Universidade do Minho

Programação em Lógica e Invariantes

Mestrado Integrado em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio

2ºSemestre 2018/19

Grupo:
Etienne Costa A76089
Joana Cruz A76270
Rui Azevedo A80789
Maurício Salgado A71407
João Coutinho A86272

Docente:
César Analide

29 de Março de 2019

1 Resumo

O presente trabalho tem como principal objetivo aprofundar os conhecimentos na linguagem de programação em lógica PROLOG.
Com base nisso foi desenvolvido o relatório explicando o desenvolvimento do primeiro exercício prático no âmbito da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio.
Este trabalho desenvolvido consiste na implementação de um sistema de representação de conhecimento e raciocínio com capacidade para caracterizar um universo de discurso na área da prestação de cuidados de saúde.

Conteúdo

2 Introdução

O relatório apresentado diz respeito ao primeiro exercício proposto no âmbito da unidade curricular de Sistemas de Conhecimento de Representação e Raciocínio, utilizando a linguagem de programação em lógica PROLOG. O universo de discurso para qual estamos a desenvolver este sistema é o universo de prestação de cuidados de saúde, assim esta base de conhecimento consiste em utentes, serviços, consultas, prestadores de serviços, medicamentos e receitas.

3 Preliminares

Para o desenvolvimento deste projeto foi necessário alguns conhecimentos previamente adquiridos de programação em lógica, e a utilização da linguagem PROLOG. Este conhecimento foi absorvido durante as aulas de Sistemas de Representação de Conhecimento e Raciocínio, e também com alguma pesquisa nossa. De seguida, apresentamos alguns conceitos fundamentais para a compreensão e realização deste trabalho.

3.1 Programação em lógica e PROLOG

Uma das principais ideias da programação em lógica é de que um algoritmo é constituído por dois elementos disjuntos: a lógica e o controle. O componente lógico corresponde à definição do que deve ser solucionado, enquanto que o componente de controle estabelece como a solução pode ser obtida. O programador precisa somente descrever o componente lógico de um algoritmo, deixando o controle da execução para ser exercido pelo sistema de programação em lógica utilizado. Em outras palavras, a tarefa do programador passa a ser simplesmente a especificação do problema que deve ser solucionado, razão pela qual as linguagens lógicas podem ser vistas simultaneamente como linguagens para especificação formal e linguagens para a programação de computadores. O paradigma fundamental da programação em lógica é o da programação declarativa, em oposição à programação procedimental típica das linguagens convencionais. Um programa em lógica é então a representação de determinado problema ou situação expressa através de um conjunto finito de um tipo especial de sentenças lógicas denominadas cláusulas. Pode-se então expressar conhecimento (programas e/ou dados) em Prolog por meio de cláusulas de dois tipos: fatos e regras. Um fato denota uma verdade incondicional, enquanto que as regras definem as condições que devem ser satisfeitas para que uma certa declaração seja considerada verdadeira. Como fatos e regras podem ser utilizados conjuntamente, nenhum componente dedutivo adicional precisa ser utilizado. Além disso, como regras recursivas e não-determinismo são permitidos, os programadores podem obter descrições muito claras, concisas e não-redundantes da informação que desejam representar. Como não há distinção entre argumentos de entrada e de saída, qualquer combinação de argumentos pode ser empregada. Os termos "programação em lógica"e "programação Prolog"tendem a ser empregados indistintamente. Deve-se, entretanto, destacar que a linguagem Prolog é apenas uma particular abordagem da programação em lógica.

4 Base de conhecimento

Um programa em Prolog é um conjunto de axiomas e de regras de inferência definindo relações entre objectos que descrevem um dado problema. A este conjunto chama-se normalmente base de conhecimento.
A base de conhecimento do sistema desenvolvido é essencial à representação do conhecimento e raciocínio,tendo em conta o sistema foram desenvolvidas as seguintes entidades:

- utente: #IdUt, Nome, Idade,Sexo, Cidade → {V,F}
- serviço: #IdServ,Especialidade,Instituição,Cidade → {V,F}
- consulta:#IdConsult,#IdUt,#IdPrestador,#IdServ,Descrição,Custo,Data → {V,F}
- prestador:#IdPrestador,#IdServ,Nome,Idade,Sexo → {V,F}
- medicamento:#IdMed,Nome,Custo → {V,F}
- receita:#IdCons,#IdMed,DataValidade,Quantidade → {V,F}

4.1 Análise conceptual do problema

Depois de uma análise ao contexto do problema, foi elaborado um esquema conceptual do mesmo. Dado o enunciado conseguimos identificar as seguintes entidades Utente, Serviço e Consulta. De forma a garantir uma melhor consistência da nossa Base de Conhecimento utilizamos a entidade Prestador de Serviço, e a aumentar este sistema cujo universo é o de prestação de cuidados de saúde identificamos as entidades Receita e Medicamento.

