Universidade do Minho

Prestação de cuidados de saúde

Mestrado Integrado em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio
(2º Semestre - 17/18)

A77531 Daniel Fernandes Veiga Maia
A78034 Diogo Afonso Silva Costa
A73909 Francisco Lira Pereira
A79607 Marco António Rodrigues Oliveira Silva

Braga
2018 Março

Resumo

Neste documento será apresentado o trabalho desenvolvido no âmbito da Unidade Curricular de Sistemas de Representação de Conhecimento e Raciocínio do 3º ano do Mestrado Integrado em Engenharia Informática. Foi proposto o desenvolvimento de uma base de conhecimento e respetivos predicados na área da saúde, mais concretamente, com utentes, cuidados e seus respetivos prestadores. Serão explicitados os predicados utilizados para navegar a base de conhecimento bem como os respetivos invariantes, responsáveis por manter a consistência da mesma.

Conteúdo

1 Introdução

O sistema de representação de conhecimento e raciocínio, representado neste projeto, baseia-se na área de prestação de cuidados de saúde. Este tem como objetivo permitir a construção e evolução de uma base de conhecimento que possa retratar as principais componentes no que toca à saúde, nomeadamente, o utente, o prestador do cuidado de saúde, o cuidado propriamente dito e a instituição em que este é realizado.

Para tal, é usada a linguagem de programação em lógica, PROLOG. Esta permite através de um conjunto de predicados, invariantes e estruturas lógicas orquestrar um base de conhecimento consistente e, acima de tudo, evolutiva.

2 Preliminares

O PROLOG é uma linguagem de programação em lógica que contém um pequeno conjunto de mecanismos básicos, nomeadamente, unificações, estruturas em árvore e *backtracking* automático. Neste relatório muitos destes mecanismos encontram-se subjacentes e são deveras importantes para a compreensão do mesmo. [1]

Além disso, são explicados de forma extensiva os diferentes predicados que compõem a base de conhecimento. Estes seguem o clausulado de *Horn*, que diz que apenas existe um termo positivo (conclusão) e que todos os predicados estão quantificados universalmente. Efetivamente, estes são fórmulas fechadas visto que todas as variáveis têm significado, não importa o seu valor.

Por fim, importa notar que o algoritmo de resolução adotado pelo PROLOG segue o conceito *modus ponens*, que permite derivar como verdadeiro uma conclusão de uma cláusula, e o *modus tollens*, que permite dirigir a prova para um ponto em particular. É a conjunção de este e outros mecanismos que permite que o PROLOG para ser usado para a representação de conhecimento e raciocínio e, consequentemente, neste projeto.

3 Descrição do Trabalho e Análise de Resultados

3.1 Predicados base

A área da prestação de cuidados de saúde é caracterizada por um sistema de representação de conhecimento e raciocínio composto por quatro predicados fundamentais.

- **utente**: #IdUt, Nome, Idade, Morada → { V,F }
- **prestador**: #IdPrest, Nome, Especialidade, #IdInst → { V,F }
- **cuidado**: Data, #IdUt, #IdPrest, Descrição, Custo → { V,F }
- **instituicao**: #IdInst, Nome, Cidade → { V,F }

O último predicado surgiu da necessidade de identificar os cuidados de saúde prestados numa determinada cidade. No entanto, o único conhecimento disponível era a morada do cliente. Não foi do agrado da equipa utilizar esta como

forma de unificar o conhecimento para concluir quais os cuidados de saúde prestados numa determinada cidade, visto que os utentes poderiam realizar consultas fora do seu local de residência. Assim sendo, foi estendida a informação referente à instituição de forma a obter uma base de conhecimento mais robusta.

Assim sendo, são estas as ferramentas iniciais que possibilitam a criação de uma base de conhecimento ligada à prestação de cuidados de saúde, com possibilidade de evolução do próprio conhecimento.

