Universidade do Minho

Exercício 1

Licenciatura em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio

2ºSemestre (2014/2015)

67673 André Geraldes

67665 Patrícia Barros

67709 Sandra Ferreira

Braga

Março de 2015

Resumo

Este trabalho foi realizado no âmbito da Unidade Curricular de Sistemas de Representação de Conhecimento e Raciocino e consiste no desenvolvimento de um sistema de representação de conhecimento capaz de descrever a árvores genealógica de uma família.

Índice

Índice de Figuras

Introdução

O trabalho prático descrito neste relatório consiste no desenvolvimento de um sistema de representação de conhecimento e raciocínio que seja capaz de descrever uma árvore genealógica de uma família.

A linguagem utilizada para desenvolver este trabalho será a linguagem de programação lógica ***PROLOG***.

Neste relatório apresentam-se o processo de desenvolvimento do sistema de raciocínio e os resultados obtidos.

Preliminares

De forma a conseguirmos realizar o trabalho proposto foi necessário, através das aulas da Unidade Curricular de Sistemas de Representação de Conhecimento e Raciocínio, possuirmos conhecimentos base de ***PROLOG*** e construção de mecanismos de raciocínio para resolução de problemas.

Descrição do Trabalho

Como referido anteriormente este trabalho consiste na realização de um sistema de representação de conhecimento e raciocino que possibilite a descrição de uma árvore genealógica.

Caso Prático de Aplicação

Para que fosse possível demonstrar as capacidades do sistema desenvolvido foi necessário criar um caso prático de aplicação do cenário criado. Segue-se a árvore genealógica da família que utilizamos.

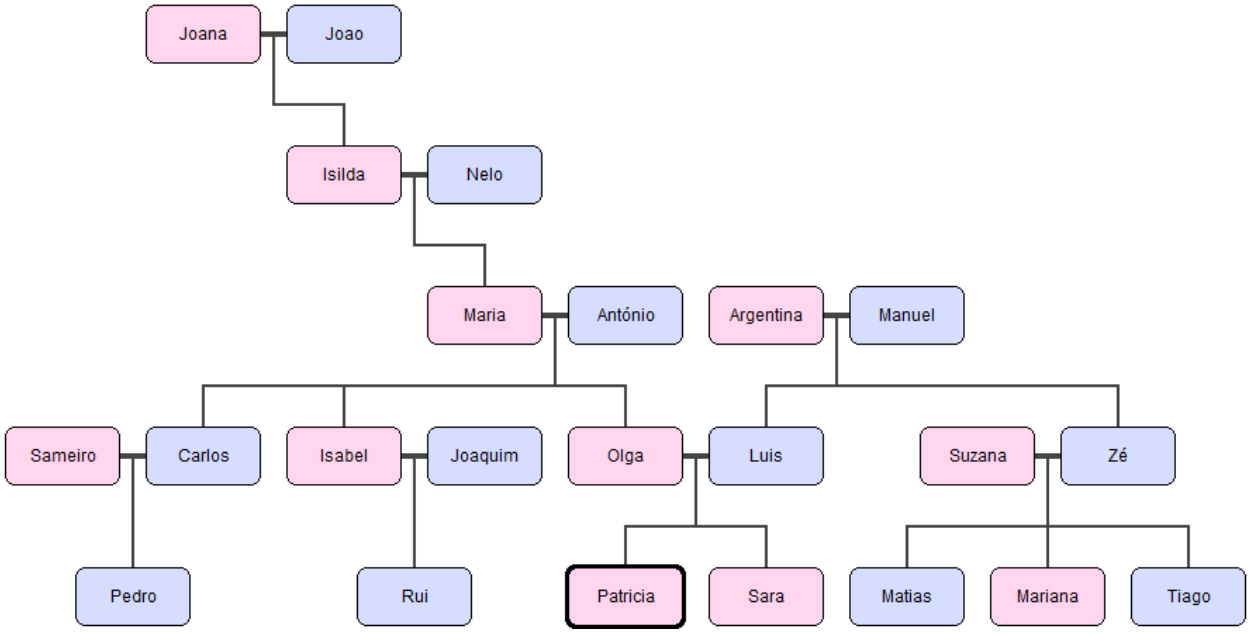

Figura 1 - Árvore genealógica do caso prático utilizado

Desenvolvimento dos Predicados

O nosso sistema permite representar as seguintes relações familiares: filho, pai, irmão, tio, sobrinho, primo, cunhado, avô, neto, bisavô, bisneto, trisavô, trisneto e companheiro (conjugue). Para isso foram criados os predicados necessários à sua descrição.