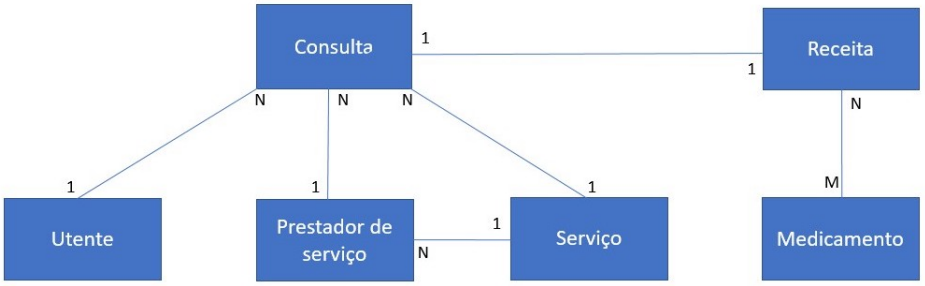

Figura 1: Esquema conceptual representativo do problema

Tentamos abordar este problema o mais perto de contextos reais, sendo que em todos os hospitais e clinicas existem consultas, que são sempre referentes a um utente, um prestador de serviço e a um serviço.

4.2 Entidades

Nesta secção são apresentadas e caraterizadas as Entidades acima propostas.

4.2.1 Utente

Num mundo real o utente é caracterizado por diversos factores , que por sua vez no contexto do Prolog esses factores passam a ser denomindados por átomos cujo o seu propósito é identifcar os objectos. Sendo assim para os utentes optou-se por usar os seguintes átomos:

- IdUt: Identificador do utente,que por sua vez é um valor único.
- Nome: Nome do utente.
- Idade: Idade do utente.
- Sexo: Sexo do utente.
- Cidade: Cidade aonde mora o utente.

4.2.2 Serviço

Um serviço por sua vez é caracterizado pelos seguintes átomos:

- IdServ: Identificador do serviço, que por sua vez é uma valor único.
- Especialidade: Referente ao serviço prestado.
- Instituição: Espaço físico aonde é efectuado o serviço.
- Cidade: Espaço geográfico aonde se encontra a instituição.

4.2.3 Consulta

Uma consulta será sempre referente a um utente, a um prestador de serviço e a um serviço, sendo que a mesma é realizada numa data específica e tem sempre uma breve descrição e custo associado , portanto optou-se por representar o conhecimento do seguinte modo:

- IdConsult: Identificador da consulta, que por sua vez é uma valor único.
- IdUt: Identificador do utente.
- IdPrestador: Identificador do prestador de serviço.
- IdServ: Identificador do serviço prestado.
- Descrição:Breve descrição da consulta.
- Custo: Custo associado à consulta.
- Data: Data da realização da consulta.

4.2.4 Prestador

O prestador por sua vez é a entidade que presta os serviços existentes na base de conhecimento e para tal o mesmo é caracterizado pelos seguintes átomos:

- IdPrestador: Identificador do prestador, que por sua vez é uma valor único.
- IdServ: Identificador do serviço prestado.
- Nome: Nome do prestador de serviço.
- Idade: Idade do prestador.
- Sexo: Sexo do prestador.

4.2.5 Medicamento

Sendo que estamos presente a um sistema de prestação de cuidados de saúde,sentimo-nos na obrigação de fazer a inserção dos medicamentos na base de conhecimento visto que na maioria das vezes após o término de uma consulta é passada uma receita que é constituída por diversos medicamentos. Com base nisso os medicamentos são caracterizados por :

- IdMed: Identificador do medicamento, que por sua vez é uma valor único.
- Nome: Nome do medicamento.
- Custo: Custo do medicamento.

4.2.6 Receita

A receita é uma rotina de cuidados com a saúde, implementadas pelo prestador, voltadas para um utente em específico. Sendo assim a receita é caracterizada por :

- IdCons: Identificador da consulta, que por sua vez é uma valor único.
- IdMed: Identificador do medicamento.
- DataValidade: Corresponde a data de validade de uma receita.
- Quantidade: Corresponde a quantidade de um medicamento pertencente à receita.

5 Integridade da Base de Conhecimento

De forma a manter a integridade da base do conhecimento, e esta esteja de acordo com a realidade que pretendemos representar,é necessário implementar algum mecanismo que nos garanta isso. Assim, ao longo do trabalho fomos dando uso ao conceito de invariante. Estes permitem-nos controlar em específico a correta inserção e remoção na Base de Conhecimento. Os invariantes em PROLOG são representados da seguinte forma:

- +Termo :: Premissas.
- -Termo :: Premissas.

