CS 470: Project 3 - Genealogy Knowledge Base

University of Idaho
Matthew Waltz

2nd May 2018

Abstract

This project implements a Prolog knowledge base which can look up the gene relationships between characters in the hit TV series Game of Thrones as of Season 8. With about 50 listed members of the tree, defined relationships can vary from queries such as 'son' and more relative terms such as 'ancestor' and 'descendant'. The maximum depth of the tree is five generations. The implementation works by taking a list of facts consisting of parents and genders for each individual in order to create the more complex rules and definitions.

Contents

1 Summary

The knowledge base of this project uses the two facts: parents and gender in order to build the list. In prolog, a parent is defined as "parent(a, b)." with "a" being the parent of the child "b".

The two gender facts used are male and female, when written in prolog appear as "male(a)." and "female(a)." In this case, "a" relates to the individual that was previously defined in one of the parental relationships.

The above base facts are then used to create the more complex rules in the knowledge base. Furthermore, rules can use other rules in order to continue the levels of abstraction. For instance, a sibling would consist of two children with the same parent, and are not the same person. Using the "sibling" rule allows us to easily create the "sister" rule, which then further checks if the query is female.

The "related" rule is implemented simply if two people share the same "ancestor" query, which is defined below.

The "ancestor" query in the knowledge database uses an interesting recursive search method to iterate over all of the parental relationships in order to determine a match. Of course, the implementation of the "descendants" interface merely searches in the inverse direction, flipping the search tree in order to determine the descendants of an individual.

2 Relationships

The syntax of the rules for relationships follows a simple request in the form (<var,name>, <query name>), where "var" can be a capitalized variable in Prolog used for searching, or a "name" in order to return a boolean value check. "query name" relates to the search pattern. This syntax is appended to the end of any of the names of the rule definition relationships.

Available relationships for use in the knowledge base is listed here:

parent, child, sibling, related, descendant, ancestor, mother, father, son, daughter, brother, sister, grandparent, grandfather, grandmother, grandchild, greatgrandparent, greatgrandchild, grandson, granddaughter, uncle, aunt, cousin, nephew, niece, secondcousin

3 Results

Query the father of Jon Snow:

```
| ?- father(A, snow).

A = rhaegar ? ;

no
```

Query the second cousins of Stannis Baratheon:

```
| ?- secondcousin(A, stannis).

A = rhaegar ? ;

A = viserys ? ;

A = daenerys ? ;

(4 ms) no
```

Query the ancestors of Robin of House Arryn:

```
| ?- ancestor(A, robin).

A = lysa ? ;

A = jon ? ;

A = hoster ? ;

A = minisa ? ;

A = robbyrt ? ;

no
```

List all of the mothers in the knowledge base:

```
| ?- mother(A, B).

A = joanna
```

```
B = jaime ? ;

A = joanna
B = cersei ? ;

A = joanna
B = tyrion ? ;

A = cersei
B = joffrey ? ;

A = cersei
B = myrcella ? ;

A = cersei
B = tommen ? ;

A = rhaelle
B = steffon ? ;

A = lyanna
B = snow ? ;

A = minisa
B = lysa ? ;

A = minisa
B = edmure ? ;

A = minisa
B = catelyn ? ;

A = catelyn
B = robb ? ;

A = catelyn
B = sansa ? ;

A = catelyn
B = arya ? ;

A = catelyn
B = brandon ? ;

A = catelyn
```

```
B = rickon ? ;

A = lysa
B = robin ? ;

A = olenna
B = mace ? ;

no
```

Check if Jon Snow and Daenerys Targarean are related (to keep the recursive rule simple, the output may be printed multiple times based on the depth of ancestors).

```
| ?- related(snow, daenerys).

true ? ; true ? ; true ? ; true ? ;

no
```

Query if Stannis Baratheon has any uncles:

```
| ?- uncle(A, stannis).

no
```

4 Discussion

The knowledge base performs quite well at gene-related (biological) relationships. However, it cannot represent non-biological relationships that well, either from a child outside of a marriage and/or adoption. It does quite well considering the incest of Game of Thrones though. Most queries can be answered with a simple or complex rule. Adding new rules is fairly simple, appending to the current list which can use the other available rules quickly makes it quite powerful. It would be nice to have a new rule and/or fact that is able to define adopted children in some way. I found that Prolog made this extremely easy, based on the logical processing that it entails. All in all, the knowledge base performs much better than I was initially thinking, and it makes sorting the huge tree a lot easier to understand.

One thing I found interesting is how easy it is to form queries. If I want everything in the knowledge base, I can just use two variables, i.e. parent(A, B). to list all of the parents. That was pretty neat.

