Universidade do Minho
Escola de Engenharia

MESTRADO INTEGRADO EM ENGENHARIA BIOMÉDICA

INFORMÁTICA MÉDICA 2020/2021

Programação em Lógica, Conhecimento e Raciocínio – Parte 1

Alunos:

João Miguel da Silva Alves (83624)

Paulo Jorge Alves (84480)

Docente:

César Analide

13 de novembro 2020

RESUMO

Com base na criação de uma base de dados de uma instituição, este trabalho tem como principal objetivo o estudo da linguagem de programação em lógica, Prolog, e na sua aplicação para representação de conhecimento.

Esta base de dados é caracterizada tendo como conhecimento os utentes de uma certa instituição, os serviços prestados, as consultas realizadas e os prestadores de saúde.

A construção deste sistema baseia-se nos predicados lógicos (factos, perguntas e regras) e na manipulação desta base de conhecimentos através do uso de invariantes.

Para caracterizar e comprovar que o sistema criado cumpre os requisitos estabelecidos recorreu-se à ferramenta SICStus.

ÍNDICE

1. INTRODUÇÃO

A fim de desenvolver um sistema de representação de conhecimento e raciocínio com capacidade para caracterizar um universo de discurso na área da prestação de cuidados de saúde, este trabalho tem como objetivo a utilização da linguagem em lógica PROLOG no âmbito da representação de conhecimento.

Para este efeito, foi construída uma base de conhecimento que permitisse responder a alguns pontos essenciais propostos:

- Registar utentes, serviços e consultas;
- Remover utentes, serviços e consultas;
- Identificar as instituições prestadoras de serviços;
- Identificar utentes/serviços/consultas por critérios de seleção;
- Identificar serviços prestados por instituição/cidade/datas/custo;
- Identificar os utentes de um serviço/instituição;
- Identificar serviços realizados por utente/instituição/cidade;
- Calcular o custo total dos cuidados de saúde por utente/serviço/instituição/data.

2. PRELIMINARES

Programação em Lógica não é sinónimo de Prolog. Prolog é uma linguagem de programação de uso geral que é especialmente associada com a inteligência artificial e linguística computacional.[2]

O Prolog é uma linguagem declarativa, ou seja, ao contrário de o programa estipular a maneira de chegar à solução, como acontece nas linguagens orientadas por objetos, ele fornece uma descrição do problema que se pretende computar. Assim sendo, é uma linguagem mais direcionada ao conhecimento do que aos próprios algoritmos.

Um programa em Prolog é uma coleção de unidades lógicas chamadas de predicados. Um predicado representa uma relação lógica (verdadeiro ou falso) entre dois indivíduos, e cada predicado é uma coleção de cláusulas (factos e regras). De notar que, num programa Prolog, os predicados, os objetos e as variáveis são representados por letras minúsculas e as constantes por letras maiúsculas. Todas as instruções são finalizadas com um ponto final. [4][5]

2.1. Factos

Os factos são algo que é sempre verdadeiro ou, por outras palavras, são verdades incondicionais e permitem que os objetos estabeleçam relações entre si num determinado contexto do problema.[5] Por exemplo:

$$utente(100, margaridaalves, 19, braga, manuelalves, joanaalves, 123456789, asma).$$

O facto acima descrito, estabelece que a Margarida Alves é a utente nº100, tem 19 anos, é de Braga, o nome do pai é Manuel Alves, o nome da mãe é Joana Alves, o NIF é 123456789 e tem asma. Ou seja, existe a relação utente entre todos os objetos descritos.

2.2. Perguntas

Quando uma pergunta é feita, num interpretador de Prolog, o Prolog realiza uma busca na sua base de conhecimento, procurando um facto que seja igual ao da pergunta. Se o facto for encontrado é retornado "YES", caso contrário o programa retornará "NO".[5] Por exemplo:

$$| \quad ? - utente(100, margaridaalves, 19, braga, manuelalves, joanaalves, 123456789, asma).$$
$$yes$$

$| \quad ? - utente(100, margaridaalves, 19, porto, manuelalves, joanaalves, 123456789, asma).$
no

No primeiro caso, como a resposta à questão é "*yes*", verifica-se que existe na base de conhecimento uma utente nº100 com todos os seus atributos. Já no segundo caso, a resposta à pergunta "Existe na base de conhecimento uma utente nº100 com os restantes atributos, mas que seja do Porto?" é "*no*", o que significa que não existe uma utente nº100 que é do Porto na base de conhecimento.

Uma outra maneira de realizar perguntas no interpretador de Prolog, é com o auxílio de variáveis. Por exemplo:

$| \quad ? - utente(100, margaridaalves, 19, X, manuelalves, joanaalves, 123456789, asma).$
$X = braga$

Neste caso, a pergunta feita é "Se existir uma utente nº100 na base de conhecimento, então qual a localidade X a que pertence?".

2.3. Regras

As regras permitem definir novas relações em termos de outras relações já existentes, ou seja, especificam-se situações que podem ser verdadeiras se algumas condições forem satisfeitas. Para utilizarmos uma regra, usamos o símbolo ":-" que indica uma condição "se" .[1]

Por exemplo, dado o facto e a regra, respetivamente:

$$utente(100, margaridaalves, 19, braga, manuelalves, joanaalves, 123456789, asma).$$

$$mae(F, M) :- utente(Idutente, F, Idade, Cidade, Pai, M, NIF, Doenca).$$

A regra acima pode ser traduzida por: M é mãe de F se no predicado utente F corresponder ao nome de um utente e M corresponder ao nome da mãe desse mesmo utente.

Num interpretador de Prolog, por exemplo:

$| \quad ? - mae(margaridaalves, joanaalves).$
yes

A resposta à pergunta acima é "yes", uma vez que a Joana Alves é mãe da Margarida Alves.

3. CONSTRUÇÃO DA BASE DE DADOS LÓGICA

3.1. Declarações iniciais

De modo a modelar o programa ao contexto é necessário a implementação de *flags* na base de conhecimento [3], através de:

$:-set\_prolog\_flag(unknown, fail).$

$:-set\_prolog\_flag(discontiguous\_warnings, off).$

$:-set\_prolog\_flag(single\_var\_warnings, off).$

$:-op(900, xfy, ´::´).$

3.2. Definições iniciais

As declarações *dynamic* são utilizadas para predicados que se pretendem que sejam dinâmicos uma vez que, estes, quando são gerados, tomam o valor de estáticos, o que não permite ao programa alterar esses mesmos predicados. Desta forma, a implementação destas definições iniciais surge para permitir que haja inserção ou remoção de conhecimento dos predicados utente, prestador, consulta e serviço. [3]

$:-dynamic\ utente/8.$

$:-dynamic\ prestador/4.$

$:-dynamic\ consulta/6.$

$:-dynamic\ servico/5.$

3.3. Factos da Base de Dados

A primeira parte do trabalho consistiu na construção da base de conhecimento, sendo que esta é resultado da declaração de utentes, serviços, consultas e prestadores. Em termos práticos, a inserção dos factos e consequente construção da base de conhecimento foi possível devido à implementação destes quatro predicados:

3.3.1. Predicado utente

$$utente(100, margaridaalves, 19, braga, manuelalves, joanaalves, 123456789, asma).$$

O predicado acima é um exemplo do predicado "utente" usado na base de conhecimento construída. Tem como atributos o id utente, o nome, a idade, localidade, nome do pai, nome da mãe, NIF e por fim a doença. Todos os predicados "utente" usados na base de conhecimento encontram-se em anexo.

3.3.2. Predicado servico

$$servico(200, 1005, oscultar, hospitaldatrofa, trofa).$$

O predicado acima é um exemplo do predicado "servico" usado na base de conhecimento construída. Tem como atributos o id do serviço, o id do prestador que o realiza, a descrição, a instituição onde o serviço é prestado e por fim a respetiva cidade. Todos os predicados "servico" usados na base de conhecimento encontram-se em anexo.

