Relatório de Lógica para Computação

Baseado na música “I’m my Own Grandpa”

Alunos:

Isabelle Queiroz Gomes de Assis (iqga@cin.ufpe.br) ;
Luis Felipe Araujo Mota (lfam@cin.ufpe.br) ;
Paulo Vitor Alves de Oliveira (pvao@cin.ufpe.br) .

1. Objetivo

Aplicar os conhecimentos adquiridos em sala de aula sobre lógica em um projeto elaborado para representação de conhecimento. Com o objetivo de um aprendizado divertido e intuitivo, entender as relações entre fatos e regras, bem como sua aplicação no dia a dia, utilizando os conceitos aprendidos sobre lógica de predicados e o ferramental para aplicação das regras com o Prolog.

Ao final da execução do relatório devemos ter um código que abrange corretamente todas as relações familiares descritas na canção que foi definida como problemática do projeto.

2. Descrição do Problema

Utilizando a música “I’m my own Grandpa” aplicar o que aprendemos sobre lógica para transcrever as relações e fatos existentes na letra da música para entender melhor o contexto. O problema gira em torno dos fatos relativos à família do narrador, sendo assim nosso projeto irá ser feito a fim de trazer à tona tais relações de modo simples e direto, além de ser possível compreender relações novas que nem sequer imaginaríamos, isto é, novas fontes de conhecimento que antes não apareciam na música.

3. Destaques da Implementação

A partir dos seguintes fatos iniciais iremos traçar o caminho a ser planejado e transcrito:

```
child(redhair, widow).
child(i, dad).
child(onrun, dad).
child(baby, i).

male(i).
male(dad).
male(onrun).
```

male(baby).

female(redhair).
female(widow).

spouse(i, widow).
spouse(widow, i).
spouse(dad, redhair).
spouse(redhair, dad).

Com os fatos iniciais em mãos partimos para a criação das regras a fim de construir as relações familiares existentes na música. Assim, criamos regras para verificar se a pessoa é filho, filha, pai, mãe, irmão, irmã, tio, tia, neto, avô ou avó de alguém. Tais fatos estão presentes no arquivo.pl enviado, mas também iremos anexar uma foto aqui para facilitar a compreensão na explicação.

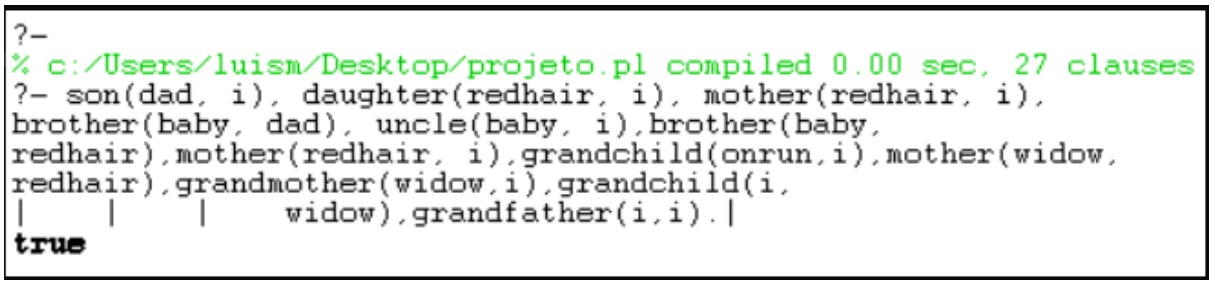
```
% Regras
son(X, Y) :-
male(X), % X é homem
(
(child(X,Y)) ; % X é filho de Y
(spouse(Y, Z), child(X,Z)) ; % X é filho do spouse de Y
(child(C, Y), spouse(C,X)) ; % X é casado com o filho de Y
(child(D,Z), spouse(D,X), spouse(Y,Z)) % X é casado com o filho do spouse de Y
).

daughter(X, Y) :-
female(X), % X é mulher
(
(child(X,Y)) ; % X é filha de Y
(spouse(Y, Z), child(X,Z)) ; % X é filha do spouse de Y
(child(C, Y), spouse(C,X)) ; % X é casada com o filho de Y
(child(D,Z), spouse(D,X), spouse(Y,Z)) % X é casada com o filho do spouse de Y
).

father(X, Y) :-
male(X), % X é homem
(son(Y,X) ; daughter(Y,X)). % Y é filhx de X

mother(X, Y) :-
female(X), % X é mulher
(son(Y,X) ; daughter(Y,X)). % Y é filhx de X

brother(X, Y) :-
son(X, A), % X é filho de A
(son(Y,A); daughter(Y,A)), % Y é filhx de A
not(X = Y). % X é diferente de Y

sister(X, Y) :-
daughter(X, A), % X é filha de A
(son(Y,A); daughter(Y,A)), % Y é filhx de A
not(X = Y). % X é diferente de Y

uncle(X, Y) :-
male(X), brother(X,A), % X é homem e é irmão de A
(son(Y,A); daughter(Y,A)). % Y é filhx de A
```

```
aunt(X, Y) :-
female(X), sister(X,A), % X é mulher e é irmã de A
(son(Y,A); daughter(Y,A)). % Y é filhx de A

grandchild(X, Y) :-
(son(X, A); daughter(X, A)), % X é filhx de A
(son(A, Y); daughter(A, Y)). % A é filhx de Y

grandfather(X, Y) :-
male(X), % X é homem
grandchild(Y,X). % Y é netx de X

grandmother(X, Y) :-
female(X), % X é mulher
grandchild(Y,X). % Y é netx de X
```

