Relatório do Projeto de SRCR

Bruno Veloso (a78352) Jaime Leite (a80757)
João Pimentel (a80874) Rodolfo Silva (a81716)
Pedro Gonçalves (a82313)

Braga, março de 2019

Universidade do Minho
Mestrado Integrado em Engenharia Informática
Sistemas de Representação de Conhecimento e Raciocínio
(2º Semestre/2018-2019)
Grupo 3

Resumo

Este projeto teve como objetivo o refinamento da utilização da linguagem de programação em lógica *Prolog*, no âmbito da representação de conhecimento e construção de raciocínio para resolução de problemas.

A lógica trata-se de uma ciência formal desprovida de conteúdo, que se dedica ao estudo das formas válidas de inferência. Refere-se, assim, ao estudo dos métodos e dos princípios utilizados para distinguir o raciocínio correcto do incorrecto.

Recorrendo à programação em lógica, foi possível criar uma base de conhecimento relativa a uma área de prestação serviços com sucesso.

Conteúdo

Lista de Figuras

1 Introdução

O principal objetivo deste projeto consiste na solidificação da utilização da linguagem de programação em lógica *Prolog*, no âmbito da representação de conhecimento e construção de raciocínio para resolução de problemas de forma otimizada.

O tema em estudo é a área de prestação de serviços, como por exemplo um centro de saúde, onde existem, serviços a realizar e consultas que relacionam os anteriores.

Em suma, a motivação deste projeto foca-se na consolidação do estudo do panorama de programacão em lógica, atráves de um exercício que consiste na gestão de uma base de conhecimento sobre uma área de prestação de serviços.

2 Preliminares

Para a total compreensão do projeto desenvolvido, é necessário que o leitor possua conhecimentos prévios no paradigma de programação em lógica. Assim sendo, o conhecimento adquirido nas primeiras aulas da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio, torna-se essencial para um desenvolvimento coerente desta proposta.

Alguns factos que podem ajudar a compreender a lógica de pensamento por detrás desta linguagem são o facto de se basear nos princípios do mundo fechado, nome único e domínio fechado. Quer isto dizer que um nome representa sempre o mesmo indivíduo e tudo o que não for dado como um facto conhecido é considerado falso para a base de conhecimento.

3 Descrição do Trabalho e Análise de Resultados

Neste capítulo será discutido todo o processo de resolução do enunciado proposto pelos docentes da unidade curricular.

3.1 Inserção de Conhecimento

De forma a inserir o conhecimento necessário, foram construídos três predicados referentes a utentes (Figura 1), serviços (Figura 2) e consultas (Figura 3). A utilização destes predicados permitiu, mais tarde, chegar a respostas sobre as interrogações propostas.

```
% utente: #IdUt, Nome, Idade, Morada -> { V, F }
utente(1,joao_pimentel,20,esposende).
utente(2,bruno_veloso,22,ponte_da_barca).
utente(3,jaime_leite,20,felgueiras).
utente(4,pedro_goncalves,20,felgueiras).
utente(5,rodolfo_silva,20,trofa).
utente(6,joana_amorim,22,viana_do_castelo).
utente(7,gloria_borga,53,portimao).
utente(8,andreia_silva,27,trofa).
```

Figura 1 - Inserção de conhecimento relativo ao Utente.

```
% serviço: #IdServ, Descrição, Instituição, Cidade -> { V, F }
servico(1,medicina_geral,trofa_saude,braga).
servico(2,cardiologia,cuf,porto).
servico(3,oftalmologia,sao_joao,porto).
servico(4,otorrinolarengologia,particular,viana).
servico(5,ginecologia,trofa_saude,braga).
servico(6,cirurgia_geral,santo_antonio,porto).
```

Figura 2 - Inserção de conhecimento relativo ao Serviço.

```
% consulta: Data, #IdUt, #IdServ, Custo  -> { V, F }
consulta(2019-03-06,1,1,14).
consulta(2014-04-06,1,3,22).
consulta(2014-03-07,3,5,32).
consulta(2019-05-12,5,2,9).
consulta(2019-02-26,2,5,10).
consulta(2019-12-29,3,2,15).
consulta(2019-3-7, 1, 1, 20).
```

Figura 3 - Inserção de conhecimento relativo à Consulta.