3.3.3. Predicado consulta

$$consulta(1, 06112020, 100, 200, 1005, 80).$$

O predicado acima é um exemplo do predicado "consulta" usado na base de conhecimento construída. Tem como atributos o id da consulta, a data de realização, o id do utente que realiza a consulta, o id do serviço que é prestado nessa consulta, o id do prestador responsável e por último o custo dessa mesma consulta. Todos os predicados "consulta" usados na base de conhecimento encontram-se em anexo.

3.3.4. Predicado prestador

$$prestador(1000, joaovasquez, medicinageral, hospitalpublicodebraga).$$

O predicado acima é um exemplo do predicado "prestador" usado na base de conhecimento construída. Tem como atributos o id prestador, o nome, a especialidade e por último a instituição onde exerce. Todos os predicados "prestador" usados na base de conhecimento encontram-se em anexo.

3.4. Predicados Auxiliares

3.4.1. Predicado soluções:

Este predicado é utilizado quando se pretende listar todas as soluções como resposta a uma dada questão.

$\%extensão\ do\ predicado\ soluções: Elemento, Lista, Lista\_com\_ocorrências -> \{V, F\}$

$solucoes(X, Y, L) :- findall(X, Y, L).$

O predicado acima contém 3 argumentos: X, que corresponde ao elemento que se pretende obter; Y que é o teorema que se pretende provar quando é feita uma certa pergunta; L que é a lista que contém o conjunto de soluções.

A ferramenta "*findall*" já se encontra no SICStus.

3.4.2. Predicado pertence:

Este predicado tem como objetivo verificar se um elemento X existe numa lista L.

$\%extensao\ do\ predicado\ pertence: Elemento, Lista -> \{V, F\}$

$pertence(X, [X|L]).$

$pertence(X, [Y|L]) :- pertence(X, L).$

O predicado acima está dividido em 2 partes. A primeira parte verifica se um elemento X está na cabeça da lista. A segunda parte verifica se X está na cauda da lista. Se X estiver na cabeça ou na cauda da lista, então X pertence à lista L.

3.4.3. Predicado comprimento:

Este predicado recebe como argumento uma lista e determina o seu comprimento.

$\%extensão\ do\ predicado\ comprimento: Lista, Resultado -> \{V, F\}$

$comprimento([X], 1).$

$comprimento((X|L), R) :- comprimento(L, R1), R\ is\ R1 + 1.$

O comprimento de uma lista com apenas um elemento é 1. Pela recursividade, tendo a lista, X como cabeça e L como cauda, e se R1 é o comprimento de L, então a lista tem comprimento R1+1.

3.4.4. Predicado apagarelemento:

Este predicado serve para eliminar todas as ocorrências de um elemento X numa lista.

$$\%extensão\ do\ predicado\ apagarelemento: Elemento, lista, lista\_sem\_elemento -> \{V, F\}$$

$$apagarelemento(X, [], []).$$

$$apagarelemento(X, [X|L], L).$$

$$apagarelemento(X, [N|L], [N|Y]) :- X\backslash== N, apagarelemento(X, L, Y).$$

O predicado acima descrito divise-se em 3 partes. Uma primeira no caso de a lista ser vazia, onde o predicado retorna uma lista vazia. Uma segunda parte que verifica se o elemento X está na cabeça da lista. Se estiver, então é retornada uma lista correspondente à cauda da lista do segundo argumento. Por fim, uma terceira parte, onde a cabeça não é X, que utiliza um processo recursivo para percorrer a lista e, desta forma, procurar o elemento a eliminar, retornando a lista sem esse elemento.

3.4.5. Predicado apagarrepetidos:

Este predicado serve para eliminar resultados que estejam repetidos numa lista, apresentando a lista sem elementos repetidos.

$$\%extensão\ do\ predicado\ apagarrepetidos: lista, lista\_sem\_elemento -> \{V, F\}$$

$$apagarrepetidos([], []).$$

$$apagarrepetidos([X|Y], [X|L]) :- apagarelemento(X, Y, NT), apagarrepetidos(NT, L).$$

O predicado acima descrito é composto por 2 partes. Se a lista for vazia, então é retornada uma lista vazia. Se a lista não for vazia, então irá com o recurso ao predicado "apagarelemento", eliminar qualquer repetido na cauda do elemento que se encontra na cabeça. Depois, por um processo recursivo irá fazer o mesmo para qualquer outro elemento que se encontre repetido, retornando, assim, uma lista sem repetidos.

3.4.6. Predicado soma:

Este predicado tem como objetivo somar todos os elementos de uma lista L.

$\%extensão\ do\ predicado\ soma: Lista, resultado -> \{V, F\}$

$soma([X], X).$

$soma([N|L], R) :- soma(L, R1), R\ is\ N + R1.$

O predicado acima é composto por 2 partes. Se a lista é composta apenas por um elemento, então o predicado retorna esse mesmo elemento. Caso contrário, pela recursividade, uma lista com o elemento N na cabeça e a lista L na cauda, se a soma dos elementos de L é R1, então a soma dos elementos da lista passada como argumento é R1+1.

3.4.7. Predicado não:

Este predicado tem como objetivo retornar o valor falso caso o valor de verdade da questão seja verdadeiro ou retornar o valor verdadeiro caso o valor de verdade da questão seja falso.

$\%extensão\ do\ predicado\ não: Questão \rightarrow \{V, F\}$

$nao\ (Questao) :- Questao, !, fail.$

$nao(Questao) :- Questao.$

O predicado acima descrito recebe como argumento uma questão. Se o valor da questão for verdadeiro, então através da implementação do cut e fail que significa insucesso, este não procurará a segunda cláusula e irá retornar falso. Caso o valor da questão seja falso, então passará para a próxima cláusula e , por isso, irá retornar verdadeiro.

4. EXTENSÃO DOS PREDICADOS

4.1. Extensão do predicado atender

Esta extensão está relacionada com os predicados utente e prestador, em que os seus atributos são o ID do utente, o ID do prestador, a especialidade desse prestador e a instituição, no qual este exerce o seu trabalho.

$$\%extensão\ do\ predicado\ atender: Nome, Nomeprestador, especialidade, Instituição -> \{V, F\}$$

$$atender(U, P, E, I) :- utente(U, Nome, Idade, Pai, Mãe, NIF, Doenca),$$
$$prestador(P, Nomep, E, I).$$

O predicado atender só existe se e só se existir um utente com o id passado como argumento, tal como o id do prestador, a sua especialidade e instituição.

4.2. Extensão do predicado pai e do predicado mãe

A extensão destes predicados recebe os atributos: filho e pai, ou filho e mãe, respetivamente. Portanto, um utente será pai de outro utente se e só se houver no predicado utente, alguém com um pai associado. O mesmo acontece com o predicado mãe, contudo, se houver uma mãe associada.

$$\%extensão\ do\ predicado\ pai: Filho, pai -> \{V, F\}$$

$$pai(F, P) :- utente(Idutente, F, Idade, Cidade, P, Mae, NIF, Doenca).$$

$$\%extensão\ do\ predicado\ mãe: Filho, mãe -> \{V, F\}$$

$$mae(F, M) :- utente(Idutente, F, Idade, Cidade, Pai, M, NIF, Doenca).$$

4.3. Evolução

A extensão deste predicado recebe como atributo um termo, seja correspondente ao predicado utente, prestador, serviço ou consulta. De seguida com o predicado soluções, iremos obter uma lista com todos os invariantes correspondentes a esse termo. Depois através da primeira cláusula do predicado inserir, irá inserir o termo na nossa base de conhecimento. Com o predicado teste, iremos testar a consistência da nossa base de conhecimento. Este irá testar

todos os invariantes associados a esse termo. Se tudo estiver correto, então o termo é inserido. Caso contrário, irá se passar para a segunda cláusula do predicado inserir em que se irá remover esse termo antes inserido e não voltará atrás para testar, uma vez que se encontra o cut e o fail que significa insucesso.