3.2 Extensões dos predicados

3.2.1 Registar utentes, prestadores e cuidados de saúde

Após a declaração dos predicados **utente**, **prestador**, **cuidado** e **instituicao**, surge a necessidade de permitir a que a base de conhecimento possa evoluir. Para tal, foi necessária a implementação do predicado **registar**. No entanto, a inserção de conhecimento pode levar a um estado inconsistente. A título exemplificativo, tomando a hipótese do `utente(1, andre, 25, lisboa).` já se encontrar a base de conhecimento, a inserção do `utente(1, guilherme, 60, faro).` levaria a que houvesse dois utentes com o mesmo identificador (*IdUt*), o que origina uma inconsistência na base de conhecimento. A solução surge na forma de invariantes. Estes representam testes à consistência da base de conhecimento, isto é, permite que haja as condições adequadas para que o sistema possa crescer.

Deste modo, foram adicionadas invariantes que não permitem que se possa inserir dois utentes com o mesmo *id*. O mesmo foi aplicado ao *id* dos prestadores e das instituições. Além disso, o predicado **cuidado** não poderá aparecer em duplicado na base de conhecimento, nem referenciar utentes e prestadores que ainda não existam, obrigando desta forma a que estes sejam adicionados primeiro.

Assim sendo, torna-se possível que o sistema evolua de forma consistente através do predicado **registar**. No entanto, é necessário que este realize os devidos testes antes de dar a inserção como concluída. Para tal foi construído o predicado **solucoes**.

```
solucoes(X,Y,Z) :- findall(X,Y,Z).
```

Este coloca na lista Z o resultado, na forma de X, de todas as unificações do predicado Y com a base de conhecimento. De seguida, insere-se o novo termo através do predicado `assert`. Por fim, são testados todos os invariantes, guardados em Z, usando o predicado `teste`. Caso algum destes falhe, o termo inserido é retirado da base de conhecimento através do predicado `retract`.

```
insere(P) :- assert(P).
insere(P) :- retract(P), !, fail.

registar( Termo ) :-
    solucoes(Inv, +Termo :: Inv, S),
        insere(Termo),
        teste(S).
```

Todos estes componentes são necessários para garantir, através do predicado

`registar`, que a base de conhecimento se mantenha consistente ao longo do seu crescimento.

3.2.2 Remover utentes, prestadores e cuidados de saúde

De facto, assim como é possível evoluir a base de conhecimento inserindo conhecimento também o é removendo conhecimento. Para tal, é o usado o predicado `remover`. Este, à semelhança do da `registar` permite que a base de conhecimento cresça mas desta vez com a remoção de predicados. No entanto, esta deve ser controlada de forma a não causar inconsistências. Para tal, implementou-se alguns invariantes, nomeadamente, a impossibilidade de remover utentes e prestadores que ainda tenham a si associados cuidados de saúde.

Desta forma, tem-se todos os componentes necessários para a implementação do predicado `remover`. Primeiramente, são recolhidos numa lista, todos os invariantes que se referem a remoção de conhecimento. De seguida, é removido o termo indicado. Por fim, é realizado um teste a todos os invariantes. Caso algum estes não tenha valor de verdade positivo, então é colocado novamente na base de conhecimento.

```
remove(P) :- retract(P).
remove(P) :- assert(P), !, fail.

remover( Termo ) :-
    solucoes(Inv, -Termo :: Inv, S),
        remove(Termo),
    teste(S).
```

Assim sendo, com os predicados `registar` e `remover` é possível crescer a base de conhecimento de forma **consistente**.

3.2.3 Identificar utentes por critérios de seleção

Este predicado, denominado *consultaUtente*, permite a procura de utentes através de cada um dos seus átomos. Por outras palavras, é possível procurar utentes pelo seu identificador, nome, idade ou morada. Permite-se também a procura através do uso simultâneo de uma ou mais das variáveis mencionadas. É de notar que efetuar tal procura através do identificador em conjunto com qualquer argumento será redundante, visto que, graças ao invariante por referência de *utente*, qualquer utente que exista na base de conhecimento terá um ID único e específico a si.

