligeira)com provavel)),
LM = ['Prednisona'],
LT = ['5mg a 60mg por dia'] .

?- rp(glomerulonefrite,G,Exp,LM,LT).
G = provavel,
Exp =  ((glomerulonefrite com provavel)porque((nefrite com provavel)porque(nauseas com 'pouco provavel')e(vomitos com
'muito provavel')e(fadiga com certo)e(anorexia(ligeira)com provavel))e(insonia com provavel)e(hipertensao com 'muito p
rovavel')),
LM = ['Indapamida Eulex', 'Prednisona'],
LT = ['1 comprimido por dia', '5mg a 60mg por dia'] .

?- rp('sindrome nefritica',G,Exp,LM,LT).
G = 'pouco provavel',
Exp =  (('sindrome nefritica'com'pouco provavel')porque(edema com 'pouco provavel')e(oliguria com certo)e((glomerulone
frite com provavel)porque((nefrite com provavel)porque(nauseas com 'pouco provavel')e(vomitos com 'muito provavel')e(.
..com...)e(...com...))e(insonia com provavel)e(hipertensao com 'muito provavel'))),
LM = ['Aldactone', 'Indapamida Eulex', 'Prednisona'],
LT = ['25 mg a 200 mg por dia', '1 comprimido por dia', '5mg a 60mg por dia'] .
```

Figura 1. Funcionamento do predicado *rp* (sistema de inferência).

Na Figura 2, é possível se verificar um exemplo do correto funcionamento dos predicados *evolucao_factos* e *involucao_factos*, cujo objetivo já referido, é permitir adicionar e remover conhecimento da base de conhecimento, respetivamente.

```
?- involucao_factos(nauseas,'pouco provavel').
true .

?- rp(nefrite,G,Exp,LM,LT).
false.

?- evolucao_factos(nauseas,'pouco provavel').
true .

?- rp(nefrite,G,Exp,LM,LT).
G = provavel,
Exp =  ((nefrite com provavel)porque(nauseas com 'pouco provavel')e(vomitos com 'muito prov
avel')e(fadiga com certo)e(anorexia(ligeira)com provavel)),
LM = ['Prednisona'],
LT = ['5mg a 60mg por dia'] .

?- evolucao_factos(nauseas,'pouco provavel').
false.

?- involucao_factos(febre,'provavel').
false.
```

Figura 2. Exemplificação do funcionamento dos predicados *evolucao_factos* e *involucao_factos*.