O registo destas entidades necessita de uma atenção elevada, de modo a não invalidar o pressuposto da representação de conhecimento. Assim, de modo a garantir a coerência de dados na base de conhecimento, foi necessário o desenvolvimento de invariantes.

Os invariantes funcionam como uma prova sobre o predicado em questão, sendo utilizados tanto no momento de inserção como de remoção, garantindo consistência.

Estes invariantes são, posteriormente, utilizados juntamente com um predicado que insere/retira (Figuras 4 e 5, respetivamente), semelhante ao apresentado pelo docente no decorrer das aulas teóricas. É merecedor de destaque que, aliados aos diversos invariantes, estes predicados de evolução/involução permitem inserir/remover conhecimento de forma adequada.

```
% ------------------------------------------------------
% Registar utentes, serviços e consultas;
adiciona(T) :- solucoes(Inv, +T::Inv, S),
               insere(T),
               testa(S).

insere(T) :- assert(T).
insere(T) :- retract(T),!,fail.

testa([]).
testa([H|T]) :- H, testa(T).
```

Figura 4 - Predicados auxiliares à inserção de conhecimento.

```
% Remover utentes, serviços e consultas;

remove(T) :- solucoes(I,-T::I,S),
             apagar(T),
             testa(S).

apagar(T) :- retract(T).
apagar(T) :- assert(T),!,fail.
```

Figura 5 - Predicados auxiliares à remoção de conhecimento.

3.2 Invariantes

3.2.1 Invariantes Estruturais

Os invariantes apresentados na Figura 6 servem para garantir que não é inserido conhecimento repetido, ou seja, redundante na base de conhecimento.

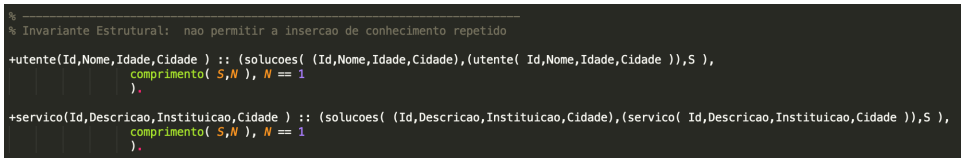


Figura 6 - Invariantes estruturais que não permitem a inserção de conhecimento já existente.

3.2.2 Invariantes Referenciais

Os invariantes que se podem observar na Figura 8 foram elaborados a pensar no facto de um *id* ser um identificador único de um utente, não sendo, assim, possível a existência do mesmo para mais do que uma pessoa. Além disso, como um serviço também deve ser identificado de forma fácil, também o seu *id* deve ser único.

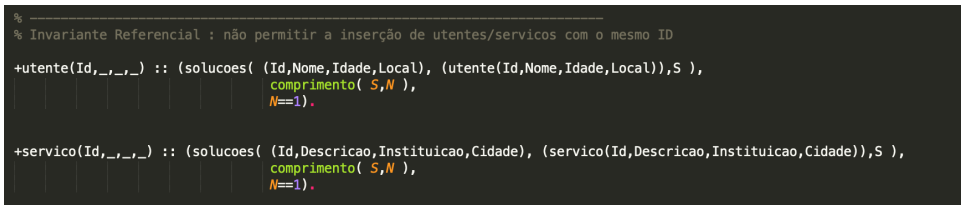


Figura 7 - Invariantes referenciais que garantem a manutenção do conhecimento sobre os *ids*.

Já no caso dos serviços, foi idealizado que um serviço com uma descrição, realizado numa determinada instituição e cidade, apenas poderia ser identificado por um *id*, pois qualquer replicação, apenas com mudança do último, não deixa de se tratar do original. Assim, veja-se a Figura 8.

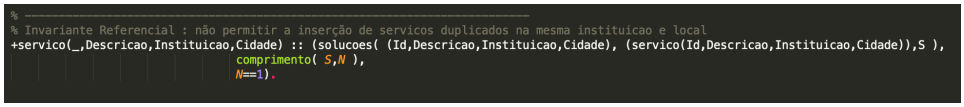


Figura 8 - Invariante referencial que garante a manutenção do conhecimento sobre os serviços.