Começou-se pela relação **filho**, definindo todas as relações deste tipo existentes para este caso prático, da seguinte forma:

```
% Extensão do predicado filho: Filho,Pai -> {V,F}
filho(patricia,luis).
filho(patricia,olga).
filho(sara,luis).
filho(sara,olga).
```

Através da anterior foi definido o predicado **pai**: se A é filho de B então B é pai de A.

```
% Extensão do predicado pai: Pai,Filho -> {V,F}
pai( P,F ) :- filho( F,P ).
```

O predicado **irmão** foi definido através do predicado pai: se os dois indivíduos tem um pai em comum então são irmãos.

```
% Extensão do predicado irmao : Irmao,Irmao -> {V,F}
irmao(A,B) :- pai(X,A), pai(X,B), A \== B.
```

Decidimos definir também um predicado para definir a relação de **companheiro**, ou conjugue, entre duas pessoas. Assumindo uma família dita funcional, decidimos que a definição de companheiro era alguém com que se tivesse um filho em comum.

```
% Extensão do predicado companheiro : Individuo, Individuo -> {V,F}
companheiro(A,B) :- filho(X,A), filho(X,B).
```

Com o predicado companheiro foi-nos possível construir o predicado **cunhado** que se define como sendo o companheiro do irmão ou então o companheiro de alguém cujo irmão tem como companheiro o segundo individuo.

```
% Extensao do predicado cunhado : Individuo, Individuo -> {V,F}
cunhado(A,B) :- companheiro(A,X), irmao(X,B).
cunhado(A,B) :- companheiro(A,X), irmao(X,Y), companheiro(Y,B).
```

Para definir o predicado **tio** recorremos aos predicados irmão, filho e companheiro: A é tio de B se um dos pais de B é irmão de A, ou se o seu companheiro é irmão do pai de B.

```
% Extensão do predicado tio: Tio,Sobrinho -> {V,F}
tio(T,S) :- irmao(T,P), filho(S,P).
tio(T,S) :- companheiro(A,T), irmao(A,X), filho(S,X).
```

Com o predicado tio definido facilmente definimos também o predicado **sobrinho**: se A é tio de B, B é sobrinho de A.

```
% Extensão do predicado sobrinho: Sobrinho,Tio -> {V,F}
sobrinho(S,T) :- tio(T,S).
```

Através também do predicado tio foi possível definir o predicado **primo**: A é primo de B se A tem um pai que é tio de B.

```
% Extensão do predicado primo: Primo, Primo -> {V,F}
primo(X,Y) :- pai(Z,X), tio(Z,Y).
```

Definimos ainda o predicado **avo** à custa do predicado filho: A é avô de B se B é filho de alguém que por sua vez é filho de A.

```
% Extensão do predicado avo: Avo,Neto -> {V,F}
avo( A,N ) :- filho( N,X ) , filho( X,A ).
```

Facilmente construímos de seguida o predicado **neto:** A é neto de B se B é avô de A.

```
% Extensão do predicado neto : Neto,Avo -> {V,F}
neto(N,A) :- avo(A,N).
```

O predicado **bisavô** foi definido utilizando os predicados pai e avô: A é bisavô de B se A é pai de alguém que é avô de B.

```
% Extensão do predicado bisavo: Bisavo,Bisneto -> {V,F}
bisavo(X, Y) :- pai(X,Z), avo(Z,Y).
```

Facilmente construímos de seguida o predicado **bisneto:** A é bisneto de B se B é bisavô de A.

```
% Extensão do predicado bisneto: Bisneto,Bisavo -> {V,F}
bisneto(X, Y) :- bisavo(Y,X).
```

Seguindo o mesmo pensamento contruímos os predicados **trisavô** e **trisneto.**

```
% Extensão do predicado trisavo: Trisavo,Trineto -> {V,F}
trisavo(X, Y) :- pai(X,Z), bisavo(Z,Y).
% Extensão do predicado trisneto: Trisneto,Trisavo-> {V,F}
trisneto(X, Y) :- trisavo(Y,Z).
```

Estes foram os predicados básicos definidos para representar as relações entre os membros da família.