Para tal, recorre-se ao predicado *solucoes*, com o qual se gera uma lista contendo toda a informação dos *utentes* cujos argumentos introduzidos coincidem com a respetiva informação. Este predicado receberá como argumentos a estrutura de cada elemento da lista, os predicados que usará para testar se contêm a informação procurada e a variável na qual será escrita a lista. Neste caso, os elementos da lista seguirão a estrutura de *utente* e o predicado de teste será o próprio predicado *utente*. Com o intuito de tomar partido disto, será necessário adicionar mais um argumento ao *consultaUtente*, no qual a lista será guardada.

3.2.4 Identificar as instituições prestadoras de cuidados de saúde

O predicado *consultaInstituicoes* permite visualizar uma lista de todas as *instituicoes* presentes na base de conhecimento cujos prestadores tenham efetuado cuidados médicos. De modo semelhante ao predicado anterior, é possível efetuar a procura através do identificador, nome e/ou cidade correspondente à instituição.

Como no predicado anterior, recorrer-se-á ao predicado *solucoes* e, como tal, necessitar-se-á de uma variável adicional no predicado *consultaInstituicoes*. Neste caso, os elementos da lista seguirão a estrutura de *instituicao* e os predicados de teste serão o predicado *instituicao*, bem como os predicados *prestador* e *cuidado*. Isto deve-se ao facto de que se pretende que apenas as instituições que dispõem de cuidados médicos sejam consideradas neste predicado.

Como se recorre a vários predicados para a procura, existe a possibilidade de ser introduzida na lista mais do que uma instância de uma dada instituição. Como tal, recorreu-se ao predicado *sort* para eliminar duplicados da lista.

3.2.5 Identificar cuidados de saúde prestados

Com o predicado *consultaCuidados*, é possível identificar os cuidados de saúde prestados numa determinada instituição, cidade ou até mesmo numa determinada data.

A construção deste foi baseada no encadeamento de outros predicados mais simples. Assim, conclui-se que os argumentos necessários serão as variáveis *IDI*, *Ci*, *D* e finalmente *S*. No que diz respeito à variável *IDI*, esta representa o identificador da instituição da qual se pretende fazer a pesquisa na base de conhecimento. De forma análoga, *Ci* representa a cidade onde a instituição se encontra sediada na qual o cuidado foi prestado e finalmente *D* representa a data sobre a qual a pesquisa deverá incidir. A variável *S* é utilizada na apresentação do resultado do predicado.

Mais uma vez, a utilização do predicado soluções foi determinante para a construção da solução. Deste modo, o primeiro elemento que o *solucoes* receberá será o formado dos elementos constituintes da lista, seguido pelos predicados *instituicao*, *prestador* e finalmente *cuidado*. De salientar que cada um dos predicados referidos anteriormente recebe as variáveis passadas como argumento, tendo especial atenção neste caso às variáveis *IDI* e *IDP* que permitem estabelecer a ligação entre os cuidados e os prestadores que por sua vez prestaram um determinado serviço numa instituição.

3.2.6 Identificação de utentes

Este predicado é uma extensão do predicado *consultaUtente* e possibilita a procura de todos os utentes que recorreram a um determinado prestador, instituição e/ou especialidade médica. Atinge isto procurando *prestadores* cujo identificador, especialidade e/ou instituição médica coincidem com o(s) procurado(s) e, tendo encontrado *prestadores* válidos, identifica os utentes que recorreram a estes serviços procurando *cuidados* cujo identificador corresponda aos *prestadores* em questão.

Para tal, utiliza-se o predicado *solucoes* de novo, desta vez com os elementos da lista seguindo a estrutura de *utente* e com os predicados de teste *utente*,

prestador e *cuidado*. Fazer-se-á uso do predicado *sort* mais uma vez, de modo a evitar a existência de informação repetida na lista resultante.