Em Prolog, já existem predicados que nos permitem inserir e remover factos da Base de Conhecimento. Estes são o assert e o retract, respetivamente. No entanto, a utilização destes predicados, exclusivamente, não garante consistência. Sendo assim, é necessário a criação de predicados auxiliares que nos garantam a intregidade da Base de Conhecimento.

5.1 Inserção de Conhecimento

Como já havia sido dito anteriormente o uso exclusivo da função assert não garante a integridade, para tal surgiu um conjunto de condições que são verificadas de modo a presevar a consistência da base de conhecimento, e só assim inserir conhecimento. Este processo foi denominado por evolução. É Implementado do seguinte modo:

```
evolucao(T) :-
findall( I,+T::I,Li ),
insercao( T ),
teste( Li ).

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

5.2 Remoção de conhecimento

Para a remoção do conhecimento o processo foi análogo ao de inserção, só que para este caso concreto o uso exclusivo do retract não garante a integridade, para tal surgiu um conjunto de condições de modo a preservar a consistência da base de conhecimento, e só assim remover conhecimento. Este processo foi denominado por involução.É implementado do seguinte modo:

```
involucao(T) :- T,
findall(I,-T::I,Li),
remocao(T),
teste(Li).

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

5.3 Invariantes Estruturais

Os invariantes estruturais são, tal como o nome indica, responsáveis por manter a estrutura do conhecimento existente. Foi necessário garantir que as inserções não pudessem introduzir conhecimento repetido e que as remoções não pudessem retirar conhecimento não existente ou que estaria associado a outros.

5.3.1 Invariantes de inserção

Este processo é implementado de forma similar em todos os casos referentes aos invariantes de inserção.
Este invariante permite garantir que o identificador de cada utente é único e do tipo inteiro.

```
+utente(IdUt,_,_,_,_) ::
(integer(IdUt),
findall(IdUt,utente(IdUt,Nome,Idade,Sexo,Cidade),S),
length(S,L),
L == 1).
```

Este invariante permite garantir que o identificador de cada serviço é único e do tipo inteiro.

```
+servico(IdServ,_,_,_) ::
(integer(IdServ),
findall(IdServ,servico(IdServ,_,_,_),S),
length(S,L),L == 1).
```

Sendo que a consulta envolve as diferentes chaves primárias da nossa base de conhecimento, houve o cuidado de garantir que o Idconsult é único e do tipo inteiro,e não só,para ser registada uma consulta faz sentido garantir que existe pelo a ocorrência do respectivo utente,prestador e o serviço prestado,sendo esses representados no invariante através dos respectivos identificadores.

```
+consulta(IdConsult,IdUt,IdPrestador,IdServ,_,_,_)::
(integer(IdConsult),
utente(IdUt,_,_,_,_),
prestador(IdPrestador,IdServ,_,_,_),
servico(IdServ,_,_,_),
findall(IdConsult,consulta(IdConsult,_,_,_,_,_,_),S),
length(S,N),
N==1).
```

O invariante de inserção do prestador segue a mesma lógica que o invariante do utente, sendo que este tem um caso particular que corresponde ao serviço na qual é especializado,ou seja, para efectuar a inserção de um prestador o serviço no qual é especializado deve existir.

```
+prestador(IdPrest,IdServ,_,_,_) ::
(integer(IdPrest),
servico(IdServ,_,_,_),
findall(IdPrest,prestador(IdPrest,_,_,_,_),S),
length(S,L) , L== 1).
```

Este invariante permite garantir que o identificador de cada medicamento é único e do tipo inteiro.

```
+medicamento(IdMed,_,_) ::
(integer(IdMed),
findall(IdMed,medicamento(IdMed,_,_),S),
length(S,L) , L == 1).
```

Relativamente a receita usa-se a combinação de dois identificadores para verificar se ocorreu a respectiva consulta e o identificar do medicamento para confirmar que o mesmo existe na base de conhecimento.

```
+receita(IdCons,IdMed,_,_) ::
(integer(IdCons) ,
integer(IdMed),
consulta(IdCons,_,_,_,_,_,_),
medicamento(IdMed,_,_),
findall((IdCons,IdMed),receita(IdCons,IdMed,_,_),S),
length(S,L) , L == 1).
```

5.3.2 Invariantes de remoção

Quanto aos invariantes de remoção seguem todos a mesma ideologia,sendo que só faz sentido fazer uma remoção da base de conhecimento de algo existente na mesma.

```
-utente(IdUt,_,_,_,_) ::
(findall(IdUt,utente(IdUt,_,_,_,_),S),
length(S,N),
N==0).

-servico(IdServ,_,_,_) ::
(findall(IdServ,servico(IdServ,_,_,_),S),
length(S,N),
N==0).

-prestador(IdPrest,_,_,_,_,_) ::
(integer(IdPrest),
findall(IdPrest,prestador(IdPrest,_,_,_,_,_),S),
length(S,L),
L == 0).

-medicamento(IdMed,_,_) ::
(integer(IdMed),
findall(IdMed,medicamento(IdMed,_,_),S),
length(S,L),
L == 0).
```

5.4 Invariantes Referenciais

Os invariantes referenciais evitam que as regras lógicas associadas ao domínio de conhecimento sejam quebradas, permitindo manter a integridade dos dados.

