Universidade do Minho

Escola de Engenharia

Licenciatura em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio

Ano Lectivo de 2014/2015

Exercício 1

Grupo 29:

Carlos Morais, a64306

Filipe Ribeiro, a64315

Março, 2015

Resumo

Neste relatório é apresento um sistema de representação de conhecimento e raciocínio em que se pretende representar uma árvore genealógica de uma família e a naturalidade de determinados indivíduos.

Para a implementação do sistema é utilizada a linguagem de programação em lógica PROLOG.

No desenvolvimento do relatório serão apresentados todos os passos para a implementação do sistema. Será feita uma demonstração dos resultados obtidos sendo feita no final uma conclusão de todo o trabalho realizado.

Índice

Índice de Figuras

1. Introdução

Neste exercício é pretendido que se implemente um sistema para representação de uma árvore genealógica familiar e a naturalidade de determinados indivíduos.

O sistema a criar deverá representar relações familiares independentes entre os vários elementos de uma família e a identificação da naturalidade de vários indivíduos.

É pretendido que a base de conhecimento suporte a inserção de relações de naturalidade e relações familiares, determine a relação familiar entre dois indivíduos (ex: primos), determine o conjunto de elementos com descendência/ascendência familiar de um individuo até um determinado grau e calcule a relação familiar entre dois indivíduos.

Pretende-se também que a inserção de conhecimento respeite eventuais invariantes identificados no sistema.

O presente relatório é constituído por três capítulos fundamentais sobre a implementação do sistema. No primeiro será apresentado o estudo que precedeu o desenvolvimento do trabalho. No seguinte capitulo serão explicados todos os passos para o desenvolvimento do sistema e os motivos que nos fizeram tomar determinadas decisões, sendo no último feita uma apresentação de resultados obtidos para uma base de conhecimento previamente defina.

No final do relatório serão apresentadas algumas conclusões finais sobre o trabalho.

2. Preliminares

Para a implementação deste exercício, pretende-se representar uma árvore genealógica familiar. Cada individuo tem sempre uma determinada relação com outro individuo da mesma família. Será ainda possível representar a naturalidade de vários indivíduos. A representação da naturalidade não estará relacionada com a representação da árvore genealógica.

De forma a que a base de conhecimento tenha uma representação estruturada e organizada, a representação do conhecimento será feita à custa da relação filho. Com o conhecimento de todos os filhos é possível representar uma árvore genealógica e a partir desta obter informação válida e verdadeira.

Quanto à representação de conhecimento sobre a naturalidade dos indivíduos bastara representar a naturalidade para cada individuo.

3. Desenvolvimento

3.1. Relações Familiares e de Naturalidade

O sistema a criar tem como base a representação de relações familiares independentes. Para a representação das relações foram implementados vários predicados para as relações familiares.

Como base para a construção da árvore genealógica é implementado o predicado *filho : Individio1, Individuo2 -> {V,F}.*

ex: filho(joao, carlos).

Neste exemplo o predicado filho representa a informação de que o *joao* é filho do *carlos*.

A partir do predicado filho são construídos novos predicados para a definição de relações familiares.

Foi também considerado que dois indivíduos são casados quando tem pelo menos um filho em comum.

```
pai(P,F) :- filho(F,P).
irmao(I1,I2) :- pai(P,I1), pai(P,I2), I1 \== I2.
casado(I1,I2) :- filho(X,I1), filho(X,I2), I1 \== I2.
tio(T,S) :- irmao(T,I), pai(I,S).
sobrinho(S,T) :- tio(T,S).
primo(P1,P2) :- filho(P1,Pai1), filho(P2,Pai2), irmao(Pai1,Pai2).
avo(A,N) :- filho(N,Y), pai(A,Y).
neto(N,A) :- avo(A,N).
bisavo(X,Y) :- avo(X,Z), pai(Z,Y).
bisneto(B,Y) :- bisavo(Y,B).
```

Foi decidido representar as relações familiares até ao 4ºgrau (bisavó e bisneto). Assim, para determinar uma relação de maior grau poderá ser utilizado o predicado descendente ou ascendente, que determinam se um individuo é descendente/ascendente de outro individuo.

```
descendente(X,Y) :- filho(X,Y);
filho(X,Z), descendente(Z,Y).

ascendente(X,Y) :- descendente(Y,X).
```

Para a representação da naturalidade de um individuo é implementado o predicado
natural : Individuo, Local - {V,F}

ex: natural(joao, guimaraes).