3.2.7 Instituições/prestadores a que um utente já recorreu

O predicado *todasInstPrest* permite identificar quais prestadores e/ou instituições nos quais um determinado utente recebeu cuidados médicos. Deste modo, recebe como argumento o identificador de um dado utente e uma variável na qual se colocará a lista de pares de identificadores (prestador, instituição).

Para tal, recorrer-se-á ao predicado *solucoes*, cujos argumentos serão a estrutura de cada elemento da lista, os predicados *prestador* e *cuidado* e a variável responsável por demonstrar a lista. Cada elemento da lista é um tuplo constituído pelo identificador do prestador e da instituição, respetivamente.

3.2.8 Calculo do custo total dos cuidados de saúde

O predicado **totalCuidados** determina a soma de todos os custos que se encontram descritos na base de conhecimento com base nos argumentos definidos. Este predicado aceita o *ID* de um *utente*, uma *especialidade*, o *ID* de um prestador, uma *data* ou qualquer combinação possível destas informações. Sendo assim, é construída uma lista que alberga todos os custos encontrados na base de conhecimento. Finalmente, recorreu-se ao predicado **somaL** que recebe uma lista e apresenta a soma de todos os seus elementos.

3.3 Funcionalidades acrescentadas

3.3.1 Receitas da instituição

Uma vez que nos encontramos numa área em que é tratada informação com custos associados, achou-se interessante construir um predicado *receitasInst* que, apenas com base na instituição, apresente o total dos custos associados à mesma.

Deste modo, para o desenvolvimento deste predicado, foi utilizado o predicado soluções. Para a obtenção da informação correta, é necessário combinar as informações presentes na base de conhecimento que estão associadas a *cuidado* e *prestador*. Assim, no segundo parâmetro do predicado soluções, é explicitado que o *ID* de um prestador em que a instituição seja a que foi fornecida, seja o mesmo presente na base de conhecimento associado ao cuidado, uma vez que a ligação entre instituições e cuidados é estabelecida através dos prestadores de cuidados. Deste modo, é construida mais uma vez uma lista com todos os custos encontrados na base de conhecimento e finalmente é utilizado o predicado **somaL** para que a soma de todos os valores da lista sejam calculada.

3.3.2 Lista dos cuidados de saúde de um utente

O predicado `cuidadosUtente` tem como propósito apresentar um lista com todos os cuidados de um determinado utente. Esta encontrar-se-á ordenada de forma decrescente em relação ao número de ocorrências de um determinado cuidado. De forma a atingir este objetivo são necessários quatro predicados: `solucoes`, `pairFreq`, `sort` e `reverse`.

Inicialmente, unificam-se todos os predicados com um determinado *id* de utente. O resultado dessa unificação é colocado numa lista **S**. Conjuntamente,

é testado o predicado `pairFreq` que associa a cada especificação de um cuidado o número de vezes que esta ocorre, com a ajuda do predicado `conta`. Por fim, é feita a conjunção com os predicados `sort` e `reverse`, que ordenam e invertem a ordem dos elementos da lista, respetivamente.

Assim sendo, é possível perceber quais os cuidados mais frequentes para um determinado utente.

3.3.3 Número de prestadores por instituição

O predicado *quantosPrest* permite determinar o número de prestadores de cuidados médicos trabalham numa determinada instituição. Como tal, recebe apenas o identificador único da instituição cuja contagem se quer efetuar. O predicado atinge isto gerando uma lista de prestadores cujo identificador da instituição para qual trabalha corresponde ao identificador da instituição em questão e calcula o comprimento da lista resultante. Para tal, recorre-se ao predicado *solucoes* para gerar a lista e ao predicado *comprimento* para calcular o número de elementos na mesma.

No caso de *solucoes*, são passados como argumentos a estrutura dos elementos da lista e o predicado *prestador*. Por sua vez, o predicado *comprimento* receberá a lista gerada e determinará o seu comprimento.