Relativamente ao utente e o prestador foi necessário garantir que a idade dos mesmos fosse maior ou igual que zero e que o sexo seja 'M' ou 'F',sendo que para a validação do sexo foi utilizado um predicado auxiliar validaSexo.

```
+utente(_,_,Idade,Sexo,_) ::
(integer(Idade),
Idade >= 0,
validaSexo(Sexo)).

+prestador(_,_,_,Idade,Sexo) ::
(integer(Idade),
Idade >=0,
validaSexo(Sexo)).
```

No que concerne a consulta foi importante garantir que o seu custo é superior ou igual à 0 u.m e que a sua data seja válida tirando partido do predicado auxilar validaData.

```
+consulta(_,_,_,_,_,Custo,Data) ::
(validaData(Data),
Custo>=0).
```

Quanto aos medicamentos tivemos que garantir que o seu custo fosse sempre superior à 0 u.m.

```
+medicamento(_,_,Custo) ::
(Custo > 0).
```

Relativamente a receita procuramos inserir uma data de modo a enquadar-se num contexto real , visto que as mesmas possuem um prazo e para tal essa data deve ser válida, e a quantidade de cada medicamento receitada não pode ser superior à 5 unidades

```
+receita(_,_,DataValidade,Quantidade) ::
(validaData(DataValidade),
Quantidade > 0).
```

6 Funcionalidades

6.1 Registar utentes,serviços,consultas,prestadores,medicamentos e receitas

```
% Registo de utentes
registarUtente(IdUt,Nome,Idade,Sexo,Cidade) :-
    evolucao(utente(IdUt,Nome,Idade,Sexo,Cidade)).

% Registo de serviços
registarServico(IdServ,Especialidade,Instituicao, Cidade) :-
    evolucao(servico(IdServ,Especialidade,Instituicao, Cidade)).

% Registo de consultas
registarConsulta(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo,Data) :-
    evolucao(consulta(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo,Data)).

% Registo de prestador
registarPrestador(IdPrestador,IdServ,Nome,Idade,Sexo) :-
    evolucao(prestador(IdPrestador,IdServ,Nome,Idade,Sexo)).

% Registo de medicamento
registarMedicamento(IdMed,Nome,Custo) :-
    evolucao(medicamento(IdMed,Nome,Custo)).

% Registo de receita
registarReceita(IdCons,IdMed,DataValidade,Quantidade) :-
    evolucao(receita(IdCons,IdMed,DataValidade,Quantidade)).
```

Figura 2: Registo dos principais intervenientes do sistema

6.2 Remover utentes,serviços,consultas,prestadores,medicamentos e receitas

```
% Remover utente
  removerUtente(IdUt) :- involucao(utente(IdUt,_,_,_,_)).

% Remover servico
  removerServico(IdServ):- involucao(servico(IdServ,_,_,_)).

% Remover consulta
  removerConsulta(IdConsult) :- involucao(consulta(IdConsult,_,_,_,_,_,_)).

% Remover prestador
  removerPrestador(IdPrestador) :- involucao(prestador(IdPrestador,_,_,_,_)).

% Remover medicamento
  removerMedicamento(IdMed) :- involucao(medicamento(IdMed,_,_)).

% Remover receita
  removerReceita(IdCons,IdMed) :- involucao(receita(IdCons,IdMed,_,_)).
```

Figura 3: Remocão dos principais intervenientes do sistema

6.3 Identificação das instituições prestadoras de serviços

```
%- Extensao do predicado listarInstituicoes: R ~ { V, F }
listarInstituicoes(R) :- findall(Instituicao,servico(_,_,Instituicao,_),S),
                         sort(S,R).
```