$\%extensão\ do\ predicado\ evolução: Termo \rightarrow \{V, F\}$

$evolução(Termo) :- \ solucoes(Inv, +Termo :: Inv, Linv),$
$inserir(Termo), teste(Linv).$

$inserir(Termo) :- assert(Termo).$
$inserir(Termo) :- retract, !, fail.$

$teste([\ ]).$
$teste([I|L) :- I, teste(L)$

4.4. Involução

A extensão do predicado involução recebe como atributo um termo tanto do predicado utente, prestador, serviço ou consulta. De seguida, através do predicado soluções, iremos obter uma lista com todos os invariantes correspondentes a esse termo. Depois através do predicado testar, irá testar se vai ser possível remover o nosso termo da base de conhecimento. Este irá testar todos os invariantes de remoção do respetivo termo. Caso derem todos corretos, então passará para o predicado remover que irá remover o termo passado como argumento. Caso contrário, o predicado remover não ocorrerá e o termo não será removido.

$\%extensão\ do\ predicado\ involução: Termo \rightarrow \{V, F\}$

$involucao(Termo) :- solucoes(Inv, -Termo :: Inv, Linv),$
$testar(Linv), remover(Termo).$

$remover(Termo) :- retract(Termo).$
$remover(Termo) :- assert(termo), !, fail.$

$testar([]).$
$testar([I|L]) :- I, testar(L).$

4.5. Extensão do predicado que identifica as instituições prestadoras de serviço

Na extensão deste predicado, o qual recebe como atributo uma instituição e verifica se esta presta ou não serviços. Para tal, utiliza o predicado soluções que procura e coloca numa lista todas as instituições que prestam serviços e, de seguida, com o auxílio do predicado apagarrepetidos, elimina possíveis instituições em repetição. No final, verifica se a instituição passada como argumento, pertence ou não à lista.

$\%extensão\ do\ predicado\ que\ identifica\ as\ inst.prestadoras\ de\ serviços: Instituição, Resultado -> \{V, F\}$

$$idtinstituicoes(X) :- solucoes(I, servico(A, P, D, I, C), L),$$
$$apagarrepetidos(L, R), pertence(X, R).$$

4.6. Extensão do predicado que mostra todas as instituições prestadoras de um determinado serviço

Neste caso, o predicado recebe como argumentos, um serviço e todas as instituições que, de facto, prestam esse serviço. Depois, com o recurso ao predicado apagarrepetidos, elimina eventuais instituições que se encontram repetidas na lista.

$\%extensao\ do\ predicado\ que\ mostra\ todas\ as\ instituições\ prestadoras\ de\ serviços: servico, Resultado -> \{V, F\}$

$$idtinstituicoes2(S, R) :- solucoes(H, servico(A, B, S, H, C), Y), apagarrepetidos(Y, R).$$

4.7. Extensão do predicado que identifica utentes por idade

Com esta extensão, a qual recebe como argumentos uma idade e uma lista com todos os utentes com essa idade. Par tal recorre ao predicado soluções para encontrar o nome dos utentes que têm a idade passada como argumento e mostra uma lista contendo esses nomes.

$\%extensão\ do\ predicado\ que\ identifica\ utentes\ por\ critérios\ de\ seleção: idade, resultado -> \{V, F\}$

$$idtutente(I, R) :- solucoes(U, utente(A, U, I, C, P, M, N, D), R).$$

4.8. Extensão do predicado que identifica utentes por cidade

Nesta extensão o mesmo acontece relativamente ao anterior, no entanto, recebe como argumentos uma cidade e uma lista contendo todos os nomes dos utentes que vivem nessa

cidade. É utilizado o predicado soluções para retornar uma lista com todos os nomes dos utentes que vivem na cidade passada como argumento.

$\%extensao\ do\ predicado\ que\ identifica\ utentes\ por\ criterios\ de\ selecao\colon cidade, resultado -> \{V, F\}$

$idtutente2(C, R) :- solucoes(U, utente(A, U, I, C, P, M, N, D), R).$

4.9. Extensão do predicado que identifica utentes por doença

Este caso é em todo semelhante aos anteriores, contudo, recebe como argumentos uma doença e uma lista contendo todos os nomes dos utentes com essa doença. Mais uma vez, recorre-se ao predicado soluções que retorna a lista que possui os nomes de todos os utentes com a doença passada como argumento.

$\%extensao\ do\ predicado\ que\ identifica\ utentes\ por\ criterios\ de\ selecao\colon doenca, resultado\ > \{V, F\}$

$idtutente3(D, R) :- solucoes(U, utente(A, U, I, C, P, M, N, D), R).$

4.10. Extensão do predicado que identifica serviços por ID de prestador

Com esta extensão, a qual recebe como argumentos um ID prestador e uma lista contendo todos os serviços prestados por esse ID. Recorre-se ao predicado soluções que retorna uma lista que mostra todos os serviços prestados pelo ID prestador passado como argumento.

$\%extensao\ do\ predicado\ que\ identifica\ serviços\ por\ criterios\ de\ selecao\colon idprestador, resultado -> \{V, F\}$

$idtservico(X, R) :- solucoes(D, servico(A, X, D, I, C), R).$

4.11. Extensão do predicado que identifica serviços por instituição

Com esta extensão, que recebe como argumentos uma instituição e uma lista contendo todos os serviços prestados nessa instituição. Recorrendo ao predicado soluções, este retornará uma lista que mostra todos os serviços prestados por uma instituição que é passada como argumento.

$\%extensao\ do\ predicado\ que\ identifica\ serviços\ por\ criterios\ de\ selecao\colon Instituição, resultado -> \{V, F\}$

$idtservico2(X, R) :- solucoes(D, servico(A, P, D, X, C), R).$

4.12. Extensão do predicado que identifica serviços por cidade

Nesta extensão, a qual recebe como argumentos uma cidade e uma lista contendo todos os serviços que são prestados numa cidade. Recorrendo ao predicado soluções, este irá retornar uma lista com todos os serviços prestados numa cidade que é passada como argumento.

$\%extensao\ do\ predicado\ que\ identifica\ serviços\ por\ criterios\ de\ selecao\colon cidade, resultado -> \{V, F\}$

$idtservico3(X, R) :- solucoes(D, servico(A, P, D, I, X), R).$

4.13. Extensão do predicado que identifica consultas por data

Esta extensão recebe como argumentos uma data e uma lista que contém todos os id de consultas que são feitas nessa data. Com o auxílio do predicado soluções, este retornará uma lista com todas as consultas ocorridas na data que é passada como argumento.

$\%extensao\ do\ predicado\ que\ identifica\ consultas\ por\ criterios\ de\ selecao\colon data, resultado -> \{V, F\}$

$idtconsulta(X, R) \colon -solucoes(A, consulta(A, X, U, S, P, C), R).$

4.14. Extensão do predicado que identifica consultas por id de utente

Esta extensão recebe como argumentos um id de utente e uma lista de todos os ids das consultas desse id utente. Recorre-se ao predicado soluções, o qual retornará uma lista de todas as consultas do id utente passado como argumento.

$\%extensao\ do\ predicado\ que\ identifica\ consultas\ por\ criterios\ de\ selecao\colon utente, resultado -> \{V, F\}$

$idtconsulta2(X, R) \colon -solucoes(A, consulta(A, D, X, S, P, C), R).$

4.15. Extensão do predicado que identifica consultas por id serviço

Neste caso, este predicado recebe como argumentos um id serviço e uma lista de todos os ids de consultas feitas com esse id serviço. Para tal, recorre-se ao predicado soluções que retorna uma lista com todos os ids de consultas feitas com o id serviço passado como argumento.

$\%extensao\ do\ predicado\ que\ identifica\ consultas\ por\ criterios\ de\ selecao\colon serviço, resultado -> \{V, F\}$

$idtconsulta3(X, R) \colon -solucoes(A, consulta(A, D, U, X, P, C), R).$

4.16. Extensão do predicado que identifica consultas por id prestador

Esta extensão recebe como argumentos um id prestador e uma lista contendo todas as consultas feitas por esse prestador. Recorre-se, então ao predicado soluções, que retornará uma lista com todos os ids das consultas prestadas pelo id prestador passado como argumento.

