RELATÓRIO PROJETO PROLOG - TURMA IF972

Grupo: Luiz Guylherme -> lgar@cin.ufpe.br
Gabriel Laroche -> glb@cin.ufpe.br
Valter Jose -> vjsj@cin.ufpe.br

Descrição do Problema

O problema é descrito como um problema familiar com combinações improváveis de casamentos, em que há 6 indivíduos, vamos usar os nomes traduzidos para o português para ficar mais fácil a compreensão. Os indivíduos são:

- Eu;
- Meu pai;
- A viúva;
- A ruiva;
- Meu bebê;
- Onrun;

Nisso se inicia a problemática, pois a viúva é mãe da ruiva, e “Eu” se casa com a viúva enquanto “Meu Pai” se casa com a ruiva. Após isso, “Eu” tem um filho com a viúva que é “Meu bebê”, enquanto “Meu Pai” tem um filho com a ruiva, que é o Onrun. Nisso, as coisas se complicam, pois “Eu” vira avô dele mesmo e padrasto de “Meu pai”, além de várias outras relações que demoram um pouco para entender. Sabendo disso, desenvolvemos um programa em prolog que retorna para o usuário se a relação entre os integrantes da família apresentada no input é verdadeira ou não, para assim descobrir quem é pai de quem, quem é tio de quem e etc.

Destaques da Implementação

Para implementar o código em prolog que fosse funcional para encontrar todas as relações existentes na família, optamos por adicionar uma regra stepchild, que funciona adicionando todas aquelas relações de pai e filho e mãe filho que não foram ditas nos fatos e foram geradas a partir dos casamentos entre “Eu”-"Viúva" e “Meu Pai”-“Ruiva”. Com essa regra criada, só precisamos adicionar essa nova relação de stepchild nas regras de Pai, Mãe, Avô, Avó, neto, etc.

Vejamos agora um exemplo de implementação da regra “father” que encontra quem é o pai de alguém, por meio do trace, para o exemplo abaixo colocaremos para o nosso verificar se “Eu” é pai de “Meu Pai”, veja como fica:

father(i,dad).
Call: (7) father(i, dad) ? creep
Call: (8) child(dad, i) ? creep
Fail: (8) child(dad, i) ? creep
Redo: (7) father(i, dad) ? creep
Call: (8) stepchild(dad, i) ? creep
Call: (9) spouse(i, _G6360) ? creep
Exit: (9) spouse(i, widow) ? creep
Call: (9) child(dad, widow) ? creep
Fail: (9) child(dad, widow) ? creep
Redo: (8) stepchild(dad, i) ? creep

Call: (9) spouse(i, _G6360) ? creep
Exit: (9) spouse(i, widow) ? creep
Call: (9) child(_G6359, widow) ? creep
Exit: (9) child(redhair, widow) ? creep
Call: (9) spouse(redhair, dad) ? creep
Exit: (9) spouse(redhair, dad) ? creep
Exit: (8) stepchild(dad, i) ? creep
Call: (8) male(i) ? creep
Exit: (8) male(i) ? creep
Exit: (7) father(i, dad) ? creep
true

Veja que primeiro ele verifica se já existe nos fatos a relação de “Meu Pai” ser filho de “Eu”, porém verifica que não há, com isso ele testa de novo, verificando agora se “Meu Pai” é stepchild de “Eu”, chamando a regra stepchild, e ao terminar todo o processo no stepchild, ele retorna que é verdadeiro que “Meu Pai” é stepchild de “Eu”, com isso basta agora ver se “Eu” é um homem para que ele seja pai, e assim retornando no final do processo o “true”, provando que “Eu” é pai de “Meu Pai”.

Veremos agora como será a implementação da regra uncles(Resp), que nos retorna uma lista com todas as relações de Tio e sobrinhos ou sobrinhas, a regra tem o seguinte esqueleto:

uncles(Resp) :- setof((X,Y), uncle(X,Y), Resp).

O que ela faz é criar uma lista, em que X é tio de Y, e pegar a informação da regra uncle(X,Y). Essa regra uncle(X,Y) funciona da seguinte maneira: primeiro ela vai verificar se existe um irmão de X, e depois verificar se o irmão de X tem um filho, olhando para os fatos e a regra de stepchild, por fim se as duas exigências estiverem verdadeiras, então, X é tio de Y.

Por fim, veja como fica a lista de tios e sobrinhos.

Resp = [(baby, i), (baby, onrun), (baby, widow), (dad, i), (dad, onrun), (dad, widow), (i, baby), (i, dad), (..., ...)|...].