No entanto representamos ainda os predicados **descendente** e **ascendente** cujo objetivo é determinar se um individuo é descendente de outro. Estes predicados tem duas variantes: uma variante com 2 argumentos (os dois indivíduos) e a outra com 3 argumentos (os dois indivíduos e o seu grau de parentesco). Sabemos que uma pessoa é descendente de outra quando é sua filha ou quando algum seu ascendente é descendente dessa pessoa.

```
% Extensão do predicado descendente: Descendente, Ascendente -> {V,F}
descendente( X,Y ) :- filho( X,Y ).
descendente( X,Y ) :-
        filho( X,A ),
        descendente( A,Y ).
% Extensão do predicado descendente: Descendente, Ascendente, Grau -> {V,F}
descendente( D,A,1 ) :- filho( D,A ).
descendente( D,A,G ) :-
        filho( D,X ),
        descendente( X,A,N),
        G is N+1.
```

É possível saber que A é ascendente de B se B for descendente de A.

```
% Extensão do predicado ascendente: Ascendente, Descendente -
> {V,F}
ascendente( X,Y ) :- descendente( Y,X ).
% Extensão do predicado ascendente: Ascendente, Descendente,
Grau -> {V,F}
ascendente( D,A,G ) :- descendente(A,D,G).
```

Foram ainda definidos os predicados **descendenteAte** e **ascendenteAte** que verificam se A é descendente (ou ascendente, respetivamente) de B num grau menor ou igual do que o passado como argumento. Para a construção destes predicados só foi necessário utilizar os predicados anteriores (descendente e ascendente) e acrescentar uma condição de desigualdade para o grau.

```
% Extensão do predicado descendenteAte : Descendente,
Ascendente, Grau -> {V,F}
descendenteAte(D,A,G) :- descendente(D,A,Z), Z=<G.
% Extensão do predicado ascendenteAte: Ascendente,
Descendente, Grau -> {V,F}
ascendenteAte(A,D,G) :- descendenteAte(D,A,G).
```

No contexto do problema é também útil ter a possibilidade de consultar todos os indivíduos que possuem uma determinada relação familiar com um outro. Então foram desenvolvidos predicados que permitem essa consulta para todos os tipos de relacionamentos definidos anteriormente. Para a sua construção utilizamos o predicado ***findall*** do ***PROLOG*** que nos permite ter uma lista de todas as soluções de um determinado predicado.

Segue-se um exemplo de como estes predicados foram definidos.

```
% Extensão do predicado filhos: Pai,Resultados -> {V,F}
filhos(I, R) :-
findall(P, filho(P, I), S),
R = S .
```

No entanto em alguns casos a lista resultante obtida continha elementos repetidos pelo que foi necessário a utilização de predicados auxiliares para a remoção dos mesmos.

```
% Extensão do predicado apagaTudo : Elemento, Lista, Resultado ->{V,F}
apagaTudo(X,[],[]).
apagaTudo(X,[X|L],R) :- apagaTudo(X,L,Res), R = Res.
apagaTudo(X,[Y|L],R) :- X\==Y, apagaTudo(X,L,NL), R = [Y|NL].
% Extensão do predicado apagaRepetidos: Lista,Resultado -> {V,F}
apagaRepetidos([],[]).
apagaRepetidos([X|L],R) :- apagaTudo( X, L, Res ), apagaRepetidos( Res, ResFinal ), R = [X|ResFinal].
```

O predicado **apagaRepetidos** foi então integrado nos predicados que devolvem a lista de elementos relacionados com um individuo de uma determinada forma da seguinte maneira:

```
% Extensão do predicado irmaos: Irmao,Resultados -> {V,F}
irmaos(I, R) :- findall(P, irmao(P, I), S),
apagaRepetidos(S,Res),
R = Res.
```

Foi também definido um predicado **relação** que dados dois indivíduos determina qual a sua relação familiar.

```
% Extensão do predicado relacao : Individuo, Individuo, Relacao -> {V,F}
relacao(A,B,pai) :- pai(A,B).
relacao(A,B,filho) :- filho(A,B).
relacao(A,B,avo) :- avo(A,B).
relacao(A,B,neto) :- neto(A,B).
relacao(A,B,tio) :- tio(A,B).
relacao(A,B,sobrinho) :- sobrinho(A,B).
relacao(A,B,primo) :- primo(A,B).
relacao(A,B,irmao) :- irmao(A,B).
relacao(A,B,bisavo) :- bisavo(A,B).
relacao(A,B,bisneto) :- bisneto(A,B).
relacao(A,B,trisavo) :- trisavo(A,B).
relacao(A,B,trisneto) :- trisneto(A,B).
relacao(A,B,companheiro) :- companheiro(A,B).
relacao(A,B,cunhado) :- cunhado(A,B).
```

Para a descrição da naturalidade de cada individuo foi criado um predicado **natural** que dado um individuo e a sua naturalidade determina se a afirmação é verdadeira ou não.