Neste exemplo o predicado natural representa a informação de que o *joao* é natural do *guimaraes*.

3.2. Definição de Invariantes

De forma a que a representação da informação mantenha os seus invariantes estruturais e funcionais foram definidos invariantes para o predicado filho e natural.

Invariante estrutural para não permitir a inserção de conhecimento repetido na base de conhecimento.

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

Invariante referencial para não admitir que um individuo tenha mais do que dois progenitores.

```
+filho( F,P ) :: ( solucoes( Y, (filho(F,Y)), S ),
        comprimento(S,N),
        N=<2).
```

Invariante referencial para não admitir que um progenitor seja descendente do seu filho.

```
+filho( F,P ) :: nao(descendente(P,F,N)).
```

Invariante estrutural para que as relações de naturalidade não sejam repetidas na base de conhecimento.

```
+natural( I,L ) :: (solucoes( (I,L),(natural( I,L )),S ),
        comprimento( S,N ),
        N == 1).
```

Invariante Referencial para que um individuo não possua mais do que uma naturalidade na base de conhecimento,

```
+natural( I,L ) :: ( solucoes( Y, (natural(I,Y)), S ),
        comprimento(S,N),
        N==1).
```

Alguns dos invariantes implementados utilizam o predicado *comprimento: Lista, Resultado -> {V,F},* que calcula o comprimento de uma detrminada lista.

```
comprimento([],0).
comprimento([C|L], R) :- comprimento(L,G), R is G+1.
```

3.3. Inserção e remoção de conhecimento

De forma a permitir a inserção e remoção de conhecimento na base de conhecimento, foram implementados os predicados *insercao* e *remocao*.

O predicado *insercao : Termo -> {V,F},* inicialmente faz a inserção do termo na base de conhecimento através do predicado *evolucao* e testa a inviolabilidade dos seus invariantes através do predicado *teste*. Caso um dos invariantes tenha sido violado, o termo é removido e a inserção falha (fail). É utilizado o operador ! (CUT), para que após a detecção de falha, o predicado *evolucao* não volte a repetir a inserção do termo.

```
insercao(Termo) :- solucoes(Invariante, +Termo::Invariante, Lista),
        evolucao(Termo),
        teste(Lista).

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

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

O predicado *remocao : Termo -> {V,F},* faz a remoção na base de conhecimento do termo.

```
remocao(Termo) :- retract(Termo).
```

3.4. Conjunto de elementos com uma relação

Para determinar o conjunto de elementos com uma determinada relação a um individuo foi inicialmente implementado o predicado *solucoes: Individuo,Teorema, Solucoes -> {V, F}*, que recorre ao predicado *findall* para determinar o conjunto de soluções para um determinado Teorema.

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

Assim para cada relação familiar inicialmente implementada, foi implementado um novo predicado que para um determinado individuo, constrói um conjunto com os indivíduos que têm a mesma relação.

Inicialmente, o cálculo de alguns destes predicados retornava valores repetidos. Uma vez que um determinado individuo pode ter até dois progenitores, em alguns predicados, o conjunto de soluções era repetido pois eram calculados dois "caminhos" para as soluções. Para resolver este problema, foi implementado o predicado *elimRepetidos: Lista, Resultado -> {V, F}*, que elimina os valores repetidos numa determinada lista. Após o cálculo do conjunto inicial, é aplicado o predicado *elimRepetidos* para a remoção de elementos repetidos.

O predicado *elimRepetidos*, para cada elemento na lista, elimina os elementos repetidos na cauda da lista. É utilizado auxiliarmente o *predicado elimElemento: Lista, Elemento, Resultado -> {V, F}.*