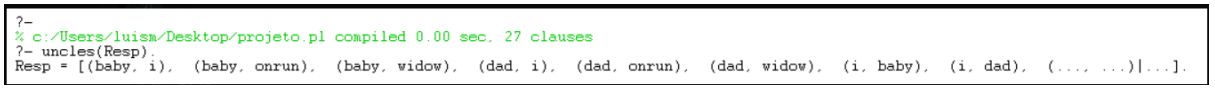
Todos esses casos fazem com que seja possível traçar qualquer tipo de relação existente entre alguém nessa família, assim, com os fatos iniciais e as regras criadas podemos finalmente compreender todas as relações familiares e até mesmo outras que possam ser difíceis de enxergar.

Com isso em mente, podemos calcular ou checar casos para ver se realmente é verdade, como o exemplo uncles(Resp) e aunts(Resp), em que deve nos retornar uma lista de todos os pares sem repetições de pessoas que são tio ou tia de alguém. Por exemplo: (A, B), em que A é tio(a) de B; (C, D), em que C é tio(a) de D, etc.

A foto a seguir mostra a veracidade da consulta necessária do projeto, isto é, a capacidade de dizer se é verdade ou não toda a consulta a seguir a partir dos fatos e regras colocados no arquivo. Como solicitado, ele responde “True” corretamente demonstrando que está tudo correto.



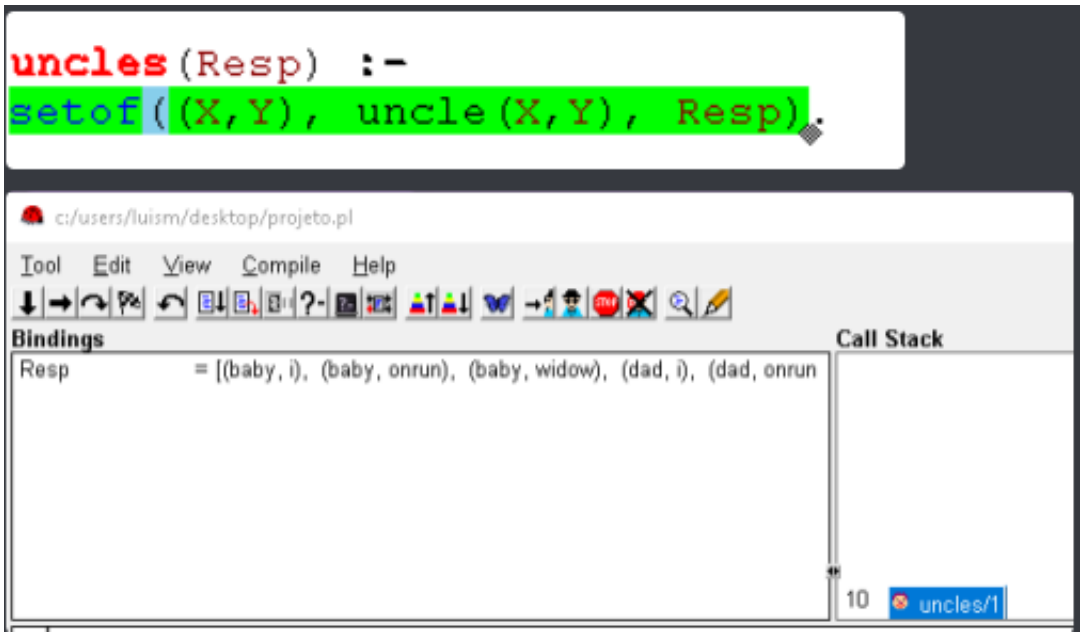
Por fim, segue o principal exemplo solicitado do uncles(Resp), sendo ele a capacidade de mostrar todas as relações existentes após chamar essa função:



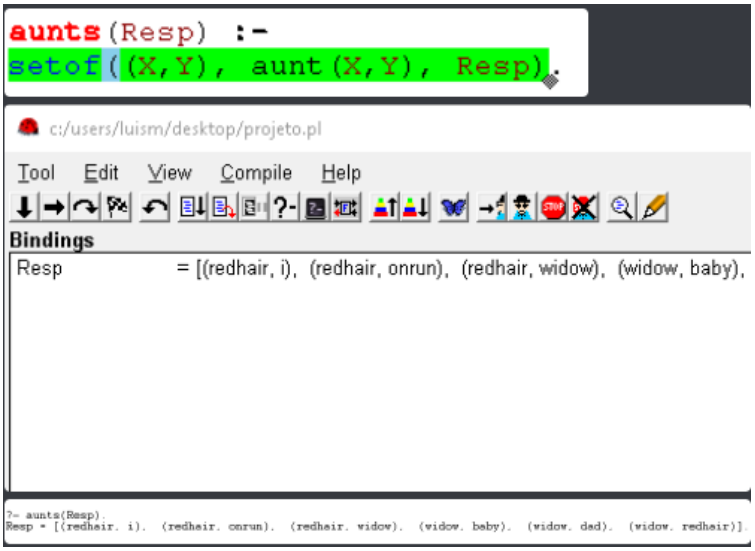
Assim, o programa consegue listar todos os pares existentes com a chamada do uncles(Resp), demonstrando que todas essas relações são verdadeiras. Para comprovar segue a imagem a seguir em que testamos uma por uma:

```
?-
% c:/Users/luism/Desktop/projeto.pl compiled 0.00 sec, 27 clauses
?- uncles(Resp).
Resp = [(baby, i), (baby, onrun), (baby, widow), (dad, i), (dad, onrun), (dad, widow

?- uncle(baby,i).
true .

?- uncle(baby,onrun).
true .

?- uncle(baby,widow).
true .

?- uncle(dad,i).
true .

?- uncle(dad,onrun).
true .
```

Podemos comprovar também utilizando o modo debug (trace) para mostrar o passo a passo do caminho percorrido pelo Prolog até achar toda a lista de pares. Primeiro ele analisa todas as possibilidades existentes, ou seja, os casos do uncle, e logo depois analisa se há repetições ou não. Com isso, o programa vai dando append nesses pares em uma lista checando se o elemento já foi inserido ou não, evitando repetições. Assim, no final iremos ter uma lista de todos os pares em que a consulta é “True” para a chamada do predicado uncle(resp).



Como exemplo extra, trouxemos o caso de aunts(Resp) também, visando demonstrar o que ele nos daria como resposta caso pedíssemos a lista de pares voltadas para a tia. A lógica será a mesma do uncles(Resp), isto é, colocando os pares em uma lista checando se há repetição ou não. Por fim, iremos ter a lista presente na imagem a seguir, o que acaba sendo diferente do uncles.



4. Versão em Lógica de Predicados

Conectivos para aplicação em Lógica de Predicados:

V → para todos
E → existe
v → ou
/\ → e
⇒ → implica
S, M, F… → predicado
a, b, c… → objeto
~ → não

Predicados para aplicação em Lógica de Predicados:

S (x,y) → x é filho(a) de y
M (x) → x é homem
F (x) → x é mulher
C(x,y) → x é cônjuge de y