No que toca ao caso das consultas, é necessário ter a garantia de que o utente ao qual a mesma está a ser associada existe, tal como o serviço, já que não faz sentido marcar consulta para algo não existente (Figura 9).

```
% Invariante Referencial: garantir que utente/servico existe antes de marcar consulta
+consulta(_,IdUt,_,_) :: (solucoes( (IdUt), (utente(IdUt,Nome,Idade,Local)),S ),
                          comprimento( S,N ),
                          N==1).

+consulta(_,_,IdServ,_) :: (solucoes( (IdServ), (servico(IdServ,Descricao,Instituicao,Cidade)),S ),
                          comprimento( S,N ),
                          N==1).
```

Figura 9 - Invariantes referenciais que garantem a manutenção do conhecimento sobre as consultas .

Note-se que as consultas não podem ter invariante que garanta a não existência de conhecimento repetido como foi falado até então, uma vez que um utente pode ter associada mais que uma consulta para o mesmo serviço no mesmo dia.

Por fim, como se vê na Figura 10, foi considerado que não poderiam ser removidos utentes e/ou serviços se ainda possuíssem consultas a si associadas. Assim, evitam-se condições de inconsistência aquando da remoção e, posterioemente, fosse necessária a consulta do mesmo, a partir dos dados na consulta.

```
% Invariante Referencial:  não permitir a remoção de utentes/serviços que têm uma consulta associada

-utente(IdUt,_,_,_) :: (solucoes( (IdUt),(consulta(_,IdUt,_,_)),S ),
                        comprimento( S,N ),
                        N==0).

-servico(IdServ,Nome,Especialidade,Local) :: (solucoes( (IdServ,Nome,Especialidade,Local), consulta(_,_,IdServ,_),S ),
                                    comprimento( S,N ),
                                    N==0).
```

Figura 10 - Invariantes referenciais que garantem a manutenção do conhecimento no momento de remoção.

3.3 Resposta às *queries* propostas

3.3.1 Identificar as instituições prestadoras de serviços

De modo a identificar as instituições prestadoras de serviços, é necessário encontrar todos os campos em análise a partir do predicado serviços. Como uma instituição pode existir em mais do que um local, as repetições não devem ser apresentadas, daí o uso do predicado *sort*, como se vê na Figura 11.

Além disso, foi definido um predicado para determinar quais os serviços proporcionados na base de conhecimento, tendo uma linha de pensamento semelhante ao anterior.

```
% Predicado instituicoes:
% Lista -> {V,F}
instituicoes(S):-
    solucoes(I, servico(_,_,I,_), L),
    sort(L,S).

% Predicado servicos:
% Lista -> {V,F}
servicos(S):-
    solucoes(D, servico(_,D,_,_), L),
    sort(L,S).
```

Figura 11 - Identificação das instituições prestadoras de serviços.

3.3.2 Identificar utentes/serviços/consultas por critérios de seleção

3.3.2.1 Utentes

Na identificação de um utente pode ser efetuada a pesquisa por um identificador, um nome, uma idade ou um local. O resultado de cada um dos predicados é obtido com o auxílio do predicado *solucoes*, que coloca numa lista a filtragem correspondente aos parâmetros indicados, como é visível na Figura 12.

```
% Identificar utentes/serviços/consultas por critérios de seleção;

% Predicado identificar_utente_por_Id:
% Id, Lista de Utentes -> {V,F}
identificar_utente_por_id(Id,S):-
  solucoes(
          (Nome,Idade,Local),
          (utente(Id,Nome,Idade,Local)),
          S
          ).

% Predicado identificar_utente_por_Nome:
% nome, Lista de Utentes -> {V,F}
identificar_utente_por_nome(Nome,S):-
  solucoes(
          (Id,Idade,Local),
          (utente(Id,Nome,Idade,Local)),
          S
          ).

% Predicado identificar_utente_por_Idade:
% Idade, Lista de Utentes -> {V,F}
identificar_utente_por_idade(Idade,S):-
  solucoes(
          (Id,Nome,Local),
          (utente(Id,Nome,Idade,Local)),
          S
          ).

% Predicado identificar_utente_por_Local:
% Local, Lista de Utentes -> {V,F}
identificar_utente_por_local(Local,S):-
  solucoes(
          (Id,Nome,Idade),
          (utente(Id,Nome,Idade,Local)),
          S
          ).
```

Figura 12 - Identificar utentes por critérios de seleção.