Inserção e Remoção de Conhecimento

Para possibilitar a inserção e remoção de conhecimento foram criados os predicados **evolução** e **remoção.** O predicado remoção não é mais do que uma chamada do ***retract.***

```
% Extensão do predicado que permite a remoção do conhecimento
remocao(Termo) :- retract(Termo).
```

O predicado evolução só permite a inserção de conhecimento que verifique os invariantes definidos por nós.

```
% Extensão do predicado que permite a evolucao do conhecimento
evolucao( Termo ) :- findall(Invariante, +Termo::Invariante, Lista),insercao( Termo),teste(Lista).

insercao(Termo) :- assert(Termo).
insercao(Termo) :-retract(Termo),!,fail.

teste([]).
teste([R|LR]) :-R,teste(LR).
```

Criamos vários invariantes para garantir a consistência do sistema: invariantes estruturais e invariantes referencias. Os primeiros foram criados para garantir que não é inserido conhecimento repetido no sistema, e são implementados da seguinte forma:

```
+filho(F,P) :: (findall( (F,P),(filho( F,P )),S ),comprimento( S,N ), N == 1).
```

Os segundos servem para garantir a coerência do sistema, por exemplo, não deixando que um individuo tenha mais do que dois pais. Da mesma forma não seria conveniente que uma pessoa tivesse mais do que quatro avôs ou mais do que uma naturalidade. Garantimos ainda que não é possível que alguém seja pai de

nenhum dos seus ascendentes, e, partindo de um princípio de que tratamos de uma família funcional, não é possível, por exemplo, o tio ser pai do sobrinho.

Análise de Resultados

Concluída a descrição da aplicação passamos para a confirmação de que a mesma se encontra funcionar corretamente. Para isso testamos vários predicados e comparamos com a informação do caso prático utilizado para verificar que as respostas são as esperadas.

```
| ?-
| ?- pai(luis,patricia).
yes
| ?- irmao(sara,patricia).
yes
| ?- tio(isabel,patricia).
yes
| ?- sobrinho(rui,olga).
yes
| ?- sobrinho(matias,isabel).
no
| ?- primo(matias,patricia).
yes
| ?- primo(matias,rui).
no
| ?- companheiro(luis,olga).
yes
| ?- cunhado(olga,suzana).
yes
| ?- descendente(patricia,manuel).
yes
| ?- descendente(patricia,manuel,2).
yes
| ?- ascendente(manuel,patricia).
yes
| ?- ascendente(manuel,patricia,2).
yes
| ?- ascendenteAte(manuel,patricia,2).
yes
| ?- ascendenteAte(manuel,patricia,1).
no
| ?- primos(patricia,R).
R = [rui,tiago,mariana,matias,pedro] ?
yes
| ?- sobrinhos(olga,R).
R = [pedro,rui,tiago,mariana,matias] ?
yes
| ?- ascendentes(patricia,R).
R = [luis,olga,argentina,manuel,maria,antonio,isilda,nelo,joana,joao] ?
yes
| ?- ascendentesAte(patricia,2,R).
R = [luis,olga,argentina,manuel,maria,antonio] ?
yes
| ?- relacao(isabel,joaquim).
no
| ?- relacao(isabel,joaquim,R).
R = companheiro ?
yes
| ?- relacao(sameiro,olga,R).
R = cunhado ?
yes
| ?-
```

Figura 2 - Resultados obtidos

Conclusões e Sugestões

Com este trabalho foi possível observarmos como os sistemas de conhecimento e raciocínio podem ser úteis e interessante quando aplicados a casos práticos. Após a sua realização todos nós percebemos e apreciamos mais o ***PROLOG.***

Consideramos que produzimos um trabalho bastante satisfatório tendo implementado todas as funcionalidades exigidas e ainda alguns pormenores que não eram pedidos.