Title:	Modeling Genealogy in Prolog
Date:	08/04/2023
By:	Ethan Corgatelli

1 Table of Contents

2 Abstract

This paper describes a knowledge base I built in SWI-Prolog that models one side of my family tree over three generations. All the names in the knowledge base except for my own have been changed for privacy. Everyone's gender is defined with atomic facts, and `child(X, Y)` is the primitive relationship used.

Relational facts for the all of the following family relationships are implemented (both general and gender-specific versions): **child**, **sibling**, **partner**, **parent**, **grandparent**, **grandchild**, **aunt/uncle**, **nephew/niece**, **cousin**, and the most general: **related**, which encompasses all the others.

Because of the structure I chose, initially, some queries would list unnecessary/unwanted duplicate results. This led me to add "pre" and "strict" versions of some of the relational facts, allowing for cleaner results (particularly while trying to list all matches for a specific relationship).

3 Description

The forms of atomic/basic facts in this knowledge base are:
`male(name)` / `female(name)` / `child(child_name, parent_name)`.
For example, here are all the atomic facts about me:

`male(ethan).`	`child(ethan, shane).`	`child(ethan, alma).`

The form of relational/general facts in this knowledge base is: `relationship(X, Y)`.
For example, here is the fact that defines a parent: `parent(X, Y) :- child(Y, X)`.
Here is the fact that defines a father: `father(X, Y) :- male(X), parent(X, Y)`.

There is one non-atomic fact in the knowledge base that doesn't take the form of the above rules: `person(X) :- male(X) ; female(X)`, which no other rule references.

There are a few general facts that would originally yield unwanted duplicates in the results, because there are multiple valid solutions that lead to the same variable assignments. For example: the sibling relationship is defined by two people who have a parent in common. Because everyone has two parents, the results were doubled. To fix this, I split the relational facts with this issue into several versions: `relationship_pre`, `relationship`, and `relationship_strict`. The "pre" variants represent the most general forms of the relationships, and lists duplicates when given general queries (queries with variables). The normal variants are constructed using the "pre" variants and `distinct()` in order to remove duplicate pairs of results. Finally, the "strict" variants remove half of the results from the normal variants by filtering on results that are in order according to the standard ordering (this means instead of both `r(X, Y)` and `r(Y, X)` appearing, only `r(X, Y)` will appear, which is useful for quickly visualizing all the relationships of a certain type with a general query).

4.0 Results

This section contains some sample queries and the results from those queries.

person(X)

X	
kevin	1
shane	2
milton	3
arthur	4
terence	5
ethan	6
marvin	7
jody	8
bryan	9
luke	10
april	11
alma	12
alyssa	13
paula	14
candice	15
bernadette	16
angela	17

related(ethan, X)

X	
shane	1
alma	2
candice	3
kevin	4
april	5
milton	6
arthur	7
terence	8
alyssa	9
paula	10
marvin	11
jody	12
bernadette	13
angela	14
bryan	15
luke	16

parent(X, Y)

X	Y	
kevin	shane	1
kevin	milton	2
kevin	arthur	3
kevin	terence	4
april	shane	5
april	milton	6
april	arthur	7
april	terence	8
shane	ethan	9
shane	candice	10
alma	ethan	11
alma	candice	12
milton	marvin	13
milton	jody	14
milton	bernadette	15
milton	angela	16
alyssa	marvin	17
alyssa	jody	18
alyssa	bernadette	19
alyssa	angela	20
arthur	bryan	21
arthur	luke	22
paula	bryan	23
paula	luke	24

sister(X, Y)

X	Y	
candice	ethan	1
bernadette	marvin	2
bernadette	jody	3
bernadette	angela	4
angela	marvin	5
angela	jody	6
angela	bernadette	7

sibling_strict(X, Y)

X	Y	
shane	milton	1
shane	arthur	2
milton	arthur	3
terence	shane	4
terence	milton	5
terence	arthur	6
ethan	candice	7
marvin	jody	8
marvin	bernadette	9
marvin	angela	10
jody	bernadette	11
jody	angela	12
bernadette	angela	13
luke	bryan	14

parent(X, ethan)

X	
shane	1
alma	2

son(ethan, X)

X	
shane	1
alma	2

wife(alma, X)

X	
shane	1

grandparent(X, Y)

X	Y	
kevin	ethan	1
kevin	candice	2
kevin	marvin	3
kevin	jody	4
kevin	bernadette	5
kevin	angela	6
kevin	bryan	7
kevin	luke	8
april	ethan	9
april	candice	10
april	marvin	11
april	jody	12
april	bernadette	13
april	angela	14
april	bryan	15
april	luke	16

grandchild(X, Y)

X	Y	
ethan	kevin	1
candice	kevin	2
marvin	kevin	3
jody	kevin	4
bernadette	kevin	5
angela	kevin	6
bryan	kevin	7
luke	kevin	8
ethan	april	9
candice	april	10
marvin	april	11
jody	april	12
bernadette	april	13
angela	april	14
bryan	april	15
luke	april	16

cousin_strict(X, Y)

X	Y	
marvin	ethan	1
marvin	candice	2
jody	ethan	3
jody	candice	4
luke	ethan	5
luke	candice	6
ethan	bernadette	7
ethan	angela	8
candice	bernadette	9
candice	angela	10
bryan	bernadette	11
bryan	angela	12
luke	jody	13
luke	bernadette	14
luke	angela	15
ethan	bryan	16
candice	bryan	17
marvin	bryan	18
marvin	luke	19
jody	bryan	20

granddaughter(X, Y)