3.3.2.2 Serviços

Um serviço pode ser identificado pelo seu identificador, a sua descrição, a sua instituição ou cidade. Para tal, é usado, mais uma vez, o predicado *solucoes*, como no caso anterior (Figura 13).

```
% Predicado identificar_servico_por_Id:
% Id, Lista de Servicos -> {V,F}
identificar_servico_por_id(Id,S):-
  solucoes(
          (Descricao,Instituicao,Cidade),
          (servico(Id,Descricao,Instituicao,Cidade)),
          S
          ).

% Predicado identificar_servico_por_Descricao:
% Descricao, Lista de Servicos -> {V,F}
identificar_servico_por_descricao(Descricao,S):-
  solucoes(
          (Id,Instituicao,Cidade),
          (servico(Id,Descricao,Instituicao,Cidade)),
          S
          ).

% Predicado identificar_servico_por_Instituicao:
% Instituicao, Lista de Servicos -> {V,F}
identificar_servico_por_instituicao(Instituicao,S):-
  solucoes(
          (Id,Descricao,Cidade),
          (servico(Id,Descricao,Instituicao,Cidade)),
          S
          ).

% Predicado identificar_servico_por_Cidade:
% Cidade, Lista de Servicos -> {V,F}
identificar_servico_por_cidade(Cidade,S):-
  solucoes(
          (Id,Descricao,Instituicao),
          (servico(Id,Descricao,Instituicao,Cidade)),
          S
          ).
```

Figura 13 - Identificação de serviços por critérios de seleção.

3.3.2.3 Consultas

Já no caso das consultas, estas podem ser caracterizadas pelo seu utente, serviço, data ou custo. De forma análoga aos casos anteriores, é necessária a utilização do predicado *solucoes* (Figura 14).

```
% Predicado identificar_consulta_por_Utente:
% Id, Lista de Consultas -> {V,F}
identificar_consulta_por_utente(IdUt,S):-
  solucoes(
          (Data,IdServ,Custo),
          (consulta(Data,IdUt,IdServ,Custo)),
          S
          ).

% Predicado identificar_consulta_por_Servico:
% Id, Lista de Consultas -> {V,F}
identificar_consulta_por_servico(IdServ,S):-
  solucoes(
          (Data,IdUt,Custo),
          (consulta(Data,IdUt,IdServ,Custo)),
          S
          ).

% Predicado identificar_consulta_por_Data:
% Data, Lista de Consultas -> {V,F}
identificar_consulta_por_data(Data,S):-
  solucoes(
          (IdUt,IdServ,Custo),
          (consulta(Data,IdUt,IdServ,Custo)),
          S
          ).

% Predicado identificar_consulta_por_Custo:
% Custo, Lista de Consultas -> {V,F}
identificar_consulta_por_custo(Custo,S):-
  solucoes(
          (Data,IdUt,IdServ),
          (consulta(Data,IdUt,IdServ,Custo)),
          S
          ).
```

Figura 14 - Identificação de consultas por critérios de seleção.

3.3.3 Identificar serviços prestados por instituição/cidade/datas/-custo

Para uma dada instituição ou cidade, é possível apresentar os serviços relacionados com as mesmas, utilizando, mais uma vez, o predicado *solucoes*, como se observa na Figura 15.

Figura 15 - Identificação de serviços por cidade ou instituição.

Já no caso de identificar as consultas prestadas pelos parâmetros requeridos (Figura 16), a linha de pensamento foi semelhante, diferindo, ligeiramente, no caso de encontrar as consultas num intervalo de datas. Neste caso, foi necessário definir um predicado auxiliar que determina se a data da consulta em análise é aceitável, a qual, aliada ao *solucoes*, permite determinar todas as consultas no intervalo (Figura 17).

```
% Predicado consultas_por_instituicao:
% Instituicao, Lista de Consultas -> {V,F}
consultas_por_instituicao(Instituicao,S) :-
    solucoes(
            (Data,IdUt,IdServ,Preco,Descricao),
            (
                consulta(Data,IdUt,IdServ,Preco),
                servico(IdServ,Descricao,Instituicao,_)
            ),
            S
        ).

% Predicado consultas_por_cidade:
% Cidade, Lista de Consultas -> {V,F}
consultas_por_cidade(Cidade,S) :-
    solucoes(
            (Data,IdUt,IdServ,Preco,Descricao),
            (
                consulta(Data,IdUt,IdServ,Preco),
                servico(IdServ,Descricao,Instituicao,Cidade)
            ),
            S
        ).

% Predicado consultas_por_data:
% Data, Lista de Consultas -> {V,F}
consultas_por_data(Data,S) :-
    solucoes(
            (IdUt,IdServ,Preco,Descricao),
            (
                consulta(Data,IdUt,IdServ,Preco),
                servico(IdServ,Descricao,Instituicao,Cidade)
            ),
            S
        ).

% Predicado consultas_por_preco:
% Preco, Lista de Consultas -> {V,F}
consultas_por_preco(Preco,S) :-
    solucoes(
            (Data,IdUt,IdServ,Descricao),
            (
                consulta(Data,IdUt,IdServ,Preco),
                servico(IdServ,Descricao,Instituicao,Cidade)
            ),
            S
        ).
```

