NANYANG TECHNOLOGICAL UNIVERSITY

CZ3005 Lab1 Logbook

Chen Zhiwei
U1620741J
TSP5

Log/Trace for Exercises in Section 1 to 5

---

3. Loading programs, editing programs (Exercise)

```
[trace]  ?- name('name',NameString).
NameString = [110, 97, 109, 101].


[trace]  ?- name(Name,"name").
Name = name.
```

---

5. Prolog as a Knowledge Base System (Exercise)

i)

```
?- p(X).
X = a ;
X = a ;
X = b ;
X = d.
```

ii)

```
[trace]  ?- p(X).
   Call: (8) p(_4104) ? creep
   Exit: (8) p(a) ? creep
X = a ;
   Redo: (8) p(_4104) ? creep
   Call: (9) q(_4104) ? creep
   Call: (10) s(_4104) ? creep
   Exit: (10) s(a) ? creep
   Exit: (9) q(a) ? creep
   Call: (9) r(a) ? creep
   Exit: (9) r(a) ? creep
   Exit: (8) p(a) ? creep
X = a ;
   Redo: (10) s(_4104) ? creep
   Exit: (10) s(b) ? creep
   Exit: (9) q(b) ? creep
   Call: (9) r(b) ? creep
   Exit: (9) r(b) ? creep
   Exit: (8) p(b) ? creep
X = b ;
   Redo: (10) s(_4104) ? creep
   Exit: (10) s(c) ? creep
   Exit: (9) q(c) ? creep
   Call: (9) r(c) ? creep
   Fail: (9) r(c) ? creep
   Redo: (8) p(_4104) ? creep
   Call: (9) u(_4104) ? creep
   Exit: (9) u(d) ? creep
   Exit: (8) p(d) ? creep
X = d.
```

Example 1 Family Tree

Facts and Definitions

Facts	Remarks
male(yeye).	My grandfather
male(xiaochun).	My father
male(zhiwei).	Myself
male(linfeng).	Cousin
mael(zhihong).	Cousin
male(william).	Husband of my aunt, hence, my uncle.
female(biqun).	Mother
female(qiuhua).	Aunt, sister of my father
female(qinhua).	Aunt, sister of my father
brother(xiaochun,qiuhua).	
brother(xiaochun,qinhua).	
sister(qiuhua,qinhua).	
sister(qiuhua,xiaochun).	
sister(qinhua,qiuhua).	
sister(qinhua,qiuhua).	
parent_of(yeye,xiaochun).	
parent_of(yeye,qiuhua).	
parent_of(yeye,qinhua).	
parent_of(xiaochun,zhiwei).	
parent_of(biqun,zhiwei).	
parent_of(qinhua,linfeng).	
parent_of(qiuhua,zhihong).	
parent_of(william,zhihong).	

Rules	Remarks
father(F,C) :- male(F), parent_of(F,C).	A father has to be a male. A father has to be parent of at least a child. If F is male and F is parent of C, then F is father of C.
mother(M,C) :- female(M), parent_of(M,C).	A mother has to be a female. A mother has to be parent of at least a child. If M is female and M is parent of C, then M is mother of C.
son(P,S) :- male(S), parent_of(P,S).	A son has to be a male. A son has to be the child of a parent. If S is male and P is parent of S, then S is son of P.
daughter(P,D) :- female(D), parent_of(P,D).	A daughter has to be a female. A daughter has to be the child of a parent. If D is female and P is parent of D, then D is daughter of P.
grandfather(Gf,Gc) :- father(Gf,P), parent_of(P,Gc).	A grandfather has to be father of a parent, who is the parent of the grandchild. * Note: Assuming Grandfather and Grandpa to be the same here. If Gf is father of P, and P is parent of Gc, then Gf is grandfather of Gc.
sibling(S1,S2) :- parent_of(P,S1), parent_of(P,S2), S1 \= S2.	Both siblings must share at least one parent. Both siblings cannot be the same person. If P is parent of both S1 and S2, and S1 is not S2, then S1 and S2 are siblings.
spouse(H,W) :- male(H), female(W), parent_of(H,C), parent_of(W,C).	A husband has to be a male. A wife has to be a female. The husband and wife has to be the parents of the same child(ren). * Note that in real-life, a pair of married couple need not raise a child. However, given the context of the question, the spouse relationship need to be derived from the stated facts. Thus, the rule of being the parents of the same child(ren) is imposed here. * Also, we are ignoring the fact that the couple may have divorced. If H is male and W is female, and H and W are both parent of C, then H and W are spouse of each other.