$\%extensao\ do\ predicado\ que\ identifica\ consultas\ por\ criterios\ de\ selecao: prestador, resultado -> \{V, F\}$

$idtconsulta4(X, R) :- solucoes(A, consulta(A, D, U, S, X, C), R).$

4.17. Extensão do predicado que identifica consultas por custo

Para esta extensão, os argumentos são o custo e uma lista contendo todas os ids consultas com esse custo. Mais uma vez, recorre-se ao predicado soluções que mostra a lista de todos os ids consultas do custo passado como argumento.

%extensao do predicado que identifica consultas por criterios de selecao: custo, resultado $-> \{V, F\}$

$idtconsulta5(X, R) :- solucoes(A, consulta(A, D, U, S, P, X), R).$

4.18. Extensão do predicado que identifica o nome dos serviços prestados por data

Esta extensão recebe como argumentos uma data e uma lista contendo o nome dos serviços prestados nessa data, sem repetidos. O predicado soluções é invocado para a partir dos predicados consultas que contém o id serviço e a data passada como argumento e o predicado serviço que a partir desse id irá buscar o nome do serviço, retornar uma lista com esses nomes. De seguida, o predicado apagarrepetidos irá apagar os eventuais nomes repetidos.

$\%extensao\ do\ predicado\ que\ identifica\ serviços\ por\ data: data, resultado -> \{V, F\}$

$$idtservico4(X, L) :- solucoes\big(D, \big(consulta(A, X, U, S, P, C), servico(S, P, D, T, Z)\big), R\big),$$
$$apagarrepetidos(R, L).$$

4.19. Extensão do predicado que identifica serviços prestados por custo

Esta extensão recebe como argumentos um custo e uma lista contendo o nome dos serviços prestados com esse custo, sem repetidos. O predicado soluções é invocado para a partir dos predicados consultas que contém o id serviço e o custo passado como argumento e o

predicado serviço que a partir desse id irá buscar o nome do serviço, retornar uma lista com esses nomes. De seguida, o predicado apagarrepetidos irá apagar os eventuais nomes repetidos.

$\%extensao\ do\ predicado\ que\ identifica\ serviços\ por\ custo: custo, resultado -> \{V, F\}$

$$idtservico5(X, L) :- solucoes\big(D, \big(consulta(A, E, U, S, P, X), servico(S, P, D, T, Z)\big), R\big), apagarrepetidos(R, L).$$

4.20. Extensão do predicado que identifica utentes de um serviço

Este predicado recebe como argumentos um nome de um serviço e uma lista contendo todos os nomes de utentes que foram prestados por esse serviço, sem repetidos. O predicado soluções a partir do predicado consultas que contém o id utente, o id serviço e o id prestador, o predicado serviço que contém o id serviço e prestador e o predicado utente que contém o id utente e o nome desse utente, irá retornar uma lista com os nomes de todos os utentes que foram prestados pelo serviço prestado como argumento. Depois com o predicado apagarrepetidos, elimina-se os eventuais nomes repetidos.

$\%extensao\ do\ predicado\ que\ identifica\ utentes\ de\ um\ serviço: serviço, resultado -> \{V, F\}$

$$idtutente4(X, L) :- solucoes(N, (consulta(A, B, U, S, P, C), servico(S, P, X, T, Z), utente(U, N, I, V, W, Y, G, H)), R), apagarrepetidos(R, L).$$

4.21. Extensão do predicado que identifica utentes por instituição

Este predicado recebe como argumentos um nome de uma instituição e uma lista contendo todos os nomes de utentes que foram a essa instituição, sem repetidos. O predicado soluções a partir do predicado consultas que contém o id utente, o id serviço e o id prestador, o predicado serviço que contém o id serviço e prestador e a instituição passada como argumento e o predicado utente que contém o id utente e o nome desse utente, irá retornar uma lista com os nomes de todos os utentes que foram a essa instituição. Depois com o predicado apagarrepetidos, elimina-se os eventuais nomes repetidos.

$\%extensao\ do\ predicado\ que\ identifica\ utentes\ de\ uma\ instituição: instituição, resultado -> \{V, F\}$

$$idtutente5(X, L) :- solucoes(N, (servico(A, P, D, X, C), consulta(F, T, U, A, P, Y), utente(U, N, I, V, W, K, G, H)), R), apagarrepetidos(R, L).$$

4.22. Extensão do predicado que identifica os serviços realizados por id utente

Esta extensão recebe como argumentos um id utente e uma lista contendo todos os nomes dos serviços realizados a esse id utente, sem repetidos. O predicado soluções a partir do predicado serviço que contém o nome do serviço, o id do serviço e o id prestador, o predicado consultas que contém o id serviço, o id prestador e o id utente passado como argumento, irá retornar uma lista com todos os nomes dos serviços realizados a esse id utente. Depois o predicado apagarrepetidos, elimina os nomes repetidos.

$\%extensão\ do\ predicado\ que\ identifica\ serviços\ realizados\ por\ utente: utente, resultado -> \{V, F\}$

$$idtservico6(X, L) :- solucoes\big(D, \big(servico(S, P, D, I, C), consulta(A, J, X, S, P, Y)\big), R\big),$$
$$apagarrepetidos(R, L).$$

4.23. Extensão do predicado que identifica os serviços realizados por instituição

Esta extensão recebe como argumentos uma instituição e uma lista contendo todos os nomes dos serviços realizados nessa instituição, sem repetidos. O predicado soluções a partir do predicado serviço que contém o nome do serviço, o id do serviço, o id prestador e a instituição passada como argumento, o predicado consultas que contém o id serviço, o id prestador irá retornar uma lista com todos os nomes dos serviços realizados nessa instituição. Depois o predicado apagarrepetidos, elimina os nomes repetidos.

$\%extensão\ do\ predicado\ que\ identifica\ serviços\ realizados\ por\ instituição: instituição, resultado -> \{V, F\}$

$$idtservico7(X, L) :- solucoes\big(D, \big(servico(S, P, D, X, C), consulta(A, J, U, S, P, Y)\big), R\big),$$
$$apagarrepetidos(R, L).$$

4.24. Extensão do predicado que identifica serviços realizados por cidade

Esta extensão recebe como argumentos uma cidade e uma lista contendo todos os nomes dos serviços realizados nessa cidade, sem repetidos. O predicado soluções a partir do predicado serviço que contém o nome do serviço, o id do serviço, o id prestador e a cidade passada como argumento, o predicado consultas que contém o id serviço, o id prestador irá retornar uma lista com todos os nomes dos serviços realizados nessa instituição. Depois o predicado apagarrepetidos, elimina os nomes repetidos.

$\%extensão\ do\ predicado\ que\ identifica\ serviços\ realizados\ por\ cidade: cidade, resultado -> \{V, F\}$

$idtservico8(X, R) :- solucoes(D, (servico(S, P, D, I, X), consulta(A, J, U, S, P, C)), R).$

4.25. Extensão do predicado que identifica o custo por utente

A extensão recebe como argumentos um nome de um utente e o custo associado a esse utente. O predicado soluções a partir do predicado consulta que contém o id utente e o custo e o predicado utente que contém o id utente e o nome desse utente passado como argumento, irá retornar uma lista com todos os custos associado a esse nome. Depois com o predicado soma obtêm-se a soma desses custos.

$\%extensão\ do\ predicado\ que\ identifica\ o\ custo\ total\ dos\ cuidados\ de\ saúde\ por\ utente: utente, valor -> \{V, F\}$

$idtcusto(N, V) :- solucoes\big(C, \big(consulta(A, D, U, S, P, C), utente(U, N, Y, V, W, Z, T, E)\big), L\big),$
$soma(L, V).$

4.26. Extensão do predicado que identifica o custo por serviço

A extensão recebe como argumentos um id de um serviço e o custo associado a esse serviço. O predicado soluções a partir do predicado consulta que contém o id serviço passado como argumento irá retornar uma lista com todos os custos associados a esse id. Depois com o predicado soma obtêm-se a soma desses custos.