Versão em Lógica de Predicados

$(\forall X)\ (\forall Y).\ (\text{stepchild}(X,Y) \Leftarrow [\text{spouse}(Y,Z) \wedge \text{child}(X,Z)]\ \mathbf{v}\ [\text{spouse}(Y,Z) \wedge \text{child}(W,Z) \wedge \text{spouse}(W,X)]\ \mathbf{v}\ [\text{child}(Z,Y) \wedge \text{spouse}(Z,X)]\ )$

$(\forall X)\ (\forall Y).\ (\text{father}(X, Y) \Leftarrow [\text{child}(Y, X) \wedge \text{male}(X)]\ \mathbf{v}\ [\text{stepchild}(Y,X) \wedge \text{male}(X)]\ )$

$(\forall X)\ (\forall Y).\ (\text{mother}(X, Y) \Leftarrow [\text{child}(Y, X) \wedge \text{female}(X)]\ \mathbf{v}\ [\text{stepchild}(Y,X) \wedge \text{female}(X)]\ )$

$(\forall X)\ (\forall Y).\ (\text{son}(X, Y) \Leftarrow [\text{child}(X, Y) \wedge \text{male}(X)]\ \mathbf{v}\ [\text{stepchild}(X, Y) \wedge \text{male}(X)]\ )$

(∀X) (∀Y). (daugther(X, Y) ⇐ [child(X, Y) ∧ female(X)] **v** [stepchild(X, Y) ∧ female(X)])

(∀X) (∀Y). (brother(X, Y) ⇐ [male(X) ∧ child(X, Z) ∧ child(Y, Z) ∧ ¬(X=Y)] **v** [male(X) ∧ stepchild(X,Z) ∧ child(Y,Z) ∧ ¬(X=Y)] **v** [male(X) ∧ stepchild(X,Z) ∧ stepchild(Y,Z) ∧ ¬(X=Y)])

(∀X) (∀Y). (sister(X, Y) ⇐ [female(X) ∧ child(X, Z) ∧ child(Y, Z) ∧ ¬(X=Y)] **v** [female(X) ∧ stepchild(X,Z) ∧ child(Y,Z) ∧ ¬(X=Y)] **v** [female(X) ∧ stepchild(X,Z) ∧ stepchild(Y,Z) ∧ ¬(X=Y)])

(∀X) (∀Y). (uncle(X, Y) ⇐ [brother(X, Z) ∧ child(Y, Z)] **v** [brother(X, Z) ∧ stepchild(Y, Z)])

(∀X) (∀Y). (grandparent(X, Y) ⇐ [child(Z, X) ∧ child(Y, Z)] **v** [stepchild(Z,X) ∧ child(Y,Z)] **v** [child(Z,X) ∧ stepchild(Y,Z)] **v** [stepchild(Z,X) ∧ stepchild(Y,Z)])

(∀X) (∀Y). (grandfather(X, Y) ⇐ [male(X) ∧ child(Z, X) ∧ child(Y, Z)] **v** [male(X) ∧ stepchild(Z,X) ∧ child(Y,Z)] **v** [male(X) ∧ child(Z,X) ∧ stepchild(Y,Z)] **v** [male(X) ∧ stepchild(Z,X) ∧ stepchild(Y,Z)])

(∀X) (∀Y). (grandmother(X, Y) ⇐ [female(X) ∧ child(Z, X) ∧ child(Y, Z)] **v** [female(X) ∧ stepchild(Z,X) ∧ child(Y,Z)] **v** [female(X) ∧ child(Z,X) ∧ stepchild(Y,Z)] **v** [female(X) ∧ stepchild(Z,X) ∧ stepchild(Y,Z)])

(∀X) (∀Y). (grandchild(X, Y) ⇐ [child(X,Z) ∧ child(Z,Y)] **v** [stepchild(X,Z) ∧ child(Z,Y)] **v** [child(X,Z) ∧ stepchild(Z,Y)] **v** [stepchild(X, Z) ∧ stepchild(Z,Y)])

Conclusão

Podemos concluir, que através da recursividade utilizada nas regras as relações conseguem ser verificadas e validadas quando os parâmetros são passados, aquela também é responsável pelas repetições pois ao verificar a parentalidade dos parâmetros as relações in_the_law são chamadas novamente fazendo com que haja duplicidades. No mais, podemos perceber que as funções implementadas possuem poucas falhas o que se torna um destaque na sua implementação e conseguimos retornar true, que sim, “Eu” é avó dele mesmo.

Bibliografia utilizada

Para o desenvolvimento deste projeto, utilizamos apenas os slides de prolog e o pdf que indica como o projeto deve ser feito.