Figura 4: Listar Instituições

6.4 Identificação de utentes por critérios de seleção

```
%--- Identificar utentes atraves de diferentes criterios ------------------------%

%- Extensao do predicado utenteByID: #IdUt , R ~ { V, F }
utenteByID(IdUt,R) :- findall((IdUt,Nome,Idade,Sexo,Cidade),utente(IdUt,Nome,Idade,Sexo,Cidade), R).

%- Extensao do predicado utenteByNome: Nome , R ~ { V, F }
utenteByNome(Nome,R) :- findall((IdUt,Nome,Idade,Sexo,Cidade), utente(IdUt,Nome,Idade,Sexo,Cidade), R).

%- Extensao do predicado utenteByIdade: Idade , R ~ { V, F }
utenteByIdade(Idade,R) :- findall((IdUt, Nome, Idade, Sexo, Cidade), utente(IdUt,Nome,Idade,Sexo,Cidade), R).

%- Extensao do predicado utenteBySexo: Sexo , R ~ { V, F }
utenteBySexo(Sexo,R) :- findall((IdUt, Nome, Idade, Sexo, Cidade), utente(IdUt,Nome,Idade,Sexo,Cidade), R).

%- Extensao do predicado utenteByCidade: Cidade , R ~ { V, F }
utenteByCidade(Cidade,R) :- findall((IdUt, Nome, Idade, Sexo, Cidade), utente(IdUt,Nome,Idade,Sexo,Cidade), R).
```

Figura 5: Identificação de utentes

6.5 Identificação de prestadores por critérios de seleção

```
%--- Identificar prestador atraves de diferentes criterios ----------------------%

%- Extensao do predicado prestadorByID: #IdPrestador , R ~ { V, F }
prestadorByID(IdPrestador,R) :- findall((IdPrestador,IdServ,Nome,Idade,Sexo),prestador(IdPrestador,IdServ,Nome,Idade,Sexo),R).

%- Extensao do predicado prestadorByIdServ: #IdServ , R ~ { V, F }
prestadorByIdServ(IdServ,R) :- findall((IdPrestador,IdServ,Nome,Idade,Sexo),prestador(IdPrestador,IdServ,Nome,Idade,Sexo),R).

%- Extensao do predicado prestadorByNome: Nome , R ~ { V, F }
prestadorByNome(Nome,R) :- findall((IdPrestador,IdServ,Nome,Idade,Sexo), prestador(IdPrestador,IdServ,Nome,Idade,Sexo), R).

%- Extensao do predicado prestadorByIdade: Idade , R ~ { V, F }
prestadorByIdade(Idade,R) :- findall((IdPrestador,IdServ, Nome, Idade, Sexo), prestador(IdPrestador,IdServ,Nome,Idade,Sexo), R)

%- Extensao do predicado utenteBySexo: Sexo , R ~ { V, F }
prestadorBySexo(Sexo,R) :- findall((IdPrestador,IdServ,Nome,Idade,Sexo), prestador(IdPrestador,IdServ,Nome,Idade,Sexo), R).
```

Figura 6: Identificação de prestadores

6.6 Identificação de medicamentos por critérios de seleção

```
%--- Identificar medicamentos atraves de diferentes criterios ----------------------%

%- Extensao do predicado medicamentoByID: #IdMed , R ~ { V, F }
medicamentoByID(IdMed,R):-findall((IdMed,Nome,Custo),medicamento(IdMed,Nome,Custo),R).

%- Extensao do predicado medicamentoByNome: Nome , R ~ { V, F }
medicamentoByNome(Nome,R):-findall((IdMed,Nome,Custo),medicamento(IdMed,Nome,Custo),R).

%- Extensao do predicado medicamentoByCusto: Custo , R ~ { V, F }
medicamentoByCusto(Custo,R):-findall((IdMed,Nome,Custo),medicamento(IdMed,Nome,Custo),R).
```

Figura 7: Identificação de medicamentos

6.7 Identificação de receitas por critérios de seleção

```
%--- Identificar receitas atraves de diferentes criterios -------------------------%

%- Extensao do predicado receitaByID: IdMed , R ~ { V, F }
receitaByID(IdConsult,R):-findall((Data,IdConsult,IdMed,Quantidade),receita(IdConsult,IdMed,Data,Quantidade),R).

%- Extensao do predicado receitaByData: Data , R ~ { V, F }
receitaByData(Data,R):-findall((Data,IdConsult,IdMed,Quantidade),receita(IdConsult,IdMed,Data,Quantidade),R).

%- Extensao do predicado receitaByQuantidade: Quantidade , R ~ { V, F }
receitaByQuantidade(Quantidade,R):-findall((Data,IdConsult,IdMed,Quantidade),receita(IdConsult,IdMed,Data,Quantidade),R).
```