Figura 16 - Identificação de consultas por parâmetros.

```
% Predicado consultas_por_intervalo:
% Data, Data, Lista de Consultas -> {V,F}
consultas_por_intervalo(DataInicio,DataFim,L) :-
    solucoes(
            (Data, IdUt, IdServ, Custo),
            (
                consulta(Data, IdUt, IdServ, Custo),
                compDate(Data,DataInicio),
                compDate(DataFim,Data)
            ),
            L
        ).
```

Figura 17 - Identificação de consultas por intervalo de tempo.

3.3.4 Identificar os utentes de um serviço/instituição

Nos predicados apresentados, é efetuada uma pesquisa sobre os conhecimentos relativos a utentes, serviços e consultas, onde é necessário encontrar as consultas com o serviço pretendido, bem como os utentes a elas associados. Este predicado encontra-se representado na Figura 18. O resultado final corresponde a uma lista com todos os utentes de um dado serviço ou instituição, sendo que as repetições não são apresentadas graças à ordenação da lista.

```
% Predicado utentes_de_servico:
% Servico, Lista de Utentes -> {V,F}
utentes_de_servico(IdServ,S) :-
    solucoes(
            (IdUt,Nome),
            (
                utente(IdUt,Nome,Idade,Local),
                servico(IdServ,_,_,_),
                consulta(_,IdUt,IdServ,_)
            ),
            L
        ),
    sort(L,S).

% Predicado utentes_de_instituicao:
% Instituicao, Lista de Utentes -> {V,F}
utentes_de_instituicao(Instituicao,S) :-
    solucoes(
            (IdUt,Nome),
            (
                utente(IdUt,Nome,Idade,Local),
                servico(IdServ,_,Instituicao,_),
                consulta(_,IdUt,IdServ,_)
            ),
            L
        ),
    sort(L,S).
```

Figura 18 - Identificação de utentes de um serviço/instituição.

3.3.5 Identificar serviços realizados por utente/instituição/cidade

Nos predicados apresentados na Figura 19, são usadas as bases de conhecimento relativas a utentes, serviços e consultas com o objetivo de apresentar a listagem de serviços realizados por utente, instituição e cidade, respetivamente. É, mais uma vez, utilizado o predicado *solucoes* como meio para encontrar os valores pretendidos, relacionando os três predicados principais do projeto.

```
% Predicado servicos_realizados_por_utente:
% Utente, Lista de Servico -> {V,F}
servicos_realizados_por_utente(IdUt,S) :-
    solucoes(
            (Data,Descricao,Preco),
            (
                utente(IdUt,Nome,Idade,Local),
                servico(IdServ,Descricao,Instituicao,Cidade),
                consulta(Data,IdUt,IdServ,Preco)
            ),
            S
        ).

% Predicado servicos_realizados_por_instituicao:
% Instituicao, Lista de Servico -> {V,F}
servicos_realizados_por_instituicao(Instituicao,S) :-
    solucoes(
            (Data,Descricao,Preco,Cidade),
            (
                utente(IdUt,Nome,Idade,Local),
                servico(IdServ,Descricao,Instituicao,Cidade),
                consulta(Data,IdUt,IdServ,Preco)
            ),
            S
        ).

% Predicado servicos_realizados_por_cidade:
% Cidade, Lista de Servico -> {V,F}
servicos_realizados_por_cidade(Cidade,S) :-
    solucoes(
            (Data,Descricao,Preco,Instituicao),
            (
                utente(IdUt,Nome,Idade,Local),
                servico(IdServ,Descricao,Instituicao,Cidade),
                consulta(Data,IdUt,IdServ,Preco)
            ),
            S
        ).
```