```
elimRepetidos([], []).
elimRepetidos([H|T], R) :- elimElemento(T, H, Raux),
        elimRepetidos(Raux, R2),
R = [H|R2].

elimElemento([], _, []).
elimElemento([H|T], H, R) :- elimElemento(T, H, R).
elimElemento([H|T], E, R)   :- H \== E,
  elimElemento(T, E, R2),
  R = [H|R2].
```

Os predicados implementados para calcular o conjunto de elementos com uma determinada relação a um individuo foram os seguintes:

```
filhos(I,R) :- solucoes(F,filho(F,I),R).
pais(I,R) :- solucoes(P,pai(P,I),R).

tios(I,R) :- solucoes(T,tio(T,I),S),
        elimRepetidos(S,R).
sobrinhos(I,R) :- solucoes(S,sobrinho(S,I),S),
        elimRepetidos(S,R).
irmaos(I,R) :- solucoes(Ir,irmao(Ir,I),S),
        elimRepetidos(S,R).
primos(I,R) :- solucoes(P,primo(P,I),S),
```

```
    elimRepetidos(S,R).
avos(I,R) :- solucoes(P,avo(P,I),S),
    elimRepetidos(S,R).
netos(I,R) :- solucoes(P,neto(P,I),R).
bisavos(I,R) :- solucoes(P,bisavo(P,I),S),
        elimRepetidos(S,R).
bisnetos(Y,R) :- solucoes(P,bisneto(P,Y),S),
        elimRepetidos(S,R).

descendentes(I, R) :- solucoes(Y, descendente(Y, I), S),
        elimRepetidos(S,R).

ascendentes(I, R) :- solucoes(A, ascendente(A, I), S),
elimRepetidos(S,R).
```

3.5. Conjunto de descendentes e ascendentes

Para determinar os descendentes e ascendentes de um individuo até um determinado grau foram implementados os predicados *descendentesategrau* e *ascendentesategrau*.

O predicado *descendentesategrau: Individuo,Grau,Solucao -> {V,F},* determina quais os elementos na base de conhecimento que têm uma relação de descendência até um determinado grau. Para cada grau de descendência (de N até 1), é utilizado o predicado *solucoes* e o predicado *descendentegrau* para determinar os elementos descendentes nesse grau. O predicado *concat: Lista, Resultado -> {V,F}* é responsável por juntar a lista do grau com a lista calculada recursivamente para os restantes gaus.

```
descendentesategrau(I,0,[]).
descendentesategrau(I,G,Res) :-
        solucoes(X,descendentegrau(X,I,G),R),
        Y is G-1,
        descendentesategrau(I,Y,R2),
        concat(R,R2,S),
        elimRepetidos(S,Res).
```

O predicado *ascendentesategrau: Individuo,Grau,Solucao -> {V,F},* é idêntico ao anterior mas determina os elementos ascendentes.

```
ascendentesategrau(I,0,[]).
ascendentesategrau(I,G,Res) :-
        solucoes(X,ascendentegrau(X,I,G),R),
        Y is G-1,
        ascendentesategrau(I,Y,R2),
        concat(R,R2,S),
        elimRepetidos(S,Res).
```

Predicado *concat* utilizado para juntar duas listas.

```
concat([],L,L).
concat([H|T],L,[H|Y]) :- concat(T,L,Y).
```

3.6. Relação familiar entre dois indivíduos

Para calcular a relação entre dois indivíduos foi implementado o predicado *relacao: Individuo1, Individuo2, Resultado -> {V,F},* que determina a relação familiar que o Individuo1 tem com o Individuo2.

Uma vez que as relações implementadas na base de conhecimento têm no máximo um grau 4 (bisavó ou bisneto), para uma relação de grau superior, apenas é calculado se o Individuo1 é descendente ou ascendente do Individuo2.