$\%extensão\ do\ predicado\ que\ identifica\ o\ custo\ total\ dos\ cuidados\ de\ saúde\ por\ serviço: serviço, valor -> \{V, F\}$

$idtcusto2(S, V) :- solucoes(C, consulta(A, D, U, S, P, C), L), soma(L, V).$

4.27. Extensão do predicado que identifica o custo por instituição

A extensão recebe como argumentos um nome de uma instituição e o custo associado a essa instituição. O predicado soluções a partir do predicado consulta que contém o id serviço, o id prestador e o custo e o predicado servico que contem o id serviço, o id prestador e a instituição passada como argumento, irá retornar uma lista com todos os custos associado a essa instituição. Depois com o predicado soma obtêm-se a soma desses custos.

$\%extensão\ do\ predicado\ que\ identifica\ o\ custo\ total\ dos\ cuidados\ de\ saúde\ por\ instituição: serviço, valor -> \{V, F\}$

$$idtcusto3(I, V) :- solucoes\big(C, \big(servico(S, P, H, I, T), consulta(A, D, U, S, P, C)\big), L\big),$$
$$soma(L, V).$$

4.28. Extensão do predicado que identifica o custo por data

A extensão recebe como argumentos uma data e o custo associado a essa data. O predicado soluções a partir do predicado consulta que contém a data passada como argumento irá retornar uma lista com todos os custos associados a essa data. Depois com o predicado soma obtêm-se a soma desses custos.

$\%extensão\ do\ predicado\ que\ identifica\ o\ custo\ total\ dos\ cuidados\ de\ saúde\ por\ data: data, valor -> \{V, F\}$

$$idtcusto4(D, V) :- solucoes(C, consulta(A, D, U, S, P, C), L), soma(L, V).$$

4.29. Extensão do predicado que identifica o custo por id prestador

A extensão recebe como argumentos uma data e o custo associado a esse id prestador. O predicado soluções a partir do predicado consulta que contém o id prestador passado como argumento irá retornar uma lista com todos os custos associados a esse id prestador. Depois com o predicado soma obtêm-se a soma desses custos.

$\%extensão\ do\ predicado\ que\ identifica\ o\ custo\ total\ dos\ cuidados\ de\ saúde\ por\ prestador: prestador, valor -> \{V, F\}$

$$idtcusto5(P, V) :- solucoes(C, consulta(A, D, U, S, P, C), L), soma(L, V).$$

5. INVARIANTES

A manipulação da base de conhecimento foi conseguida através da utilização de invariantes. A utilização destes invariantes garante uma integridade estrutural e referencial da base de conhecimento. Estes impedem a inserção de conhecimento repetido (invariantes estruturais) e asseguram a consistência da informação existente na base de conhecimento (invariantes referenciais).

5.1. Invariantes associados ao predicado utente

5.1.1. Invariante de Inserção Estrutural: não permite a inserção de um novo utente se este já existir na base de conhecimento.

$$+utente(A,B,C,D,E,F,G,H) :: (solucoes((A,B,C,D,E,F,G,H), utente(A,B,C,D,E,F,G,H),S), comprimento(S,N), N == 1).$$

Este invariante estrutural é desenvolvido de forma a não permitir a inserção de factos relativos a utentes já presentes na base de conhecimento. Através do predicado “solucoes” obtém-se todos os factos "utente", cujos argumentos coincidem com Id, nome, idade, cidade, nome do pai, nome da mãe, NIF e doença do utente que se quer inserir, ficando estes guardados numa lista S. No fim da inserção, através do predicado comprimento, garante-se a existência de apenas um utente com os dados inseridos.

5.1.2. Invariante de Inserção Referencial: não permite a inserção de um utente com o mesmo id.

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

Da mesma forma, é criado um invariante que permite que não seja adicionado um utente com o mesmo Id. A criação do invariante segue a mesma estrutura apresentada no invariante anterior, comparando neste caso, apenas o Id dos utentes.

5.1.3. Invariante de Inserção Referencial: não permite a inserção de um utente com o mesmo NIF.

$$+utente(\_,\_,\_,\_,\_,N,\_) :: (solucoes(N, utente(\_,\_,\_,\_,\_,\_,N,\_), S), comprimento(S,R), R == 1).$$

Da mesma forma que o anterior, é criado um invariante que permite que não seja adicionado um utente com o mesmo NIF. A criação do invariante segue a mesma estrutura apresentada nos invariantes anteriores, comparando neste caso, apenas o NIF dos utentes.

5.1.4. Invariante de Inserção Referencial: não permite a inserção de um utente com mais de um pai.

$$+utente(A,B,C,D,E,F,G,H) :: (solucoes(Ps, utente(A,B,C,D,Ps,F,G,H), S), comprimento(S,R), R =< 1).$$

Da mesma forma que os anteriores, é criado um invariante que permite que não seja adicionado um utente com mais que um pai. Através do predicado "solucoes" obtém-se o nome do pai, que fica guardado numa lista S. No fim da inserção, através do predicado comprimento, garante-se que cada utente só tem, no máximo, um pai.

5.1.5. Invariante de Inserção Referencial: não permite a inserção de um utente com mais de uma mãe.

$$+utente(A,B,C,D,E,F,G,H) :: (solucoes(Ms, utente(A,B,C,D,E,Ms,G,H), S), comprimento(S,R), R =< 1).$$

Da mesma forma que os anteriores, é criado um invariante que permite que não seja adicionado um utente com mais que uma mãe. A criação do invariante segue a mesma estrutura apresentada no invariante anterior, só que neste caso é comparado o nome da mãe.

5.1.6. Invariante de Remoção Estrutural: permite a remoção de um utente se este existir na base de conhecimento.

$$-utente(A,B,C,D,E,F,G,H) :: (solucoes((A,B,C,D,E,F,G,H), utente(A,B,C,D,E,F,G,H), S), comprimento(S,N), N == 1).$$

Neste caso, é criado um invariante que permite que só haja remoção de um utente, caso este se encontre na base de conhecimento. A criação do invariante segue a mesma estrutura apresentada no invariante estrutural anterior.

5.1.7. Invariante de Remoção Referencial: permite a remoção de um utente se este não estiver associado a nenhuma consulta.

$$-utente(A,B,C,D,E,F,G,H) :: (solucoes(A, consulta(\_,\_,A,\_,\_,\_), S), comprimento(S,N), N == 0).$$

Da mesma forma que os anteriores, é criado um invariante que permite que só haja remoção de um utente, caso este não esteja associado a nenhuma consulta. Através do predicado solucoes obtém-se uma lista S com o id do utente em cada consulta. Se esta lista for vazia, então o utente com o id em questão pode ser removido da base de conhecimento.

Tal como os invariantes anteriores (associados ao predicado utente), todos os invariantes a seguir associados aos predicados prestador, servico e consulta seguem a mesma estrutura.