Figura 8: Identificação de receitas

6.8 Identificação de serviços por critérios de seleção

```
%--- Identificar servicos atraves de diferentes criterios ---------------------%

%- Extensao do predicado servicoByID: #IdServ , R ~ { V, F }
servicoByID(IdServ,R) :-
        findall((IdServ,Especialidade,Instituicao,Cidade), servico(IdServ,Especialidade,Instituicao,Cidade), R).

%- Extensao do predicado servicoByEspecialidade: Especialidade , R ~ { V, F }
servicoByEspecialidade(Especialidade,R) :-
    findall((IdServ,Especialidade,Instituicao,Cidade), servico(IdServ,Especialidade,Instituicao,Cidade), R).

%- Extensao do predicado servicoByInstituicao: #IdServ , R ~ { V, F }
servicoByInstituicao(Instituicao,R) :-
    findall((IdServ,Especialidade,Instituicao,Cidade), servico(IdServ,Especialidade,Instituicao,Cidade), R).

%- Extensao do predicado servicoByCidade: #IdServ , R ~ { V, F }
servicoByCidade(Cidade,R) :-
    findall((IdServ,Especialidade,Instituicao,Cidade), servico(IdServ,Especialidade,Instituicao,Cidade), R).
```

Figura 9: Identificação de serviços

6.9 Identificação de consultas por critérios de seleção

```
%--- Identificar consultas atraves de diferentes criterios ----------------------%

%- Extensao do predicado consultaByIdConsulta: #IdConsult , R ~ { V, F }
consultaByIdConsulta(IdConsult,R) :- findall( (IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo,Data),
                                     consulta(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo,Data),R).

%- Extensao do predicado consultaByIdUtente: #IdUt , R ~ { V, F }
consultaByIdUtente(IdUt,R) :- findall((IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo,Data),
                              consulta(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo,Data), R).

%- Extensao do predicado consultaByIdPrestador: #IdPrestador , R ~ { V, F }
consultaByIdPrestador(IdPrestador,R) :- findall( (IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo,Data),
                                        consulta(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo,Data),R).

%- Extensao do predicado consultaByIdServ: #IdServ , R ~ { V, F }
consultaByIdServ(IdServ,R) :- findall( (IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo,Data),
                              consulta(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo,Data),R).

%- Extensao do predicado consultaByData: Data , R ~ { V, F }
consultaByData(Data,R) :- findall( (IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo),
                          consulta(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo,Data),R).

%- Extensao do predicado consultaByCusto: Custo , R ~ { V, F }
consultaByCusto(Custo,R) :- findall( (IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo),
                            consulta(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo,Data),R).
```

Figura 10: Identificação de consultas

6.10 Identificação de serviços prestados por instituição/cidade

```
%--- Identificar servicos prestados por Instituicao ----------------------------%
servicoPerInstituicao(Instituicao,R):-findall((Especialidade,Cidade),servico(_,Especialidade,Instituicao,Cidade),R

%--- Identificar servicos prestados por Cidade ----------------------------%
servicoPerCidade(Cidade,R):-findall((Instituicao,Especialidade),servico(_,Especialidade,Instituicao,Cidade),R).
```

Figura 11: Identificação de serviços por instituição e cidade

6.11 Identificação de utentes de um serviço/instituição

```
%--- Identificar os utentes de um servico prestado ----------------------------%
searchUtentePerService(Especialidade,R):-findall(IdServ,servico(IdServ,Especialidade,Instituicao,_),S),
                                         auxserUt(S,K),
                                        infoUtente(K,R).

%--- Identificar os utentes que frequentaram uma Instituicao ------------------%
searchUtentePerInstituicao(Instituicao,R):- findall(IdServ,servico(IdServ,_,Instituicao,_),S),
                                            auxserUt(S,K),
                                            infoUtente(K,R).
```

Figura 12: Identificação de utentes que frequentaram um serviço ou instituição

6.12 Identificação de serviços realizados por utente/instituição/data/custo

```
%--- Identificar servicos realizados por utente/instituicao/datas/custo

consultaUtente(IdUt,R) :- findall(((Data,IdPrestador,IdServ,Custo)),consulta(_,IdUt,IdPrestador,IdServ,_,Custo,Data),S) ,
                          map(S,R).

consultaData(Data,R) :- findall(((Data,IdPrestador,IdServ,Custo)),consulta(_,IdUt,IdPrestador,IdServ,_,Custo,Data),S) ,
                        map(S,R).

consultaCusto(Custo,R) :- findall(((Data,IdPrestador,IdServ,Custo)),consulta(_,IdUt,IdPrestador,IdServ,_,Custo,Data),S) ,
                          map(S,R).

consultaByInst(Instituicao,R) :- findall(IdServ, servico(IdServ,_,Instituicao,_), LS),
                                 auxInst(LS, S),
                                 map(S,R).
```