```
relacao(X,Y,filho) :- filho(X,Y).
relacao(X,Y,pai) :- pai(X,Y).
relacao(X,Y,tio) :- tio(X,Y).
relacao(X,Y,sobrinho) :- sobrinho(X,Y).
relacao(X,Y,primo) :- primo(X,Y).
relacao(X,Y,irmao) :- irmao(X,Y).
relacao(X,Y,casado) :- casado(X,Y).
relacao(X,Y,avo) :- avo(X,Y).
relacao(X,Y,neto) :- neto(X,Y).
relacao(X,Y,bisavo) :- bisavo(X,Y).
relacao(X,Y,bisneto) :- bisneto(X,Y).
relacao(X,Y,descendente) :- descendente(X,Y).
relacao(X,Y,ascendente) :- ascendente(X,Y).
```

4. Resultados

Depois de feita a implementação do sistema, para testar o mesmo foi utilizado um pequeno exemplo para a base de conhecimento.

```
filho(pedro,joao).
filho(raul,joao).
filho(jose,joao).
filho(filipe,pedro).
filho(pedro,ana).
filho(raul,ana).
filho(jose,ana).
filho(rui,jose).
filho(rui,rita).
filho(andre,rui).
filho(andre,maria).
filho(carlos,andre).
filho(marco,andre).

natural(raul,guimaraes).
natural(jose,porto).
natural(ana,lisboa).
natural(carlos,guimaraes).
natural(filipe,braga).
```

A árvore genealógica representada pela base de conhecimento é a seguinte:

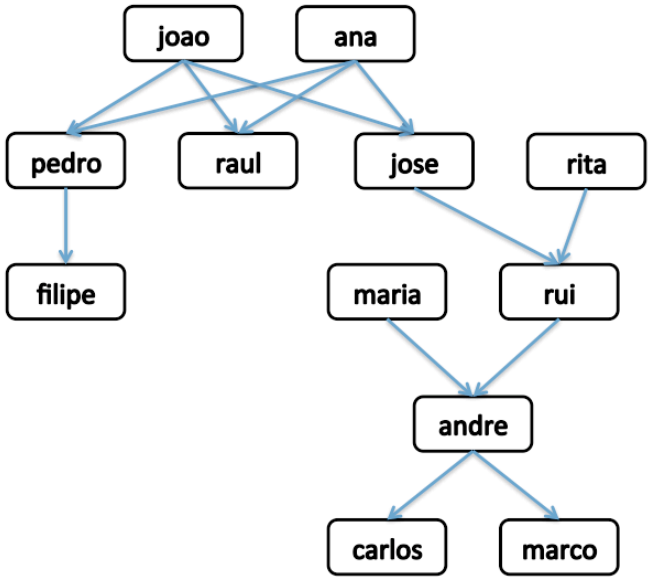

Figure 1 árvore genealógica de exemplo

A partir daqui é possível colocar algumas questões ao sistema.

```
| ?- filho(andre,rui).
yes
| ?- filho(pedro,andre).
no
| ?- pai(maria,andre).
yes
| ?- irmao(pedro,raul).
yes
| ?- avo(joao,rui).
yes
| ?- avo(joao,andre).
no
| ?- bisavo(joao,andre).
yes
```

Figure 2 conhecimento de relações

```
| ?- natura(carlos,braga).
no
| ?- natural(carlos,guimaraes).
no
| ?- natural(rui,porto).
```

Figure 3 conhecimento de naturalidade

```
| ?- ascendente(pedro,filipe).
yes
| ?- ascendente(pedro,lopes).
no
| ?- descendente(carlos,ana).
```

Figure 4 conhecimento de descendência/ascendência

Ao consultar a árvore da figura 1 podemos verificar que as questões introduzidas ao sistema devolvem o resultado esperado.

4.1. Inserção e remoção de relações de naturalidade

Para testar a inserção de conhecimento de naturalidade é utilizado o predicado *insercao*, que faz a verificação dos invariantes do termo a inserir.