X	Y	
candice	kevin	1
candice	april	2
bernadette	kevin	3
bernadette	april	4
angela	kevin	5
angela	april	6

nephew(marvin, X)

X	
shane	1
arthur	2
terence	3
alma	4
paula	5

aunt(alma, X)

X	
marvin	1
jody	2
bernadette	3
angela	4
bryan	5
luke	6

cousin(*candice*, X)

X	
marvin	1
jody	2
bernadette	3
angela	4
bryan	5
luke	6

grandchild(*jody*, X)

X	
kevin	1
april	2

granddaughter(X, Y)

X	Y	
candice	kevin	1
candice	april	2
bernadette	kevin	3
bernadette	april	4
angela	kevin	5
angela	april	6

5.0 Discussion

I'm quite happy with what I've built, as it does exactly what I've designed it to do; it captures knowledge about all of my family relationships over the last 3 generations on my Dad's side.

Given the atomic relationships that I used, there are almost no questions that my knowledge base should be able to answer but can't (the exception being in-laws). Querying is very convenient, with different rules for whether or not both versions of a relationship should be listed. It is also relatively easy to add new rules to this knowledge base (with the caveat of needing to add a few rules for some new relationships for convenience while querying).

There are a few ways in which it could be improved. Firstly, it contains no concept of relationships that result from divorce/remarriage. Although it would be a bit complicated to model such relationships along with temporal information in Prolog, a simple implementation could certainly be implemented. Secondly, it only spans over 3 generations, and therefore doesn't contain any "great" relationships.

6.0 Appendix: Full Knowledge Base

```
% ---- BASIC FACTS ---- %
% males
male(kevin).
male(shane).
male(milton).
male(arthur).
male(terence).
male(ethan).
male(marvin).
male(jody).
male(bryan).
male(luke).

% females
female(april).
female(alma).
female(alyssa).
female(paula).
female(candice).
female(bernadette).
female(angela).

% kevin & april's family
child(shane, kevin).
child(milton, kevin).
child(arthur, kevin).
child(terence, kevin).
```

```
% ---- BASIC FACTS (cont.) ---- %
child(shane, april).
child(milton, april).
child(arthur, april).
child(terence, april).

% shane & alma's family
child(ethan, shane).
child(candice, shane).
child(ethan, alma).
child(candice, alma).

% milton & alyssa's family
child(marvin, milton).
child(jody, milton).
child(bernadette, milton).
child(angela, milton).
child(marvin, alyssa).
child(jody, alyssa).
child(bernadette, alyssa).
child(angela, alyssa).

% arthur & paula's family
child(bryan, arthur).
child(luke, arthur).
child(bryan, paula).
child(luke, paula).
```

```
% ---- RELATIONSHIP DEFINITIONS ---- %
person(X)            :- male(X) ; female(X).

son(X, Y)            :- male(X), child(X, Y).
daughter(X, Y)       :- female(X), child(X, Y).

sibling_pre(X, Y)    :- child(X, Z), child(Y, Z), X \== Y.
sibling(X, Y)        :- distinct(sibling_pre(X, Y)).
brother(X, Y)        :- male(X), sibling(X, Y).
sister(X, Y)         :- female(X), sibling(X, Y).
sibling_strict(X, Y) :- sibling(X, Y), X @> Y.
```

```
partner_pre(X, Y)    :- child(Z, X), child(Z, Y), X \== Y.
partner(X, Y)        :- distinct(partner_pre(X, Y)).
husband(X, Y)        :- male(X), partner(X, Y).
wife(X, Y)           :- female(X), partner(X, Y).
partner_strict(X, Y) :- partner(X, Y), X @> Y.

parent(X, Y)         :- child(Y, X).
father(X, Y)         :- male(X), parent(X, Y).
mother(X, Y)         :- female(X), parent(X, Y).

grandparent(X, Y)    :- parent(X, Z), parent(Z, Y).
grandfather(X, Y)    :- male(X), grandparent(X, Y).
grandmother(X, Y)    :- female(X), grandparent(X, Y).

grandchild(X, Y)     :- grandparent(Y, X).
grandson(X, Y)       :- male(X), grandchild(X, Y).
granddaughter(X, Y)  :- female(X), grandchild(X, Y).

uncle_aunt(X, Y)     :- sibling(X, S), child(Y, S).
uncle_aunt(X, Y)     :- partner(X, P), sibling(P, S), child(Y, S).
uncle(X, Y)          :- male(X), uncle_aunt(X, Y).
aunt(X, Y)           :- female(X), uncle_aunt(X, Y).

nephew_niece(X, Y)   :- uncle_aunt(Y, X).
nephew(X, Y)         :- male(X), nephew_niece(X, Y).
niece(X, Y)          :- female(X), nephew_niece(X, Y).

cousin_pre(X, Y)     :- nephew_niece(X, UA), child(Y, UA).
cousin(X, Y)         :- distinct(cousin_pre(X, Y)).
cousin_strict(X, Y)  :- cousin(X, Y), X @> Y.

related(X, Y) :- distinct(
    child(X, Y);
    sibling(X, Y);
    parent(X, Y);
    grandparent(X, Y); grandchild(X, Y);
    uncle_aunt(X, Y); nephew_niece(X, Y);
    cousin(X, Y)
).
```