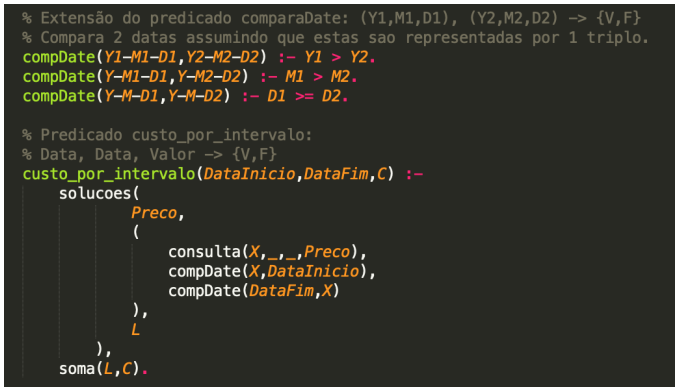
Figura 19 - Identificação de serviços realizados por utente/instituição/cidade.

3.3.6 Calcular o custo total dos cuidados de saúde por utente/-serviço/instituição/data

O custo total dos cuidados de saúde é obtido utilizando o predicado consulta, já que este possui um custo associado a si mesmo. Estes valores são obtidos com auxílio do predicado *solucoes*. No final, estes valores serão todos somados, devolvendo o valor pretendido, como se vê na Figura 20.

Figura 20 - Cálculo do custo total dos cuidados de saúde por utente/serviço/instituição/-data.

No caso de se pretender conhecer o custo total dos cuidados de saúde num determinado intervalo temporal, o método tem de saber quais as consultas com uma data dentro do intervalo, daí o uso de um predicado auxiliar, como se observa na Figura 21.



```
% Extensão do predicado comparaDate: (Y1,M1,D1), (Y2,M2,D2) -> {V,F}
% Compara 2 datas assumindo que estas sao representadas por 1 triplo.
compDate(Y1-M1-D1,Y2-M2-D2) :- Y1 > Y2.
compDate(Y-M1-D1,Y-M2-D2) :- M1 > M2.
compDate(Y-M-D1,Y-M-D2) :- D1 >= D2.

% Predicado custo_por_intervalo:
% Data, Data, Valor -> {V,F}
custo_por_intervalo(DataInicio,DataFim,C) :-
    solucoes(
            Preco,
            (
                consulta(X,_,_,Preco),
                compDate(X,DataInicio),
                compDate(DataFim,X)
            ),
            L
        ),
    soma(L,C).
```

Figura 21 - Cálculo do custo total dos cuidados de saúde num intervalo.

4 Extras

Com o intuito de bonificar a avaliação deste projeto, o grupo decidiu que seria interessante acrescentar mais algumas funcionalidades, comparativamente às propostas pelo enunciado.

Assim sendo, o grupo decidiu adicionar o predicado sexo ao projeto, de modo a torná-lo o mais realista possível. Foram também adicionados alguns predicados que tiram proveito desta nova característica dos utentes.

Todas as funcionalidades extras encontram-se exemplificadas nas figuras seguintes.

De modo a inserir o conhecimento necessário sobre o sexo dos utentes, foi construído um predicado referente ao mesmo (Figura 22). A utilização deste predicado permite obter respostas sobre as mais variadas interrogações.

```
% Predicado sexo: IdUt, Sexo -> {V,F}
sexo(1,m).
sexo(2,m).
sexo(3,m).
sexo(4,m).
sexo(5,m).
sexo(6,f).
sexo(7,f).
sexo(8,f).
```

Figura 22 - Inserção de conhecimento relativo ao Sexo.

Com a inserção do predicado sexo, foi necessário construir invariantes (Figura 23), que garantem que este novo predicado não interfere com a manutenção do conhecimento já existente.

```
% Invariante Estrutural:  nao permitir a insercao de conhecimento repetido
+sexo(IdUt,Valor) :: (solucoes( (IdUt,Valor), (sexo(IdUt,Valor)),S ),
                      comprimento( S,N ),
                      N==1).

% ----------------------------------------------------------------------------
% Invariante Referencial : não permitir a inserção de informacao sobre o genero de um utente que nao existe
+sexo(IdUt,_) :: (solucoes( (IdUt), (utente(IdUt,_,_,_)),S ),
                      comprimento( S,N ),
                      N==1).

% ----------------------------------------------------------------------------
% Invariante Referencial : não permitir a inserção de informacao sobre o genero de um utente que nao seja valida
+sexo(IdUt,Valor) :: valida_sexo(Valor).

%Predicado valida_sexo:
%genero -> {V,F}
valida_sexo(m).
valida_sexo(f).

% ----------------------------------------------------------------------------
% Invariante Referencial : não permitir a remoção de um utente que possua informação sobre o seu genero
-utente(IdUt,_,_,_) :: (solucoes( (IdUt),(sexo(IdUt,_)),S ),
                      comprimento( S,N ),
                      N==0).
```