```
| ?- insercao(natural(marco,coimbra)).
yes
| ?- insercao(natural(carlos,lisboa)).
no
| ?- insercao(natural(rita,lisboa)).
yes
| ?- remocao(natural(filipe,braga)).
yes
| ?- remocao(natural(filipe,braga)).
no
| ?- insercao(natural(filipe,faro)).
yes
```

Figure 5 inserção/remoção de conhecimento de naturalidade

A inserção, comporta-se como pretendido, permitindo apenas a inserção da naturalidade apenas para indivíduos para os quais o sistema não tem conhecimento.

Ao consultar o conhecimento sobre *natural,* verifica-se que este é o esperado após as inserções/remoções realizadas.

```
| ?- listing(natural).
natural(raul, guimaraes).
natural(jose, porto).
natural(ana, lisboa).
natural(carlos, guimaraes).
natural(marco, coimbra).
natural(rita, lisboa).
natural(filipe, faro).
```

Figure 6 resultado do conhecimento de natural

4.2. Inserção e remoção de relações familiares (relação filho)

Como justificado no desenvolvimento do sistema, foi decidido inserir todo o conhecimento no sistema através da relação filho e a partir daí obter as relações familiares correspondentes.

```
| ?- insercao(filho(nuno,diana)).
yes
| ?- insercao(filho(nuno,filipe)).
yes
| ?- insercao(filho(nuno,sara)).
no
| ?- remocao(filho(nuno,diana)).
yes
| ?- insercao(filho(nuno,sara)).
yes
```

Figure 7 inserção/remoção de conhecimento de relações familiares

As inserções têm o resultado esperado, podemos verificar, por exemplo, que não foi possível inserir o conhecimento *filho(nuno,sara)* uma vez que o individuo *nuno* já tem dois progenitores na base de conhecimento.

Ao consultar a relação filho podemos verificar o novo conhecimento da relação filho.

```
| ?- listing(filho).
filho(pedro, joao).
filho(raul, joao).
filho(jose, joao).
filho(filipe, pedro).
filho(pedro, ana).
filho(raul, ana).
filho(jose, ana).
filho(rui, jose).
filho(rui, rita).
filho(andre, rui).
filho(andre, maria).
filho(carlos, andre).
filho(marco, andre).
filho(nuno, filipe).
filho(nuno, sara).
```

Figure 8 resultado do conhecimento de filho após a inserção/remoção de conhecimento

4.3. Determinar conjunto de elementos com determinada uma relação

Para obter o conjunto com todos os elementos com uma determinada relação a um individuo, basta invocar o respetivo predicado com o nome do individuo e uma lista R que ficará com as soluções.

```
| ?- pais(nuno,R).
R = [filipe,sara] ?
yes
| ?- filhos(joao).
no
| ?- pais(nuno,R).
R = [filipe,sara] ?
yes
| ?- filhos(joao,R).
R = [pedro,raul,jose] ?
yes
| ?- netos(ana,R).
R = [filipe,rui] ?
yes
| ?- irmaos(raul,R).
R = [pedro,jose] ?
yes
| ?- irmaos(rita,R).
R = [] ?
yes
```

Figure 9 resultado do conjunto de elementos com uma relação

O mesmo pode ser feito para determinar os descendentes e ascendentes de um determinado individuo.

```
| ?- descendentes(jose,R).
R = [rui,andre,carlos,marco] ?
yes
| ?- ascendentes(andre,R).
R = [rui,maria,jose,rita,joao,ana] ?
yes
```

Figure 10 resultado do conjunto de elementos com descendência/descendência

4.4. Determinar os descendentes e ascendentes de um individuo até grau N

Para determinar o conjunto de indivíduos com descendência/ascendência a um individuo até um determinado grau utilizando os predicados *descendenteategrau/ ascendenteategrau*.