5.2. Invariantes associados ao predicado prestador

5.2.1. Invariante de Inserção Estrutural: não permite a inserção de um novo prestador se este já existir na base de conhecimento.

$$+prestador(A,B,C,D) :: (solucoes((A,B,C,D), prestador(A,B,C,D), S), comprimento(S,N), N == 1).$$

5.2.2. Invariante de Inserção Referencial: não permite a inserção de um prestador com o mesmo id.

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

5.2.3. Invariante de Remoção Estrutural: permite a remoção de um prestador se este existir na base de conhecimento.

$$-prestador(A,B,C,D) :: (solucoes((A,B,C,D), prestador(A,B,C,D), S), comprimento(S,N), N == 1).$$

5.2.4. Invariante de Remoção Referencial: permite a remoção de um prestador se este não estiver associado a nenhum serviço.

$$-prestador(A,B,C,D) :: (solucoes(A, servico(\_,A,\_,\_,\_), S), comprimento(S,N), N == 0).$$

5.3. Invariantes associados ao predicado serviço

5.3.1. Invariante de Inserção Estrutural: não permite a inserção de um novo serviço se este já existir na base de conhecimento.

$$+servico(A,B,C,D,E) :: (solucoes((A,B,C,D,E), servico(A,B,C,D,E), S), comprimento(S,N), N == 1).$$

5.3.2. Invariante de Inserção Referencial: não permite a inserção de um serviço com o mesmo id.

$$+servico(A,\_,\_,\_,\_) :: (solucoes(A, servico(A,\_,\_,\_,\_), S), comprimento(S,N), N == 1).$$

5.3.3. Invariante de Remoção Estrutural: permite a remoção de um serviço se este existir na base de conhecimento.

$$-serviço(A,B,C,D,E) :: (solucoes((A,B,C,D,E), serviço(A,B,C,D,E), S), comprimento(S,N), N == 1).$$

5.4. Invariantes associados ao predicado consulta

5.4.1. Invariante de Inserção Estrutural: não permite a inserção de uma nova consulta se esta já existir na base de conhecimento.

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

5.4.2. Invariante de Inserção Referencial: não permite a inserção de uma consulta com o mesmo id.

$$+consulta(A,\_,\_,\_,\_,\_) :: (solucoes(A, consulta(A,\_,\_,\_,\_,\_), S), comprimento(S, N), N == 1).$$

5.4.3. Invariante de Remoção Estrutural: permite a remoção de uma consulta se esta existir na base de conhecimento.

$$-consulta(A,B,C,D,E,F) :: (solucoes((A,B,C,D,E,F), consulta(A,B,C,D,E,F), S), comprimento(S, N), N == 1).$$

6. CONCLUSÃO

A realização deste trabalho permitiu-nos aplicar a linguagem Prolog num caso prático e verificar a sua poderosa vertente de linguagem declarativa. Para tal, foi necessário elaborar vários factos, predicados, extensões de predicados e invariantes para tornar a nossa base de conhecimento consistente. Desta forma, foi possível desenvolver as competências adquiridas e praticadas durantes as aulas desta Unidade Curricular.

Para a realização da base de dados clínica proposta é plausível explorar várias resoluções e formas de pensamento para a sua implementação, permitindo, assim, o desenvolvimento de mecanismos de raciocínio de nível avançado. Visto que, todos os predicados propostos juntamente com os que acrescentamos foram implementados e testados, é possível mostrar que os objetivos para este trabalho prático nº1 foram cumpridos.

7. REFERÊNCIAS

[1] Lago, Silvio. Introdução à linguagem prolog.

[2] Favero, Eloi L. Progração em Prolog - uma abordagem prática, CCEN-UFPA, 2006.

[3] https://www.swi-prolog.org/, consultado a 06/11/2020

[4] https://www.ensinoeinformacao.com/computacao-linguagem-prolog, consultado a 07/11/2020

[5] http://www.linhadecodigo.com.br/artigo/1697/descobrindo-o-prolog.aspx, consultado a 06/10/2020

8. ANEXOS

Anexo A: Factos da base de conhecimento

%$\boldsymbol{Predicado\ utente}(\boldsymbol{idutente}, \boldsymbol{nome}, \boldsymbol{idade}, \boldsymbol{localidade}, \boldsymbol{nome_{pai}}, \boldsymbol{nome_{mae}}, \boldsymbol{NIF}, \boldsymbol{doença})$.

$utente(100, margaridaalves, 19, braga, manuelalves, joanaalves, 123456789, asma)$.

$utente(101, josepereira, 23, porto, henriquepereira, anapereira, 456789123, bronquite)$.

$utente(102, luisdelgado, 33, famalicao, luisdelgado, dianadelgado, 345678912, covid)$.

$utente(103, rodrigomendes, 25, barcelos, josemendes, luisamendes, 789123456, covid)$.

$utente(104, claudiasoares, 20, famalicao, andresoares, joanasoares, 891234567, febre)$.

$utente(105, ruigoncalves, 41, braga, sergiogoncalves, inesgoncalves, 912345678, diarreia)$.

$utente(106, leonorgarcia, 9, guimaraes, diogogarcia, saragarcia, 657891234, covid)$.

$utente(107, mariamagalhaes, 29, porto, ricardomagalhaes, soniamagalhaes, 567891234, bronquite)$.

$utente(108, manuelsilva, 23, braga, joaosilva, teresasilva, 213456789, pneumonia)$.

$utente(109, fernandocerqueira, 12, braga, nunocerqueira, joanacerqueira, 765891234, covid)$.

$utente(110, ismaelbarbosa, 54, barcelos, jorgebarbosa, patriciabarbosa, 987654321, asma)$.

%$\boldsymbol{Predicado\ servico}(\boldsymbol{idservico}, \boldsymbol{idprestador}, \boldsymbol{descricao}, \boldsymbol{instituicao}, \boldsymbol{cidade})$.

$servico(200,1005, oscultar, hospitaldatrofa, trofa)$.

$servico(201,1003, medirtensao, hospitaldatrofa, trofa)$.

$servico(202,1010, analises, hospitalsantaluzia, vianadocastelo)$.

$servico(203,1004, medirfebre, hospitalpublicodebraga, braga)$.

$servico(204,1006, analisessecrecoes, hospitalprivadodebraga, braga)$.

$servico(205,1002, medirtensao, hospitalpublicodebraga, braga)$.

$servico(206,1001, oscultar, hospitalpublicodebraga, braga)$.

$servico(207,1008, oscultar, hospitalprivadodebraga, braga)$.

$servico(208,1004, analises, hospitalpublicodebraga, braga)$.

%***Predicado consulta***(***idconsulta***, ***data***, ***idutente***, ***idservico***, ***idprestador***, ***preço***).

consulta(1,06112020,100,200,1005,80).

consulta(2,10102020,105,204,1006,110).

consulta(3,22122020,110,206,1001,30).

consulta(4,25112020,109,208,1004,25).

consulta(5,30112020,106,203,1004,30).

consulta(6,10102020,109,202,1010,60).

consulta(7,06112020,107,206,1001,30).

consulta(8,25112020,102,203,1004,35).

consulta(9,30112020,110,206,1001,30).

consulta(10,10102020,103,203,1004,40).

consulta(11,06112020,104,203,1004,25).

consulta(12,30112020,109,208,1004,35).

%***Predicado prestador***(***idprestador***, ***nome***, ***especialidade***, ***instituicao***).

prestador(1000, *joaovasquez*, *medicinageral*, *hospitalpublicodebraga*).

prestador(1001, *thomaskall*, *pneumologista*, *hospitalpublicodebraga*).

prestador(1002, *anasilva*, *medicinageral*, *hospitalprivadodebraga*).

prestador(1003, *josecorreia*, *pediatra*, *hospitaldatrofa*).

prestador(1004, *gabrielarodrigues*, *medicinageral*, *hospitalpublicodebraga*).

prestador(1005, *franciscomenezes*, *pneumologista*, *hospitaldatrofa*).

prestador(1006, *franciscaalves*, *endocrinologia*, *hospitalprivadodebraga*).

prestador(1007, *danielnunes*, *endocrinologia*, *hospitalsantaluzia*).

prestador(1008, *henriquelage*, *pneumologista*, *hospitalprivadodebraga*).

prestador(1009, *saralurdes*, *pediatra*, *hospitalsantaluzia*).

prestador(1010, *hugopereira*, *medicinageral*, *hospitalsantaluzia*).