Figura 23 - Invariantes sobre o predicado sexo, que garantem a manutenção de conhecimento já existente.

De modo a calcular o preço médio de um determinado serviço, dado o seu *id* e recorrendo uma vez mais à função *solucoes*, procuram-se todas as consultas do respetivo serviço e armazenam-se os vários preços numa lista, para ser posível calcular a sua média posteriormente. O código corresponde a esta funcionalidade é apresentado na Figura 24.

Ainda na Figura 24, encontra-se o predicado que possibilita o cálculo do valor total gasto por género. Tal como o predicado mecionado anteriormente, este é auxiliado pela função *solucoes*. Este, para um dado sexo, e tendo em conta os *ids* dos utentes do respetivo género, recolhe os preços das consultas dos mesmos e faz o seu somatório.

```
%Predicado preco_medio_de_servico:
%ID do servico, media dos precos -> {V,F}
preco_medio_de_servico(IdServ,M):-
    solucoes(
        P,
        (
            consulta(_,_,IdServ,P)
        ),
        S
    ),
    media(S,M).

% -------------------------------------------------------
% Verificar o valor total gasto consoante o genero do utente:

%Predicado valor_gasto_por_genero:
%Genero, Valor -> {V,F}
valor_gasto_por_genero(G,V) :-
    solucoes(
        Preco,
        (
            sexo(IdUt,G),
            consulta(_,IdUt,_,Preco)
            ),
        S),
    soma(S,V).
```

Figura 24 - Predicados para os cálculos da média gasta em consultas de um determinado serviço e do valor total que um determinado género gasta em consultas.

Na Figura 25 encontram-se os predicados para a determinação da idade média dos utentes que têm consultas de um dado serviço ou numa determinada instituição. Ambas possuem um princípio de funcionamento muito semelhante. Recorrem à função *solucoes* e, dado um serviço ou instituição, procura todos os utentes que tiveram consultas desse mesmo serviço ou instituição. Sabendo os seus *ids*, recolhe as respetivas idades e calcula a sua média.

```
% Predicado idade_media_por_servico:
% servico, Idade media -> {V,F}
idade_media_por_servico(Desc,Valor) :-
    solucoes(
        Idade,
        (
            utente(IdUt,_,Idade,_),
            servico(IdServ,Desc,_,_),
            consulta(_,IdUt,IdServ,_)
        ),
        S),
    media(S,Valor).

% ------------------------------------------------------
% Verificar a idade media dos utentes de uma instituicao:

% Predicado idade_media_por_instituicao:
% instituicao, Idade media -> {V,F}
idade_media_por_instituicao(Inst,Valor) :-
    solucoes(
        Idade,
        (
            utente(IdUt,_,Idade,_),
            servico(IdServ,_,Inst,_),
            consulta(_,IdUt,IdServ,_)
        ),
        S),
    media(S,Valor).
```

Figura 25 - Predicados para os cálculos da idade média dos utentes que têm consultas num determinado serviço ou instituição.

5 Conclusões e Sugestões

O desenvolvimento deste projeto permitiu um aumento do conhecimento relativamente ao paradigma da programação em lógica, fortalecendo a noção da sua utilidade na resolução de diversos problemas.

Tendo em conta que todas as funcionalidades propostas foram implementadas com sucesso, o grupo considera este primeiro projeto um sucesso.

Em suma, foi possível solidificar os conhecimentos na linguagem *Prolog* e também compreender a utilidade deste paradigma na resolução de problemas que necessitem de uma abordagem imparcial.