```
| ?- descendentesategrau(joao,3,R).
R = [andre,nuno,filipe,rui,pedro,raul,jose] ?
yes
| ?- ascendentesategrau(marco,4,R).
R = [joao,ana,jose,rita,rui,maria,andre] ?
yes
```

Figure 11 resultado de descendência/ascendência até grau N

4.5. Calcular a relação familiar entre dois indivíduos

Para o cálculo da relação familiar entre dois indivíduos utilizando o predicado *relação.*

```
| ?- relacao(joao,ana,R).
R = casado ?
yes
| ?- relacao(joao,carlos,R).
R = ascendente ?
yes
| ?- relacao(rui,raul,R).
R = sobrinho ?
yes
| ?- relacao(carlos,ana,R).
R = descendente ?
yes
| ?- relacao(carlos,marco,R).
R = irmao ?
yes
| ?- relacao(pedro,rita,R).
no
```

Nos casos em que a relação não esteja identificada na base de conhecimento, mas os indivíduos tenham uma descendência/ascendência entre si, esse é o resultado, tal como pretendido.

5. Conclusão

Com a realização de primeiro exercício foi possível interiorizar os conhecimento fundamentais da representação do conhecimento e raciocínio e da linguagem de programação e lógica PROLOG.

O sistema implementado apresenta todas as funcionalidades inicialmente pretendidas, sendo ainda adicionadas novas funcionalidades ao sistema de forma a obter um resultado final mais completo.

Anexos

I. Código

```
%_____________________________________________________________________________
% SIST. REPR. CONHECIMENTO E RACIOCINIO - LEI/3

%_____________________________________________________________________________
% Exercicio 1 - Grupo X

%_____________________________________________________________________________
% SICStus PROLOG: Declaracoes iniciais

:- set_prolog_flag( discontiguous_warnings,off ).
:- set_prolog_flag( single_var_warnings,off ).
:- set_prolog_flag( unknown,fail ).

%_____________________________________________________________________________
% SICStus PROLOG: definicoes iniciais

:- op(900,xfy,'::' ).
:- dynamic filho/2.
:- dynamic natural/2.

% Conhecimento-------------------------

filho(pedro,joao).
filho(raul,joao).
filho(jose,joao).
filho(filipe,pedro).
filho(pedro,ana).
filho(raul,ana).
filho(jose,ana).
filho(rui,jose).
filho(rui,rita).
filho(andre,rui).
filho(andre,maria).
filho(carlos,andre).
filho(marco,andre).

natural(raul,guimaraes).
natural(jose,porto).
natural(ana,lisboa).
natural(carlos,guimaraes).
natural(filipe,braga).

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

%_____________________________________________________________________________
% Extensao do predicado natural: Individuo,Local -> {V,F}

% Invariante Estrutural: nao permitir a insercao de conhecimento
repetido

+natural( I,L ) :: (solucoes( (I,L),(natural( I,L )),S ),
    comprimento( S,N ),
    N == 1).
```

```
% Invariante Referencial: nao admitir mais do que uma naturalidade
para o mesmo individuo

+natural( I,L ) :: ( solucoes( Y, (natural(I,Y)), S ),
    comprimento(S,N),
    N==1).

%_____________________________________________________________________
% Extensao do predicado filho: Filho,Pai -> {V,F}

% Invariante Estrutural:  nao permitir a insercao de conhecimento
repetido

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

% Invariante Referencial: nao admitir mais do que 2 progenitores para
um mesmo individuo

+filho( F,P ) :: ( solucoes( Y, (filho(F,Y)), S ),
    comprimento(S,N),
    N=<2).


% Invariante Referencial: nao admitir que um progenitores seja
descendente so seu filho

+filho( F,P ) :: nao(descendente(P,F,N)).


%_____________________________________________________________________
% Extensao do predicado tios: Individuo, Resultado -> {V,F}

filhos(I,R) :- solucoes(F,filho(F,I),R).


%_____________________________________________________________________
% Extensao do predicado pai: Pai,Filho -> {V,F}

pai(P,F) :- filho(F,P).

%_____________________________________________________________________
% Extensao do predicado tios: Individuo, Resultado -> {V,F}

pais(I,R) :- solucoes(P,pai(P,I),R).

%_____________________________________________________________________
% Extensao do predicado tio: Tio,Sobrinho -> {V,F}

tio(T,S) :- irmao(T,I), pai(I,S).

%_____________________________________________________________________
% Extensao do predicado tios: Individuo, Resultado -> {V,F}

tios(I,R) :- solucoes(T,tio(T,I),S),
    elimRepetidos(S,R).