5 Code Appendix

```
/* prolog family tree for game of thrones  */
/* matt waltz  */
/* spring 2018  */

/* house lannister */

parent(tywin, jaime).
parent(tywin, cersei).
parent(tywin, tyrion).

parent(joanna, jaime).
parent(joanna, cersei).
parent(joanna, tyrion).

parent(jaime, joffrey).
parent(jaime, myrcella).
parent(jaime, tommen).

parent(cersei, joffrey).
parent(cersei, myrcella).
parent(cersei, tommen).

/* house baratheon */

parent(ormund, steffon).
parent(rhaelle, steffon).

parent(steffon, robert).
parent(steffon, stannis).
parent(steffon, renly).

parent(stannis, shireen).

/* house targaryen */

parent(maekar, aemon).
parent(maekar, aegon).

parent(aegon, jaehaerys).
parent(aegon, rhaelle).

parent(jaehaerys, aerys).

parent(aerys, rhaegar).
parent(aerys, viserys).
parent(aerys, daenerys).

parent(rhaegar, snow).
parent(lyanna, snow).

/* house stark */

parent(rickard, brandon).
parent(rickard, eddard).
parent(rickard, benjen).
parent(rickard, lyanna).
```

```
57
58 parent(eddard, robb).
59 parent(eddard, sansa).
60 parent(eddard, arya).
61 parent(eddard, brandon).
62 parent(eddard, rickon).
63
64 /* house tully */
65
66 parent(robbyrt, hoster).
67 parent(robbyrt, brynden).
68
69 parent(hoster, lysa).
70 parent(hoster, edmure).
71 parent(hoster, catelyn).
72
73 parent(minisa, lysa).
74 parent(minisa, edmure).
75 parent(minisa, catelyn).
76
77 parent(catelyn, robb).
78 parent(catelyn, sansa).
79 parent(catelyn, arya).
80 parent(catelyn, brandon).
81 parent(catelyn, rickon).
82
83 parent(lysa, robin).
84
85 /* house tyrell */
86
87 parent(luthor, mace).
88
89 parent(olenna, mace).
90
91 parent(mace, margaery).
92 parent(mace, loras).
93
94 /* house martell */
95
96 parent(oberyn, elia).
97
98 parent(doran, elia).
99
100 /* house arryn */
101
102 parent(jon, robin).
103
104 /* male - female */
105
106 male(jon).
107 male(robin).
108 male(oberyn).
109 male(doran).
110 male(luthor).
111 male(mace).
112 male(loras).
113 male(robbyrt).
114 male(hoster).
```

```
115 male(bryden).
116 male(edmure).
117 male(rickard).
118 male(brandon).
119 male(eddard).
120 male(benjen).
121 male(robb).
122 male(brandon).
123 male(rickon).
124 male(ormund).
125 male(steffon).
126 male(robert).
127 male(stannis).
128 male(renly).
129 male(tywin).
130 male(jaime).
131 male(tyrion).
132 male(joffrey).
133 male(tommen).
134 male(maekar).
135 male(aemon).
136 male(aegon).
137 male(jaehaerys).
138 male(aerys).
139 male(rhaegar).
140 male(viserys).
141 male(snow).
142 female(rhaelle).
143 female(daenerys).
144 female(shireen).
145 female(myrcella).
146 female(cersei).
147 female(joanna).
148 female(sansa).
149 female(arya).
150 female(lyanna).
151 female(catelyn).
152 female(lysa).
153 female(minisa).
154 female(elia).
155 female(olenna).
156 female(margaery).
157
158 /* rule defines */
159
160 child(A, B) :-
161     parent(B, A).
162
163 sibling(A, B) :-
164     parent(C, A),
165     parent(C, B),
166     A \= B.
167
168 related(A, B) :-
169     ancestor(C, A),
170     ancestor(C, B).
171
172 descendant(A, B) :-
```

```
173     ancestor(B, A).
174
175 ancestor(A, B) :-
176     parent(A, B).
177
178 ancestor(A, B) :-
179     parent(C, B),
180     ancestor(A, C).
181
182 mother(A, B) :-
183     parent(A, B),
184     female(A).
185
186 father(A, B) :-
187     parent(A, B),
188     male(A).
189
190 son(A, B) :-
191     child(A, B),
192     male(A).
193
194 daughter(A, B) :-
195     child(A, B),
196     female(A).
197
198 sister(A, B) :-
199     sibling(A, B),
200     female(A),
201     A \= B.
202
203 brother(A, B) :-
204     sibling(A, B),
205     male(A),
206     A \= B.
207
208 grandparent(A, B) :-
209     parent(A, C),
210     parent(C, B).
211
212 grandfather(A, B) :-
213     grandparent(A, B),
214     male(A).
215
216 grandmother(A, B) :-
217     grandparent(A, B),
218     female(A).
219
220 grandchild(A, B) :-
221     grandparent(B, A).
222
223 greatgrandparent(A, B) :-
224     parent(P, B),
225     grandparent(A, P).
226
227 greatgrandchild(A, B) :-
228     greatgrandparent(B, A).
229
230 granddaughter(A, B) :-
```

```
231     grandchild(A, B),
232     female(A).
233
234 grandson(A, B) :-
235     grandchild(A, B),
236     male(A).
237
238 uncle(A, B) :-
239     brother(A, C),
240     child(B, C).
241
242 aunt(A, B) :-
243     sister(A, C),
244     child(B, C).
245
246 cousin(A, B) :-
247     grandparent(C, A),
248     grandparent(C, B),
249     \+sibling(A, B),
250     A \= B.
251
252 nephew(A, B) :-
253     aunt(B, A),
254     male(A);
255     uncle(B, A),
256     male(A).
257
258 niece(A, B) :-
259     aunt(B, A),
260     female(A);
261     uncle(B, A),
262     female(A).
263
264 secondcousin(A, B) :-
265     greatgrandparent(C, A),
266     greatgrandparent(C, B),
267     \+sibling(A, B),
268     \+cousin(A, B),
269     A \= B.
```