Anexo B: Perguntas no SICStus

```
| ?- evolucao(utente(111,jorgerodrigues,25,porto,vascorodrigues,anarodrigues,315672346,covid)).
yes
```

Figura 1 - Exemplo de inserir um novo utente

```
| ?- evolucao(utente(110,jorgerodrigues,25,porto,vascorodrigues,anarodrigues,315672346,covid)).
no
```

Figura 2 - Exemplo de não ocorrer a inserção de um utente com um ID já existente

```
| ?- evolucao(utente(110,ismaelbarbosa,54,barcelos,jorgebarbosa,patriciabarbosa,987654321,asma))
no
```

Figura 3 - Exemplo de não ocorrer inserção de conhecimento repetido no predicado utente

```
| ?- evolucao(utente(111,jorgerodrigues,25,porto,vascorodrigues,anarodrigues,123456789,covid)).
no
```

Figura 4 - Exemplo de não ocorrer inserção de um utente com o mesmo nif

```
| ?- evolucao(112,ismaelbarbosa,54,barcelos,ruibarbosa,patriciabarbosa,987654331,asma).
no
```

Figura 5 -Exemplo de não ocorrer inserção de um utente com outro pai

```
| ?- evolucao(prestador(1000,joaovasquez,medicinageral,hospitalpublicodebraga))
no
| ?- evolucao(prestador(1000,jorgeovasquez,medicinageral,hospitalpublicodebraga)
no
```

Figura 6 - Exemplo de não permitir a inserção de conhecimento repetido no predicado prestador e da não inserção de um prestador com o mesmo ID

```
| ?- evolucao(servico(200,1005,oscultar,hospitaltrofa,trofa)).
no
| ?- evolucao(servico(200,1006,verfebre,hospitaltrofa,trofa)).
no
| ?- listing(consulta).
consulta(1, 6112020, 100, 200, 1005, 80).
consulta(2, 10102020, 105, 204, 1006, 110).
consulta(3, 22122020, 110, 206, 1001, 30).
consulta(4, 25112020, 109, 208, 1004, 25).
consulta(5, 30112020, 106, 203, 1004, 30).
consulta(6, 10102020, 109, 202, 1010, 60).
consulta(7, 6112020, 107, 206, 1001, 30).
consulta(8, 25112020, 102, 203, 1004, 35).
consulta(9, 30112020, 110, 206, 1001, 30).
consulta(10, 10102020, 103, 203, 1004, 40).
consulta(11, 6112020, 104, 203, 1004, 25).
consulta(12, 30112020, 109, 208, 1004, 35).

yes
| ?- evolucao(consulta(1,06112020,100,200,1005,80)).
no
| ?- evolucao(consulta(1,06112020,102,201,1005,80)).
no
```

Figura 7 - Exemplos de não inserção de um serviço ou consulta com o mesmo ID e de não inserção de conhecimento repetido tanto no predicado serviço como no predicado consulta

```
| ?- idtinstituicoes(hospitalprivadodebraga)
yes
```

Figura 8 - Exemplo de identificação se uma instituição presta serviços

```
| ?- idtinstituicoes2(oscultar,R).
R = [hospitaldatrofa,hospitalpublicodebraga,hospitalprivadodebraga] ?
yes
| ?- idtutente(23,R).
R = [josepereira,manuelsilva] ?
yes
```

Figura 9 - Exemplo de identificação de uma instituição que presta o serviço "oscultar" e exemplo de identificação dos utentes com a idade 23

```
| ?- idtutente2(braga,R).
R = [margaridaalves,ruigoncalves,manuelsilva,fernandocerqueira]
yes
```

Figura 10 - Exemplo de identificação dos utentes que vivem em Braga

```
| ?- idtutente3(covid,R).
R = [luisdelgado,rodrigomendes,leonorgarcia,fernandocerqueira] ?
yes
| ?-
```

Figura 11 - Exemplo de identificação dos utentes que têm covid

```
| ?- idtservico(1004,R).
R = [medirfebre,analises] ?
yes
```

Figura 12 - Exemplo de identificação dos serviços prestados pelo prestador de ID 1004

```
| ?- idtservico2(hospitalprivadodebraga,R).
R = [analisessecrecoes,oscultar] ?
yes
```

Figura 13 - Exemplo de identificação dos serviços prestado no hospital privado de Braga

```
| ?- idtservico3(braga,R).
R = [medirfebre,analisessecrecoes,medirtensao,oscultar,oscultar,analises] ?
yes
```

Figura 14 - Exemplo dos serviços prestados na cidade de Braga

```
| ?- idtservico4(06112020,R).
R = [oscultar,medirfebre] ?
yes
```

Figura 15 - Exemplo dos serviços prestados na data 06/11/2020

```
| ?- idtservico5(30,R)

R = [oscultar,medirfebre] ?
```

Figura 16 - Exemplo dos serviços prestados que têm o custo de 30 unidades

```
| ?- idtconsulta(06112020,R)
R = [1,7,11] ?
yes
```

Figura 17 - Exemplo de ID consultas ocorridas na data de 06/11/2020

```
| ?- idtconsulta2(100,R)
R = [1] ?
yes
```

Figura 18 - Exemplo dos ID consulta no utente de ID 100

```
| ?- idtconsulta3(202,R).
R = [6] ?
yes
| ?- idtconsulta4(1004,R).
R = [4,5,8,10,11,12] ?
yes
| ?- idtconsulta5(30,R).
R = [3,5,7,9] ?
```

Figura 19 - Exemplo de ID consultas por ID serviço, ID prestador e por custo, respetivamente

```
| ?- idtutente4(oscultar,R).
R = [margaridaalves,ismaelbarbosa,mariamagalhaes] ?
yes
```

Figura 20 - Exemplo de utentes do serviço "oscultar"

```
| ?- idtutente5(hospitalprivadodebraga,R).
R = [ruigoncalves] ?
yes
```

Figura 21 - Exemplo de utentes que foram ao hospital privado de Braga

```
| ?- idtservico6(100,R).
R = [oscultar] ?
```

Figura 22 - Exemplo de serviços realizados ao utente de ID 100

```
| ?- idtservico7(hospitaldatrofa,R).
R = [oscultar] ?
yes
```

Figura 23 - Exemplo de serviços realizados no hospital da Trofa

```
| ?- idtservico8(trofa,R).
R = [oscultar] ?
yes
```

Figura 24 - Exemplo de serviços realizados na cidade da Trofa

```
| ?- idtcusto(margaridaalves,R).
R = 80 ?
yes
```

Figura 25 - Exemplo do custo total para o utente Margarida Alves

```
| ?- idtcusto2(203,R).
R = 130 ?
yes
| ?-
```

Figura 26 - Exemplo do custo total para o serviço de ID 203

```
| ?- idtcusto3(hospitaldatrofa,R).
R = 80 ?
yes
| ?-
```

Figura 27 - Exemplo do custo total para a instituição “Hospital da Trofa”

```
| ?- idtcusto4(06112020,R).
R = 135 ?
yes
| ?-
```

Figura 28 - Exemplo do custo total na data 06/11/2020

```
| ?- idtcusto5(1001,R).
R = 90 ?
yes
| ?-
```