%tios(I,R) :- solucoes(P,tio(P,I),R).
```

```
%_____________________________________________________________________________
% Extensao do predicado sobrinho: Sobrinho, Tio -> {V,F}

sobrinho(S,T) :- tio(T,S).

%_____________________________________________________________________________
% Extensao do predicado sobrinhos: Individuo, Resultado -> {V,F}

sobrinhos(I,R) :- solucoes(S,sobrinho(S,I),S),
    elimRepetidos(S,R).

%_____________________________________________________________________________
% Extensao do predicado irmao: Irmao1, Irmao2 -> {V,F}

irmao(I1,I2) :- pai(P,I1), pai(P,I2), I1 \== I2.

%_____________________________________________________________________________
% Extensao do predicado irmaos: Individuo, Resultado -> {V,F}

irmaos(I,R) :- solucoes(Ir,irmao(Ir,I),S),
    elimRepetidos(S,R).

%_____________________________________________________________________________
% Extensao do predicado primo: Primo1, Primo2 -> {V,F}

primo(P1,P2) :- filho(P1,Pai1), filho(P2,Pai2), irmao(Pai1,Pai2).

%_____________________________________________________________________________
% Extensao do predicado primos: Individuo, Resultado -> {V,F}

primos(I,R) :- solucoes(P,primo(P,I),S),
    elimRepetidos(S,R).

%_____________________________________________________________________________
% Extensao do predicado avo: Avo, Neto -> {V,F}

avo(A,N) :- filho(N,Y), pai(A,Y).

%_____________________________________________________________________________
% Extensao do predicado avos: Individuo, Resultado -> {V,F}

avos(I,R) :- solucoes(P,avo(P,I),S),
    elimRepetidos(S,R).

%_____________________________________________________________________________
% Extensao do predicado neto: Neto, Avo -> {V,F}

neto(N,A) :- avo(A,N).

%_____________________________________________________________________________
% Extensao do predicado netos: Individuo, Resultado -> {V,F}

netos(I,R) :- solucoes(P,neto(P,I),R).

%_____________________________________________________________________________
% Extensao do predicado bisavo: Bisavo, Bisneto -> {V,F}

bisavo(X,Y) :- avo(X,Z), pai(Z,Y).
```

```
%_____________________________________________________________________________
% Extensao do predicado primos: Individuo, Resultado -> {V,F}

bisavos(I,R) :- solucoes(P,bisavo(P,I),S),
    elimRepetidos(S,R).

%_____________________________________________________________________________
% Extensao do predicado bisneto: Bisneto, Bisavo -> {V,F}

bisneto(B,Y) :- bisavo(Y,B).

%_____________________________________________________________________________
% Extensao do predicado bisnetos: Individuo, Resultado -> {V,F}

bisnetos(Y,R) :- solucoes(P,bisneto(P,Y),S),
    elimRepetidos(S,R).

%_____________________________________________________________________________
% Extensao do predicado casado: Individuo1, Individuo2 -> {V,F}

casado(I1,I2) :- filho(X,I1), filho(X,I2), I1 \== I2.

%_____________________________________________________________________________
% Extensao do predicado descendente: Descendente,Ascendente -> {V,F}

descendente(X,Y) :- filho(X,Y).
descendente(X,Y) :- filho(X,Z), descendente(Z,Y).

%_____________________________________________________________________________
% Extensão do predicado descendentes: Individuo,Grau,Solucao -> {V, F}

descendentes(I, R) :- solucoes(Y, descendente(Y, I), S),
    elimRepetidos(S,R).

%_____________________________________________________________________________
% Extensao do predicado ascendente: Ascendente,Descendente -> {V,F}

ascendente(X,Y) :- descendente(Y,X).

%_____________________________________________________________________________
% Extensão do predicado ascendentes: Individuo,Grau,Solucao -> {V, F}

ascendentes(I, R) :- solucoes(A, ascendente(A, I), S),
    elimRepetidos(S,R).

