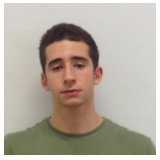
Sistemas de Representação de Conhecimento e Raciocínio
Grupo 31
MIEI

Gonçalo Pereira

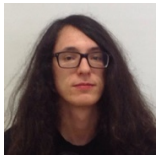
A74413

António Silva



A73827

André Diogo



A75505

19 de Março de 2017

Conteúdo

Capítulo 1

Introdução

O objetivo deste trabalho terá sido o de desenvolvimento de um sistema capaz de caraterizar três tipos de conhecimento, nomeadamente:

- Utente, caracterizado por `#IDUtente, Nome, Idade, Morada`;
- Cuidado prestado, caracterizado por `#IDServiço, Descrição, Instituição, Cidade`;
- Ato médico, caracterizado por `Data, #IDUtente, #IDServiço, Custo`.

Como tal, este sistema deverá respeitar a noção de certos invariantes que o limitam e, ao mesmo tempo, representar virtualmente um universo de prestação de cuidados de saúde através da realização de atos médicos. De igual modo, demonstrar-se-á, em seguida e nas demais páginas, qual a interpretação desenvolvida sobre este problema, exemplificando a forma como este sistema foi construído.

Interpretou-se o enunciado de tal forma que se considerou que um ato médico potencia múltiplos cuidados prestados, mas que estes cuidados prestados tem uma identificação de serviço igual entre eles para o mesmo ato médico e estes cuidados são também únicos ao utente a quem os cuidados foram prestados. Os utentes serão também únicos, identificados pelo seu identificador e o ato médico terá então para cada identificador de serviço apenas um possível utente para esse serviço.

Considerou-se ainda que um ato médico pode ser efetuado parcialmente em múltiplos estabelecimentos, caso uma pessoa seja transferida para outro estabelecimento para prestar cuidados que não são possíveis no primeiro.

Capítulo 2

Predicados envolvidos na evolução e regressão de conhecimento

De forma a seguir a convenção ilustrada nas aulas práticas, decidiu-se utilizar o seguinte excerto para evolução e regressão de conhecimento. Deste modo, foi possível o teste de invariantes, de modo a que verificar a não existência de duplicados nem ambiguidades.

```
evolucao( P ) :-
  insercao(P),
  findall(Inv,+P::Inv,LInv),
  testa(LInv).

regressao( P ) :-
  remocao(P),
  findall(Inv,-P::Inv,LInv),
  testa(LInv).

insercao( P ) :-
  assert(P).

insercao( P ) :-
  retract(P),
  !,
  fail.

remocao( P ) :-
  retract(P).

remocao(P) :-
  assert(P),
  !,
  fail.

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

Capítulo 3

Invariantes

Como enunciado anteriormente, definiram-se certos invariantes para que o conhecimento não aparecesse nem duplicado nem de modo ambíguo nos casos de **adição de conhecimento**, e a não permissão de **remoção de conhecimento** caso existissem outros conhecimentos dependentes daquele.

No caso da adição de conhecimento, definimos os seguintes invariantes:

```
%Apenas uma ocorrência de um ID único para utente
+utente(ID,Nome,Idade,Morada) :: (
  findall(ID,utente(ID,NomeB,IdadeB,MoradaB),S),
  length(S,N),
  N == 1).

%Apenas uma ocorrência de um ID de utente para cada ato médico
%O mesmo cuidado prestado a um utente diferente tem de possuir
%um id único.
+atomedico(Data, IDUtente, IDServ, Preco) :: (
  findall(ID,atomedico(Data, ID, IDServ, Preco),S),
  length(S,N),
  N == 1).

%Ato médico tem IDs existentes na base de Conhecimento
+atomedico(Data, IDUtente, IDServ, Preco) :: (
  findall(IDUtente,utente(IDUtente,Nome,Idade,Morada),S1),
  findall(IDServ,cuidadoprestado(IDServ,Serv,Estab,Local),S2),
  pertence(IDUtente,S1),
  pertence(IDServ,S2)).
```

No caso da remoção de conhecimento, definimos os seguintes invariantes:

```
%Não remover utentes nem cuidadoprestado
%sem remover os atos médicos em que participam
-utente(ID,Nome,Idade,Morada) :: (
  findall(ID,atomedico(Data,ID,IDServ,Preco),S),
  length(S,N),
  N==0).