3.3.4 Cuidados oferecidos por uma instituição

Achou-se também interessante oferecer a um possível utilizador um catálogo de cuidados que já foram prestados por uma determinada instituição com o predicado *cuidadosInst*. Este catálogo apresenta a descrição do mesmo, uma vez que esta última secção pode conter informações relevantes para o utilizador.

Mais uma vez, recorreu-se ao predicado soluções como forma de construir a lista a apresentar com a informação pedida.

Deste modo, apenas se necessita de fornecer o *ID* da instituição em causa. Uma vez que, cada prestador tem associada a si uma instituição pelo seu número de identificação, utilizar-se-á o predicado prestador para estabelecer a ligação com os cuidados prestados, verificando todos os prestadores que têm associados a si o número identificativo da instituição fornecida ao predicado. Assim apenas serão apresentados resultados referentes à instituição previamente definida.

3.3.5 Relatório mensal de uma instituição

Com o predicado *relatContas* é possível obter uma visão geral de todos os cuidados prestados sendo possível fornecer um mês, ano, e finalmente um identificador de uma instituição.

Para a construção deste, foram utilizados dois predicados auxiliares. Inicialmente, é construida uma lista com todos os cuidados que correspondem aos argumentos fornecidos. De seguida é utilizado o predicado *calcTotal*, que irá calcular o total de despesas presentes na lista resultante do predicado *solucoes*. Por último, com o predicado *addTotal* é construída a lista final a apresentar adicionando apenas no final da lista resultante do predicado *solucoes* um elemento que especifica o total das despesas para uma melhor visão do panorama mensal.

4 Conclusões e Sugestões

Este projeto permite a realização de questões a uma base de conhecimento relacionada com a prestação de cuidados de saúde.

Efetivamente, a base de conhecimento concebida permite responder a uma panóplia de questões, no mínimo interessantes. A título exemplificativo, é possível saber qual quais os cuidados que um utente recebeu, quais os prestadores de cuidados numa determinada instituição ou mesmo qual o relatório de cuidados de saúde numa instituição em específico.

No inicio tivemos algumas dificuldades em perceber ao certo o que teríamos de fazer e como o deveríamos de implementar para conseguir respeitar todos os requisitos, mas com algum esforço, facilmente conseguiu-se superar estas dificuldades e desenvolver predicados que satisfizessem as questões predefinidas. Desta forma, tem-se como exemplo a utilização do *sort*. Este permitiu que se resolvesse o problema da existência de parâmetros duplicados em listas. É um facto que este também altera a ordem da ocorrência destes elementos na lista, mas como esta não é importante, acaba por ser até uma ajuda no momento em que, enquanto utilizadores, é feita a procura, a lista tem variados elementos e queremos encontrar entre eles um especial, basta ajustar o nosso critério de procura pois se sabe como ela está organizada.

No entanto, a área da prestação de cuidados de saúde é bastante abrangente e a base de conhecimento aqui desenvolvida pode, e deve, ser evoluída em iterações futuras do projeto. Desta forma, propõe-se o acrescento de um predicado "receitas médicas"que será associado aos utentes, com os vários medicamentos prescritos. A adição deste novo predicado permitiria formular um novo conjunto de questões bastantes pertinentes, como por exemplo, que medicamentos se encontra a tomar um determinado utente ou quais os medicamentos mais receitados por instituição num determinado ano. Assim sendo, é possível constatar que esta é um base de conhecimento com bastantes frentes de crescimento.

Assim, a realização deste primeiro trabalho foi bastante importante para entender como funciona uma linguagem de programação em lógica e os problemas que esta se propõe a resolver. Em especial, como funciona e como se resolve este tipo de problemas em PROLOG. Conseguiu-se consolidar os conhecimentos adquiridos nas aulas práticas, e também complementar estes com alguma pesquisa realizada ao longo do projeto. Conclui-se então com um sentimento de satisfação derivado do trabalho desenvolvido, sendo que esta é uma boa maneira de introduzir a matéria lecionada na UC.

Referências

[1] Bratko, I. *PROLOG Programming for Artificial Intelligence*, 2nd ed. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.