%_____________________________________________________________________________
% Extensao do predicado descendentegrau: Descendente,Ascendente,Grau -
> {V,F}

descendentegrau(X,Y,1) :- filho(X,Y).
descendentegrau(X,Y,G) :- filho(X,Z), descendentegrau(Z,Y,N), G is
N+1.

%_____________________________________________________________________________
% Extensao do predicado ascendentegrau: Ascendente,Descendente,Grau ->
{V,F}

ascendentegrau(X,Y,G) :- descendentegrau(Y,X,G).
```

```
%_____________________________________________________________________________
% Extensão do predicado descendentesategrau: Individuo,Grau,Solucao ->
{V,F}

descendentesategrau(I,0,[]).
descendentesategrau(I,G,Res) :-
    solucoes(X,descendentegrau(X,I,G),R),
    Y is G-1,
    descendentesategrau(I,Y,R2),
    concat(R,R2,S),
    elimRepetidos(S,Res).

%_____________________________________________________________________________
% Extensão do predicado ascendentesategrau: Individuo,Grau,Solucao ->
{V,F}

ascendentesategrau(I,0,[]).
ascendentesategrau(I,G,Res) :-
    solucoes(X,ascendentegrau(X,I,G),R),
    Y is G-1,
    ascendentesategrau(I,Y,R2),
    concat(R,R2,S),
    elimRepetidos(S,Res).


%_____________________________________________________________________________
% Extensão do predicado relacao: Individuo1, Individuo2, Resultado ->
{V,F}

relacao(X,Y,filho) :- filho(X,Y).
relacao(X,Y,pai) :- pai(X,Y).
relacao(X,Y,tio) :- tio(X,Y).
relacao(X,Y,sobrinho) :- sobrinho(X,Y).
relacao(X,Y,primo) :- primo(X,Y).
relacao(X,Y,irmao) :- irmao(X,Y).
relacao(X,Y,casado) :- casado(X,Y).
relacao(X,Y,avo) :- avo(X,Y).
relacao(X,Y,neto) :- neto(X,Y).
relacao(X,Y,bisavo) :- bisavo(X,Y).
relacao(X,Y,bisneto) :- bisneto(X,Y).
relacao(X,Y,descendente) :- descendente(X,Y).
relacao(X,Y,ascendente) :- ascendente(X,Y).


%_____________________________________________________________________________
% Extensao do predicado que permite a evolucao do conhecimento
insersao: Termo -> {V,F}

insercao(Termo) :- solucoes(Invariante, +Termo::Invariante, Lista),
    evolucao(Termo),
    teste(Lista).

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

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


%_____________________________________________________________________________
```

```
% Extensao do predicado que permite a remoção do conhecimento: Termo -
> {V,F}

remocao(Termo) :- retract(Termo).


%_____________________________________________________________________________
% Extensao do predicado comprimento: Lista, Resultado -> {V,F}

comprimento([],0).
comprimento([C|L], R) :- comprimento(L,G), R is G+1.


%_____________________________________________________________________________
% Extensao do predicado concat: Lista, Resultado -> {V,F}

concat([],L,L).
concat([H|T],L,[H|Y]) :- concat(T,L,Y).


%_____________________________________________________________________________
% Extensao do predicado nao: Questao -> {V,F}

nao(Q) :- Q,!,fail.
nao(Q).


%_____________________________________________________________________________
% Extensão do predicado solucoes: X,Teorema,Solucoes -> {V, F}

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


%_____________________________________________________________________________
% Extensão do predicado elimRepetidos: Lista, Resultado -> {V, F}

elimRepetidos([], []).
elimRepetidos([H|T], R) :- elimElemento(T, H, Raux),
    elimRepetidos(Raux, R2),
    R = [H|R2].


%_____________________________________________________________________________
% Extensão do predicado elimElemento: Lista, Elemento, Resultado ->
{V, F}

elimElemento([], _, []).
elimElemento([H|T], H, R) :- elimElemento(T, H, R).
elimElemento([H|T], E, R)     :- H \== E,
    elimElemento(T, E, R2),
    R = [H|R2].
```