-cuidadoprestado(ID,Serv,Estab,Local) :: (
  findall(ID,atomedico(Data,IDUtente,ID,Preco),S),
  length(S,N),
  N==0).
```

Capítulo 4

Funcionalidades

Já na **representação de conhecimento**, foram pedidas, como **requisito mínimo**, as seguintes funcionalidades:

- Registar utentes, cuidados prestados e atos médicos;
- Identificar os utentes por critérios de seleção;
- Identificar as instituições prestadoras de cuidados de saúde;
- Identificar os cuidados prestados por instituição/cidade;
- Identificar os utentes de uma instituição/serviço;
- Identificar os atos médicos realizados, por utente/instituição/serviço;
- Determinar todas as instituições/serviços a que um utente já recorreu;
- Calcular o custo total dos atos médicos por utente/serviço/instituição/data;
- Remover utentes, cuidados e atos médicos.

Além das funcionalidades aqui enunciadas, foram também implementadas funcionalidades adicionais, de modo a enriquecer o conhecimento do sistema. Tais funcionalidades são:

- Instituição que gerou maior/menor valor monetário total
- Instituição com maior/menor numero de atos médicos realizados

Assim, e como tal, demonstrar-se-ão tais funcionalidades nas seguintes páginas.

4.1 Registar utentes, cuidados prestados e atos médicos

Para proceder ao registo de utentes, cuidados prestados e atos médicos, recorreu-se ao raciocínio de evolução, explicitado anteriormente, conjugado com a verificação das condições dos invariantes. Como tal, e após verificação dos invariantes, o sistema evoluirá em termos de conhecimento, adicionando um dos três parâmetros pedidos no registo.

```
registarUtente( ID,Nome,Idade,Morada ) :-
  evolucao( utente( ID,Nome,Idade,Morada ) ).

registarCuidados( ID,Serv,Estab,Local ) :-
  evolucao( cuidadoprestado( ID,Serv,Estab,Local ) ).

registarAtos( Data, IDUtente, IDServ, Preco ) :-
  evolucao( atomedico( Data, IDUtente, IDServ, Preco ) ).
```

4.2 Identificar os utentes por critérios de seleção

Para listar todos os utentes por Nome/Idade/Morada, bastou usar o predicado disponibilizado pela linguagem `findall`, fazendo matching com todos os utentes que tenham como parâmetro o elemento recebido. `listaUtentesNome/Idade/Morada` dá como resultado todos os parâmetros desses utentes numa lista.

```
listaUtentesNome(Nome,L):-
  findall([IDUtente,Nome,Idade,Morada],utente(IDUtente,Nome,Idade,Morada),L).

listaUtentesIdade(Idade,L):-
  findall([IDUtente,Nome,Idade,Morada],utente(IDUtente,Nome,Idade,Morada),L).

listaUtentesMorada(Morada,L):-
  findall([IDUtente,Nome,Idade,Morada],utente(IDUtente,Nome,Idade,Morada),L).
```

4.3 Identificar as instituições prestadoras de cuidados de saúde

Para listar todas instituições prestadoras de cuidados de saúde, foi usado o predicado `findall`, fazendo matching com todos os cuidados prestados e acumulando a instituição de cada um numa lista. Foi necessário o uso do predicado `sort` para remover instituições duplicadas da lista.

```
listaInstituicoesCuidados(L):-
  findall(Instituicao,cuidadoprestado( IDServ,Desc,Instituicao,Cidade ),X),
  sort(X,L).
```

4.4 Identificar os cuidados prestados por instituição/cidade

Para listar todos os cuidados prestados por instituição/cidade, recorremos novamente ao predicado `findall`, de modo a acumular todos os cuidados prestados que correspondam aos dados recebidos.

```
listaCuidadosPrestadosInstituicao(Instituicao,L):-
  findall([IDServ,Desc,Instituicao,Cidade],cuidadoprestado( IDServ,Desc,Instituicao,Cidade ),L).

listaCuidadosPrestadosCidade(Cidade,L):-
  findall([IDServ,Desc,Instituicao,Cidade],cuidadoprestado( IDServ,Desc,Instituicao,Cidade ),L).
```

4.5 Identificar os utentes de uma instituição/serviço

Para o predicado `listaUtentesInstituicao`, foi necessário construir um predicado auxiliar `iterarAtosMedicosU` que dada uma lista de IDs de Serviço(gerada novamente pelo predicado `findall`), gera uma outra lista com os utentes que participaram no ato médico referente a cada um dos IDs. Também aqui foi usado o predicado `sort` para remover utentes duplicados.

Para o predicado `listaUtentesServico`, foi também necessário um predicado auxiliar, desta vez para gerar uma lista de utentes com os IDs de Utente da lista gerada.