Figura 29 - Exemplo do custo total para o prestador de ID 1001

```
| ?- listing(utente).
utente(100, margaridaalves, 19, braga, manuelalves, joanaalves, 123456789, asma).
utente(101, josepereira, 23, porto, henriquepereira, anapereira, 456789123, bronquite).
utente(102, luisdelgado, 33, famalicao, luisdelgado, dianadelgado, 345678912, covid).
utente(103, rodrigomendes, 25, barcelos, josemendes, luisamendes, 789123456, covid).
utente(104, claudiasoares, 20, famalicao, andresoares, joanasoares, 891234567, febre).
utente(105, ruigoncalves, 41, braga, sergiogoncalves, inesgoncalves, 912345678, diarreia).
utente(106, leonorgarcia, 9, guimaraes, diogogarcia, saragarcia, 657891234, covid).
utente(107, mariamagalhaes, 29, porto, ricardomagalhaes, soniamagalhaes, 567891234, bronquite).
utente(108, manuelsilva, 23, braga, joaosilva, teresasilva, 213456789, pneumonia).
utente(109, fernandocerqueira, 12, braga, nunocerqueira, joanacerqueira, 765891234, covid).
utente(110, ismaelbarbosa, 54, barcelos, jorgebarbosa, patriciabarbosa, 987654321, asma).

yes
| ?- involucao(utente(101,josepereira, 23,porto, henriquepereira, anapereira, 456789123,bronquite))
yes
| ?- listing(utente).
utente(100, margaridaalves, 19, braga, manuelalves, joanaalves, 123456789, asma).
utente(102, luisdelgado, 33, famalicao, luisdelgado, dianadelgado, 345678912, covid).
utente(103, rodrigomendes, 25, barcelos, josemendes, luisamendes, 789123456, covid).
utente(104, claudiasoares, 20, famalicao, andresoares, joanasoares, 891234567, febre).
utente(105, ruigoncalves, 41, braga, sergiogoncalves, inesgoncalves, 912345678, diarreia).
utente(106, leonorgarcia, 9, guimaraes, diogogarcia, saragarcia, 657891234, covid).
utente(107, mariamagalhaes, 29, porto, ricardomagalhaes, soniamagalhaes, 567891234, bronquite).
utente(108, manuelsilva, 23, braga, joaosilva, teresasilva, 213456789, pneumonia).
utente(109, fernandocerqueira, 12, braga, nunocerqueira, joanacerqueira, 765891234, covid).
utente(110, ismaelbarbosa, 54, barcelos, jorgebarbosa, patriciabarbosa, 987654321, asma).

yes
| ?-
```

Figura 30 - Exemplo de remoção de um utente

```
yes
| ?- listing(prestador).
prestador(1000, joaovasquez, medicinageral, hospitalpublicodebraga).
prestador(1001, thomaskall, pneumologista, hospitalpublicodebraga).
prestador(1002, anasilva, medicinageral, hospitalprivadodebraga).
prestador(1003, josecorreia, pediatra, hospitaldatrofa).
prestador(1004, gabrielarodrigues, medicinageral, hospitalpublicodebraga).
prestador(1005, franciscomenezes, pneumologista, hospitaldatrofa).
prestador(1006, franciscaalves, endocrinologia, hospitalprivadodebraga).
prestador(1007, danielnunes, endocrinologia, hospitalsantaluzia).
prestador(1008, henriquelage, pneumologista, hospitalprivadodebraga).
prestador(1009, saralurdes, pediatra, hospitalsantaluzia).
prestador(1010, hugopereira, medicinageral, hospitalsantaluzia).

yes
| ?- involucao(prestador(1007,danielnunes,endocrinologia,hospitalsantaluzia)).
yes
| ?- listing(prestador).
prestador(1000, joaovasquez, medicinageral, hospitalpublicodebraga).
prestador(1001, thomaskall, pneumologista, hospitalpublicodebraga).
prestador(1002, anasilva, medicinageral, hospitalprivadodebraga).
prestador(1003, josecorreia, pediatra, hospitaldatrofa).
prestador(1004, gabrielarodrigues, medicinageral, hospitalpublicodebraga).
prestador(1005, franciscomenezes, pneumologista, hospitaldatrofa).
prestador(1006, franciscaalves, endocrinologia, hospitalprivadodebraga).
prestador(1008, henriquelage, pneumologista, hospitalprivadodebraga).
prestador(1009, saralurdes, pediatra, hospitalsantaluzia).
prestador(1010, hugopereira, medicinageral, hospitalsantaluzia).

yes
```

Figura 31-Exemplo de remoção de um prestador

```
| ?- listing(servico).
servico(200, 1005, oscultar, hospitaldatrofa, trofa).
servico(201, 1003, medirtensao, hospitaldatrofa, trofa).
servico(202, 1010, analises, hospitalsantaluzia, vianadocastelo).
servico(203, 1004, medirfebre, hospitalpublicodebraga, braga).
servico(204, 1006, analisessecrecoes, hospitalprivadodebraga, braga).
servico(205, 1002, medirtensao, hospitalpublicodebraga, braga).
servico(206, 1001, oscultar, hospitalpublicodebraga, braga).
servico(207, 1008, oscultar, hospitalprivadodebraga, braga).
servico(208, 1004, analises, hospitalpublicodebraga, braga).

yes
| ?- involucao(servico(205,1002,medirtensao,hospitalpublicodebraga,braga)).
yes
| ?- listing(servico).
servico(200, 1005, oscultar, hospitaldatrofa, trofa).
servico(201, 1003, medirtensao, hospitaldatrofa, trofa).
servico(202, 1010, analises, hospitalsantaluzia, vianadocastelo).
servico(203, 1004, medirfebre, hospitalpublicodebraga, braga).
servico(204, 1006, analisessecrecoes, hospitalprivadodebraga, braga).
servico(206, 1001, oscultar, hospitalpublicodebraga, braga).
servico(207, 1008, oscultar, hospitalprivadodebraga, braga).
servico(208, 1004, analises, hospitalpublicodebraga, braga).

yes
```

Figura 32 – Exemplo de remoção de um serviço

```
| ?- listing(consulta)

consulta(1, 6112020, 100, 200, 1005, 80).
consulta(2, 10102020, 105, 204, 1006, 110).
consulta(3, 22122020, 110, 206, 1001, 30).
consulta(4, 25112020, 109, 208, 1004, 25).
consulta(5, 30112020, 106, 203, 1004, 30).
consulta(6, 10102020, 109, 202, 1010, 60).
consulta(7, 6112020, 107, 206, 1001, 30).
consulta(8, 25112020, 102, 203, 1004, 35).
consulta(9, 30112020, 110, 206, 1001, 30).
consulta(10, 10102020, 103, 203, 1004, 40).
consulta(11, 6112020, 104, 203, 1004, 25).
consulta(12, 30112020, 109, 208, 1004, 35).

yes
| ?- involucao(consulta(1,06112020,100,200,1005,80)).
yes
| ?- listing(consulta).
consulta(2, 10102020, 105, 204, 1006, 110).
consulta(3, 22122020, 110, 206, 1001, 30).
consulta(4, 25112020, 109, 208, 1004, 25).
consulta(5, 30112020, 106, 203, 1004, 30).
consulta(6, 10102020, 109, 202, 1010, 60).
consulta(7, 6112020, 107, 206, 1001, 30).
consulta(8, 25112020, 102, 203, 1004, 35).
consulta(9, 30112020, 110, 206, 1001, 30).
consulta(10, 10102020, 103, 203, 1004, 40).
consulta(11, 6112020, 104, 203, 1004, 25).
consulta(12, 30112020, 109, 208, 1004, 35).

yes
```

Figura 33 - Exemplo de remoção de uma consulta

```
| ?- listing(utente).
utente(100, margaridaalves, 19, braga, manuelalves, joanaalves, 123456789, asma).
utente(102, luisdelgado, 33, famalicao, luisdelgado, dianadelgado, 345678912, covid).
utente(103, rodrigomendes, 25, barcelos, josemendes, luisamendes, 789123456, covid).
utente(104, claudiasoares, 20, famalicao, andresoares, joanasoares, 891234567, febre).
utente(105, ruigoncalves, 41, braga, sergiogoncalves, inesgoncalves, 912345678, diarreia).
utente(106, leonorgarcia, 9, guimaraes, diogogarcia, saragarcia, 657891234, covid).
utente(107, mariamagalhaes, 29, porto, ricardomagalhaes, soniamagalhaes, 567891234, bronquite).
utente(108, manuelsilva, 23, braga, joaosilva, teresasilva, 213456789, pneumonia).
utente(109, fernandocerqueira, 12, braga, nunocerqueira, joanacerqueira, 765891234, covid).
utente(110, ismaelbarbosa, 54, barcelos, jorgebarbosa, patriciabarbosa, 987654321, asma).

yes
| ?- nao(utente(100,margaridaalves,19,braga,manuelalves,joanaalves,123456789,asma)).
no
| ?- nao(utente(112,margaridaalves,19,braga,manuelalves,joanaalves,123456789,asma)).
yes
| ?-
```

Figura 34 – Exemplo de predicado não.