6 Anexos

Anexo I - Demonstração de Predicados

```
| ?- listing(servico).
servico(1, medicina_geral, trofa_saude, braga).
servico(2, cardiologia, cufe, porto).
servico(3, oftalmologia, sao_joao, porto).
servico(4, otorrinolarengologia, particular, viana).
servico(5, ginecologia, trofa_saude, braga).
servico(6, cirurgia_geral, santo_antonio, porto).

yes
| ?- listing(consulta).
consulta(2019-3-6, 1, 1, 14).
consulta(2014-4-6, 1, 3, 22).
consulta(2014-3-7, 3, 5, 32).
consulta(2019-5-12, 5, 2, 9).
consulta(2019-2-26, 2, 5, 10).
consulta(2019-12-29, 3, 2, 15).
consulta(2019-3-7, 1, 1, 20).

yes
| ?- listing(utente).
utente(1, joao_pimentel, 20, esposende).
utente(2, bruno_veloso, 22, ponte_da_barca).
utente(3, jaime_leite, 20, felgueiras).
utente(4, pedro_goncalves, 20, felgueiras).
utente(5, rodolfo_silva, 20, trofa).

yes
| ?- servicos_realizados_por_utente(1,S).
S = [(2019-3-6,medicina_geral,14),(2019-3-7,medicina_geral,20),(2014-4-6,oftalmologia,22)] ?
yes
| ?- servicos_realizados_por_utente(2,S).
S = [(2019-2-26,ginecologia,10)] ?
yes
| ?- servicos_realizados_por_utente(3,S).
S = [(2019-12-29,cardiologia,15),(2014-3-7,ginecologia,32)] ?
yes
| ?- servicos_realizados_por_utente(4,S).
S = [] ?
yes
| ?- servicos_realizados_por_utente(5,S).
S = [(2019-5-12,cardiologia,9)] ?
yes
| ?- servicos_realizados_por_utente(6,S).
S = [] ?
yes
| ?- servicos_realizados_por_utente(7,S).
S = [] ?
yes
| ?-
```

Figura 26 - Teste predicado *serviços_realizados_por_utente*.

```
| ?- consultas_por_intervalo(2014-01-11,2014-05-12,S).
S = [(2014-4-6,1,3,22),(2014-3-7,3,5,32)] ?
yes
| ?- listing(consulta).
consulta(2019-3-6, 1, 1, 14).
consulta(2014-4-6, 1, 3, 22).
consulta(2014-3-7, 3, 5, 32).
consulta(2019-5-12, 5, 2, 9).
consulta(2019-2-26, 2, 5, 10).
consulta(2019-12-29, 3, 2, 15).
consulta(2019-3-7, 1, 1, 20).

yes
| ?-
```

Figura 27 - Teste predicado *consultas_por_intervalo*.

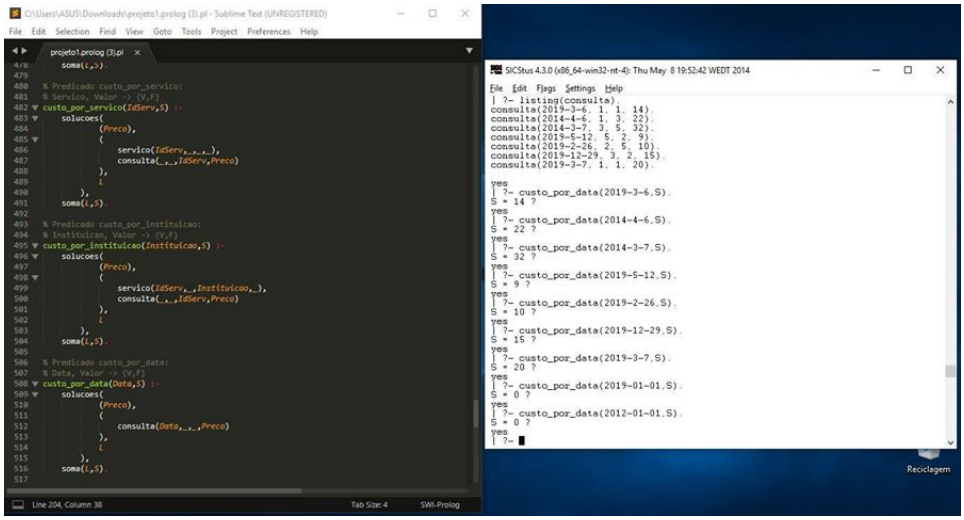


Figura 28 - Teste predicado *custo_por_data*.