```
listaUtentesInstituicao(Instituicao,L):-
  findall(IDServ,cuidadoprestado( IDServ,Desc,Instituicao,Cidade ),X),
  iterarAtosMedicosUtentes(X,[],Y),
```

```
  sort(Y,L).

listaUtentesServico(IDServ,L):-
  findall(IDUtente,atomedico(Data,IDUtente,IDServ,Preco),X),
  listaUtentes(X,[],L).
```

4.6 Identificar os atos médicos realizados, por utente/instituição/serviço

Para os predicados `listaAtosMedicosUtente` e `listaAtosMedicosServico`, bastou recorrer ao predicado `findall`), acumulando todos os atos médicos que se refiram ao ID de Utente e ID de Serviço respetivamente.

O predicado `listaAtosMedicosInstituicao` começa por gerar uma lista com os IDs de Serviço dos cuidados prestados que se refiram à Instituição em causa. De seguida, recorre a um predicado auxiliar `iterarAtosMedicos` para, a partir dessa lista, gerar outra com todos os atos médicos que cujo ID de Serviço esteja contido na mesma.

```
listaAtosMedicosUtente(IDUtente,L) :-
  findall([Data,IDUtente,IDServ,Preco],atomedico(Data,IDUtente,IDServ,Preco),L).

listaAtosMedicosInstituicao(Instituicao,L) :-
  findall(IDServ,cuidadoprestado( IDServ,Desc,Instituicao,Cidade ),X),
  iterarAtosMedicos(X,[],L).

listaAtosMedicosServico(IDServ,L) :-
  findall([Data,IDUtente,IDServ,Preco],atomedico(Data,IDUtente,IDServ,Preco),L).
```

4.7 Determinar todas as instituições/serviços a que um utente já recorreu

Para listar as instituições a que um utente recorreu, foi criado o predicado `listaInstituicoesUtente`, que começa por gerar uma lista com todos os IDs de Serviço de atos médicos em que o utente tenha participado. De seguida, recorre a um predicado auxiliar `iterarCuidadosEstab` que encontra todas as instituições que estejam envolvidas num cuidado prestado com os IDs de Serviço da lista anterior. Foi usado novamente o predicado `sort` para remover instituições duplicadas.

Para gerar uma lista com todos os serviços a que um utente recorreu, bastou o uso do predicado `findall` para acumular todos IDs de Serviço dos atos médicos em que o utente participou.

```
listaInstituicoesUtente(IDUtente,L) :-
  findall(IDServ,atomedico(Data,IDUtente,IDServ,Preco),X),
  iterarCuidadosEstab(X,[],Y),
  sort(Y,L).

listaServicosUtente(IDUtente,L) :-
  findall(IDServ,atomedico(Data,IDUtente,IDServ,Preco),L).
```

4.8 Calcular o custo total dos atos médicos por utente/serviço/instituição/data

De modo a calcular o custo total dos atos médicos por utente/serviço/instituição/data, implementou-se um raciocínio que percorrerá uma lista e somará todos os elementos, servindo de contador.

```
acumular([],0).
acumular([H | T],N):-
  acumular(T,S),
  N is S+H.
```

De seguida, percorreu-se todo o conhecimento do tipo pedido (*i.e.: utente*), filtrando-o pelo parâmetro enunciado, de modo a que seja construída uma lista com todos os que obedecem a tal filtro. Por fim, utilizou-se a função definida anteriormente, `acumular`, para que se pudesse obter um valor total de custos.

```
custoTotalUtente( IDUtente,N ) :-
  findall( Custo,atomedico( Data,IDUtente,IDServico,Custo ),S ),
  acumular( S,N ).

custoTotalServico( IDServico,N ) :-
  findall( Custo,atomedico( Data,IDUtente,IDServico,Custo ),S ),
  acumular( S,N ).

custoTotalData( Data,N ) :-
  findall( Custo,atomedico( Data,IDUtente,IDServico,Custo ),S ),
  acumular( S,N ).
```

Porém, e no cálculo de custo total através de uma instituição, este raciocínio provou-se difícil, visto que era necessário iterar sobre duas listas, ao invés de uma. Como tal, foi definido o seguinte raciocínio para que se pudesse iterar sobre a lista de cuidados prestados:

```
iterarAtosMedicosPreco([],I,L):-L=I.
iterarAtosMedicosPreco([H|T],I,L) :-
  findall(Preco,atomedico(Data,IDUtente,H,Preco),Y),
  append(Y,I,Z),
  iterarAtosMedicosPreco(T,Z,L).
```

Para que, por fim, pudesse ser aplicada ao seguinte raciocínio:

```
custoTotalInstituicao( Instituicao,N ) :-
  findall( IDServico,cuidadoprestado( IDServico,Desc,Instituicao,Cidade ),X ),
  iterarAtosMedicosPreco(X,[],S),
  acumular( S,N ).
```

4.9 Remover utentes, cuidados e atos médicos

Tal como na evolução para registar utentes, cuidados prestados e atos médicos, recorreu-se ao raciocínio de regressão, explicitado anteriormente, conjugado com a verificação das condições dos invariantes. A regressão funciona de modo inverso ao da evolução, sendo que removerá conhecimento após verificação dos invariantes de negação.