Fatos iniciais em Lógica de Predicados:

a = redhair; b = widow; c = dad
d = onrun; e = baby; i = eu

child(redhair, widow). → (Ex)(Ey) S(a,b) /\ ((x = a) /\ (y = b))
child(i, dad). → (Ex)(Ey) S(i,c) /\ ((x = i) /\ (y = c))
child(onrun, dad). → (Ex)(Ey) S(d, c) /\ ((x = d) /\ (y = c))
child(baby, i). → (Ex)(Ey) S(e, i) /\ ((x = e) /\ (y = i))

male(i). → (Ex) M(i) /\ (x = i)
male(dad). → (Ex) M(c) /\ (x = c)
male(onrun). → (Ex) M(d) /\ (x = d)
male(baby). → (Ex) M(e) /\ (x = e)

female(redhair). → (Ex) F(a) /\ (x = a)
female(widow). → (Ex) F(b) /\ (x = b)

spouse(i, widow). → (Ex)(Ey) C(i, b) /\ ((x = i) /\ (y = b))
spouse(widow, i). → (Ex)(Ey) C(b, i) /\ ((x = b) /\ y = i))
spouse(dad, redhair). → (Ex)(Ey) C(c, a) /\ ((x = c) /\ (y = a))
spouse(redhair, dad). → (Ex)(Ey) C(a , c) /\ (x = a) /\ (y = c))

Regras criadas anteriormente transcritas para Lógica de Predicados:

son(X, Y): Son(x, y) ⇒ (Vx)(Vy)(Vz)(Vc)(Vd) M(x) /\ ((S(x, y)) v (C(y, z) /\ S(x, z)) v (S(c, y) /\ C(c, x)) v (S(d, z) /\ C(d, x) /\ C(y, z))

daughter(X, Y): Daughter(x, y) ⇒ (Vx)(Vy)(Vz)(Vc)(Vd) F(x) /\ ((S(x, y)) v (C(y, z) /\ S(x, Zz)) v (S(c, y) /\ C(c, x)) v (S(d, z) /\ C(d, x) /\ C(y, z))

father(X, Y): Father(x, y) ⇒ (Vx)(Vy) M(x) /\ (Son(y, x) v Daughter(y, x))

mother(X, Y): Mother(x, y) ⇒ (Vx)(Vy) F(x) /\ (Son(y, x) v Daughter(y, x))

brother(X, Y): Brother(x, y) ⇒ (Vx)(Vy)(Va) Son(x, a) /\ (Son(y, a) v Daughter(y, a)) /\ ~(x = y)

sister(X, Y): Sister(x, y) ⇒ (Vx)(Vy)(Va) Daughter(x, a) /\ (Son(y, a) v Daughter(y, a)) /\ ~(x = y)

uncle(X, Y): Uncle(x, y) ⇒ (Vx)(Vy)(Va) M(x) /\ Brother(x, a) /\ (Son(y , a) v Daughter(y, a))

aunt(X, Y): Aunt(x, y) ⇒ (Vx)(Vy)(Va) F(x) /\ Brother(x, a) /\ (Son(y, a) v Daughter(y, a))

grandchild(X, Y): Grandchild(x, y) ⇒ (Vx)(Vy)(Va) (Son(x, a) v Daughter(x, a)) /\ (Son(a, y) v Daughter(a, y))

grandfather(X, Y): Grandfather(x, y) ⇒ (Vx)(Vy) M(x) /\ Grandchild(y, x)

grandmother(X, Y): Grandmother(x, y) ⇒ (Vx)(Vy) F(x) /\ Grandchild(y, x)

5. Conclusão

Por meio desse projeto foi possível ver como é simples tratar de relações lógicas e confusas quando utilizamos um ferramental voltado para essa área, isto é, conseguimos traçar facilmente as relações familiares do narrador, bem como interpretar e revelar outras relações que estavam ali escondidas ou não tão visíveis. Com isso, o objetivo de aplicar os conhecimentos de lógica juntamente do Prolog para análise e compreensão dos fatos e regras foi um sucesso, como demonstrado anteriormente por meio das fotos e explicações.

Poderíamos continuar aprimorando e colocando relações mais avançadas ou exemplos mais precisos, porém acabaria ficando muito longo e já conseguimos demonstrar a eficiência através dos exemplos. Sempre podemos ir aperfeiçoando ou tratando de outras situações que possam ser facilmente resolvidas utilizando a lógica. Até mesmo casos em que uma frase possa parecer ambígua, sua transcrição para lógica de predicados ou até mesmo para o Prolog é facilitada e sua interpretação se torna simples.

6. Bibliografia

Programa utilizado:
https://www.swi-prolog.org/

Informações adicionais:
https://www.cin.ufpe.br/~srmq/prolog/

Vídeo adicional ajudando com modo debug:
https://www.youtube.com/watch?v=-OnE4Owf7xE

Vídeo ajudando a compreender melhor a letra da música:
https://drive.google.com/file/d/19oY_W_2WSDPpdIntc3KLO5zk18REjM5o/view