Figura 13: Consultas concretizadas

6.13 Cálculo do custo total dos cuidados de saúde por utente/serviço/instituição/data

```
%--- Calcular o custo total dos cuidados de saude por utente / servico / instituicao / data

custoByUtente(IdUt,R) :- findall(Custo,consulta(_,IdUt,_,_,_,Custo,_),C) , sumlist(C,R).

custoByServico(IdServ,R) :- findall(Custo,consulta(_,_,_,IdServ,_,Custo,_),C) , sumlist(C,R).

custoByInstituicao(Instituicao,R) :-findall(IdServ,servico(IdServ,_,Instituicao,_),S),
                                   auxiliar(S,T),
                                   sumlist(T,R).

custoByData(Data,R) :- findall(Custo,consulta(_,_,_,_,_,_,Data),C),sumlist(C,R).
```

Figura 14: Custos

7 Extras

De forma voluntária decidimos acrescentar novas funcionalidades de acordo com a nossa base de conhecimento,isto é, invariantes referenciais e estruturais sobre os predicados prestador,receita e medicamentos estando os mesmos referenciados acima. De forma a explorar informação sobre os mesmos foram implementadas as seguintes funcionalidades:

- getCustoReceita : Devolve o custo total de uma receita.

```
getCustoReceita(IdCons, R) :-
findall((IdMed,Quantidade),receita(IdCons,IdMed,_,Quantidade),S),
getCustoMedicamento(S,R).
```

- getPrestadores : Dado o identificador de um utente, retorna a lista dos prestadores de serviços que o utente frequentou e as suas respectivas datas.

```
getPrestadores(IdUt , R) :-
findall((IdPrest,Data),consulta(_,IdUt,IdPrest,_,_,_,Data),S1) ,
auxiliarPrestador(S1,R).
```

- getMedicamentosByReceita: Dado o identificador da consulta, lista os medicamentos que fazem parte desta receita.

```
getMedicamentosByReceita(IdConsult, R) :-
findall(IdMed,receita(IdConsult, IdMed,_,_), S),
auxiliarMedicamentos(S,R).
```

- getNumeroMedicosByInst: Devolve o número de prestadores de uma dada especialidade em uma dada instiuição.

```
getNumeroMedicosByInst(Especialidade, Instituicao, R) :-
findall(IdServ,servico(IdServ,Especialidade,Instituicao,_),S1) ,
head(S1,H),
findall(IdPrest,prestador(IdPrest,H,_,_,_), S2),
length(S2,R).
```

- despesaAnualConsultas: Devolve o total gasto por um utente em um determinado ano.

```
despesaAnualConsultas(IdUt, Ano, R) :-
findall(Custo,consulta(_,IdUt,_,_,_,Custo,(Ano,_,_)), S),
sumlist(S,R).
```

- medicamentoMaisCaro : Devolve o medicamento mais caro.

```
medicamentoMaisCaro(R) :-
findall((Custo,Nome), medicamento(_,Nome,Custo),S) ,
sort(S,N) ,
last(N,L),
swap_pair(L,R).
```

- medicamentoMaisBarato : Devolve o medicamento mais barato.

```
medicamentoMaisBarato(R) :-
findall((Custo, Nome), medicamento(_,Nome,Custo),S),
sort(S,N),
head(N,L),
swap_pair(L,R).
```

- freq_esp: Dada uma especialidade devolve a frequencia de utentes dessa especialidade.

```
freq_esp(Especialidade, R) :-
findall(IdServ, servico(IdServ,Especialidade,_,_), L),
auxFreq(L, S),
findall(IdConsult, consulta(IdConsult,_,_,_,_,_,_), A),
length(S,S1), length(A,A1),
R is S1/A1 * 100.
```

- freq_all: Devolve as frequencias de utentes de todas as especialidades.

```
freq_all(R) :-
findall(Especialidade, servico(_,Especialidade,_,_), L),
sort(L, L1),
maplist(freq_esp, L1, L2),
zip(L1, L2, R).
```

Além destas funcionalidades inserimos ainda um Menu, de modo que futuros utilizadores possam tirar partido do sistema desenvolvido sem perceber concretamente como o mesma está implementado.