```
removerutente( ID,Nome,Idade,Morada ) :-
  regressao( utente( ID,Nome,Idade,Morada ) ).

removercuidados( ID,Serv,Estab,Local ) :-
  regressao( cuidadoprestado( ID,Serv,Estab,Local ) ).

removeratos( Data, IDUtente, IDServ, Preco ) :-
  regressao( atomedico( Data, IDUtente, IDServ, Preco ) ).
```

Capítulo 5

Funcionalidades extra

Neste capítulo demonstrar-se-ão as funcionalidades extra implementadas à parte, já fora do enunciado. Tais serão igualmente explicadas e citadas com o raciocínio específico.

5.1 Instituição que gerou maior/menor valor monetário total

Ambos os predicados começam por encontrar todas as instituições que prestaram cuidados através do `findall`, acumulando-as numa lista. Usam de seguida o predicado auxiliar `mapInstituicaoValor`, que cria uma lista com "pares"(são listas de dois elementos) com o primeiro elemento sendo o valor monetário total da instituição e o segundo a própria instituição(note-se o uso da função `custoTotalInstituicao` pedida pelo enunciado). É usado o predicado `sort` e, como o valor monetário é o primeiro elemento de cada "par", a lista fica ordenada por esse valor.

No final, o predicado de menor valor gerado usa o predicado auxiliar `first` para obter o "par"à cabeça e o `last` para extrair a instituição. O predicado de maior valor gerado usa o predicado auxiliar `last` para obter o último "par"e o `last` para extrair instituição.

```
instituicaoMaiorValor(L):-
   findall(Instituicao,cuidadoprestado( IDServico,Desc,Instituicao,Cidade ),X),
   mapInstituicaoValor(X,[],Y),
   sort(Y,Z),
   last(Z,A),
   last(A,L).

instituicaoMenorValor(L):-
   findall(Instituicao,cuidadoprestado( IDServico,Desc,Instituicao,Cidade ),X),
   mapInstituicaoValor(X,[],Y),
   sort(Y,Z),
   first(Z,A),
   last(A,L).

 mapInstituicaoValor([],I,L):-L=I.
 mapInstituicaoValor([H|T],I,L):-
   custoTotalInstituicao(H,C),
   append([[C,H]],I,Z),
   mapInstituicaoValor(T,Z,L).
```

5.2 Instituição com maior/menor numero de atos médicos realizados

Ambos os predicados começam por encontrar todas as instituições que prestaram cuidados através do `findall`, acumulando-as numa lista. Usam de seguida o predicado auxiliar `mapInstituicaoNumAtos`,

que cria uma lista com "pares"(são listas de dois elementos) com o primeiro elemento sendo número total de atos médicos da instituição e o segundo a própria instituição. É usado o predicado `sort` e, como o número de atos é o primeiro elemento de cada "par", a lista fica ordenada por esse valor.

No final, o predicado de menor número de atos usa o predicado auxiliar `first` para obter o "par"à cabeça e o `last` para extrair a instituição. O predicado de maior número de atos usa o predicado auxiliar `last` para obter o último "par"e o `last` para extrair a instituição.

```
instituicaoMaisAtos(L):-
   findall(Instituicao,cuidadoprestado( IDServico,Desc,Instituicao,Cidade ),X),
   mapInstituicaoNumAtos(X,[],Y),
   sort(Y,Z),
   last(Z,A),
   last(A,L).

instituicaoMenosAtos(L):-
   findall(Instituicao,cuidadoprestado( IDServico,Desc,Instituicao,Cidade ),X),
   mapInstituicaoNumAtos(X,[],Y),
   sort(Y,Z),
   first(Z,A),
   last(A,L).

 mapInstituicaoNumAtos([],I,L):-L=I.
 mapInstituicaoNumAtos([H|T],I,L):-
   findall(IDServ,cuidadoprestado(IDServ,Serv,H,Local),X),
   sort(X,Y),
   numElems(Y,C),
   append([[C,H]],I,Z),
mapInstituicaoNumAtos(T,Z,L).
```

Capítulo 6

Conclusão

Com este trabalho registou-se um melhoramento geral das capacidades do grupo de expressar conhecimento de uma forma declarativa através dos mecanismos de extensão à programação lógica aprendidos durante as aulas teóricas e práticas.