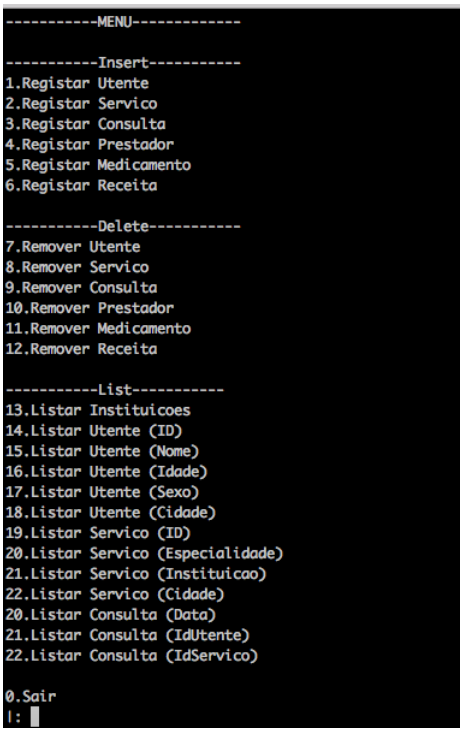

Figura 15: Esquema conceptual representativo do problema

8 Conclusão

A realização deste trabalho prático permitiu consolidar o conhecimento adquirido ao longo das aulas, no que concerne à programação em lógica e invariantes. Como tal, o suporte utilizado para caracterizar um universo de discurso na área da prestação de cuidados de saúde foi o PROLOG. Para o sistema em causa procurou-se fazer sempre uma aproximação ao contexto real do problema sendo que toda base de conhecimento surge à custa de um modelo conceptual, procurando não repetir conhecimento e inserir funcionalidades extras de modo a representar o conhecimento de forma inteligente. Em forma de conclusão, podemos afirmar que fomos capazes de aplicar os conhecimentos lecionados e com êxito implementar as funcionalidades exigidas , sendo que de forma voluntária foram implementadas algumas funcionalidades extras.

9 Referências Bibliográficas

10 Funções Auxiliares

```
%-------------------------------------------------------------------------------
auxiliar([],[]).
auxiliar([IdServ|Tail],R):-
findall(Custo,consulta(_,_,_,IdServ,_,Custo,_),S),
auxiliar(Tail,T),
append(S,T,R).
%-------------------------------------------------------------------------------
auxiliarPrestador([],[]).
auxiliarPrestador([(IdPrest,Ano,Mes,Dia)|Tail],R) :-
findall(Nome,prestador(IdPrest,_,Nome,_,_,_),S),
auxiliarPrestador(Tail,T),
head(S,P) ,
append([P,Ano,Mes,Dia],T,R).
%-------------------------------------------------------------------------------
auxiliarMedicamentos([],[]).
auxiliarMedicamentos([IdMed|Tail],R) :-
findall(Nome,medicamento(IdMed,Nome,_),S),
auxiliarMedicamentos(Tail,T),
append(S,T,R).
%-------------------------------------------------------------------------------
auxInst([],[]).
auxInst([IdServ|Tail], R) :-
findall((Data,IdPrestador,IdServ,Custo),
consulta(_,_,IdPrestador,IdServ,_,Custo,Data), L),
auxInst(Tail,S),
append(L,S,R).
%-------------------------------------------------------------------------------
map([],[]).
map([((Data,IdPrestador,IdServ,Custo))|Tail],R) :-
map(Tail,T) ,
findall(Nome,prestador(IdPrestador,_,Nome,_,_),P) ,
findall(Especialidade,servico(IdServ,Especialidade,_,_),S),
findall(Instituicao,servico(IdServ,_,Instituicao,_),Q),
head(P,PR) , head(S,SR) ,head(Q,INS),
append([(Data,PR,SR,Custo,INS)],T,R).
%-------------------------------------------------------------------------------
zip([],[],[]).
zip([X|XS],[Y|YS], [(X,Y)|Z]) :- zip(XS,YS,Z).

%-------------------------------------------------------------------------------
infoUtente([],[]).
infoUtente([IdUt|Tail],R):-findall((IdUt,Nome,Idade,Sexo,Cidade),
utente(IdUt,Nome,Idade,Sexo,Cidade),S),
infoUtente(Tail,T),
append(S,T,R).
%-------------------------------------------------------------------------------
auxserUt([],[]).
auxserUt([Id|Tail],R):-
findall(IdUt,consulta(_,IdUt,_,Id,_,_,_),S),
auxserUt(Tail,T),
append(S,T,K),
```

```
sort(K,R).
%-------------------------------------------------------------------------------
getCustoMedicamento([],0).
getCustoMedicamento([Head | Tail] , R) :-
getCustoMedicamento(Tail,S1),
factor(Head,S2),
R is S1 + S2.
%-------------------------------------------------------------------------------
factor((IdMed,Quantidade), R) :-
findall(Custo, medicamento(IdMed,_,Custo), S) ,
head(S,S1) ,
R is S1 * Quantidade.

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

auxFreq([],[]).
auxFreq([IdServ|Tail], R) :-
findall(IdConsult, consulta(IdConsult,_,_,IdServ,_,_,_), L),
auxFreq(Tail, S),
append(L,S,R).
%-------------------------------------------------------------------------------

swap_pair((X,Y),(Y,X)).
%-------------------------------------------------------------------------------
```