Rules	Remarks
aunt(A,C) :- (female(A), sibling(A,P), parent_of(P,C)); (female(A), spouse(B,A), sibling(B,P), parent_of(P,C)).	An aunt has to be a female. [An aunt has to be the sibling of the child's parent.] OR [An aunt has to be the spouse of the sibling of the child's parent.] * Note: assuming both aunt by blood and aunt by law to be legit aunt. If A is female and A is sibling of P, and P is parent of C, then A is aunt of C. OR If A is female and A is spouse of B, and B is sibling of P, and P is parent of C, then A is aunt of C.
uncle(U, C) :- (male(U), sibling(U,P), parent_of(P,C)); (male(U), spouse(U,S), sibling(S, P), parent_of(P,C)).	An uncle has to be a male. [An uncle has to be the sibling of the child's parent.] OR [An uncle has to be the spouse of the sibling of the child's parent.] * Note: assuming both uncle by blood and uncle by law to be legit uncle. If U is male and U is sibling of P, and P is parent of C, then U is uncle of C. OR If U is male and U is spouse of S, and S is sibling of P, and P is parent of C, then U is uncle of C.
cousin(C1,C2) :- sibling(P1,P2), parent_of(P1,C1), parent_of(P2,C2).	At least one parent of each cousin has to be siblings. If P1 and P2 are siblings, and P1 and P2 are parent of C1 and C2 respectively, then C1 and C2 are cousins.

---

1. spouse(X,Y) and uncle(X,myself)

1.1 Results for spouse(X,Y).

Prolog Rule:

spouse(Husband,Wife) :-
male(Husband), female(Wife), parent_of(Husband,Child), parent_of(Wife,Child).

Output of 'spouse(X,Y).':

X = xiaochun, Y = biqun ;
X = william, Y = qiuhua ;

Trace:

```
[trace]  ?- spouse(X,Y).
   Call: (8) spouse(_3974, _3976) ? creep
   Call: (9) male(_3974) ? creep
   Exit: (9) male(yeye) ? creep
   Call: (9) female(_3976) ? creep
   Exit: (9) female(biqun) ? creep
   Call: (9) parent_of(yeye, _4230) ? creep
   Exit: (9) parent_of(yeye, xiaochun) ? creep
   Call: (9) parent_of(biqun, xiaochun) ? creep
   Fail: (9) parent_of(biqun, xiaochun) ? creep
   Redo: (9) parent_of(yeye, _4230) ? creep
   Exit: (9) parent_of(yeye, qiuhua) ? creep
```

Call: (9) parent_of(biqun, qiuhua) ? creep
Fail: (9) parent_of(biqun, qiuhua) ? creep
Redo: (9) parent_of(yeye, _4230) ? creep
Exit: (9) parent_of(yeye, qinhua) ? creep
Call: (9) parent_of(biqun, qinhua) ? creep
Fail: (9) parent_of(biqun, qinhua) ? creep
Redo: (9) female(_3976) ? creep
Exit: (9) female(qiuhua) ? creep
Call: (9) parent_of(yeye, _4230) ? creep
Exit: (9) parent_of(yeye, xiaochun) ? creep
Call: (9) parent_of(qiuhua, xiaochun) ? creep
Fail: (9) parent_of(qiuhua, xiaochun) ? creep
Redo: (9) parent_of(yeye, _4230) ? creep
Exit: (9) parent_of(yeye, qiuhua) ? creep
Call: (9) parent_of(qiuhua, qiuhua) ? creep
Fail: (9) parent_of(qiuhua, qiuhua) ? creep
Redo: (9) parent_of(yeye, _4230) ? creep
Exit: (9) parent_of(yeye, qinhua) ? creep
Call: (9) parent_of(qiuhua, qinhua) ? creep
Fail: (9) parent_of(qiuhua, qinhua) ? creep
Redo: (9) female(_3976) ? creep
Exit: (9) female(qinhua) ? creep
Call: (9) parent_of(yeye, _4230) ? creep
Exit: (9) parent_of(yeye, xiaochun) ? creep
Call: (9) parent_of(qinhua, xiaochun) ? creep
Fail: (9) parent_of(qinhua, xiaochun) ? creep
Redo: (9) parent_of(yeye, _4230) ? creep
Exit: (9) parent_of(yeye, qiuhua) ? creep
Call: (9) parent_of(qinhua, qiuhua) ? creep
Fail: (9) parent_of(qinhua, qiuhua) ? creep
Redo: (9) parent_of(yeye, _4230) ? creep
Exit: (9) parent_of(yeye, qinhua) ? creep
Call: (9) parent_of(qinhua, qinhua) ? creep
Fail: (9) parent_of(qinhua, qinhua) ? creep
Redo: (9) female(_3976) ? creep
Exit: (9) female(kather) ? creep
Call: (9) parent_of(yeye, _4230) ? creep
Exit: (9) parent_of(yeye, xiaochun) ? creep
Call: (9) parent_of(kather, xiaochun) ? creep
Fail: (9) parent_of(kather, xiaochun) ? creep
Redo: (9) parent_of(yeye, _4230) ? creep
Exit: (9) parent_of(yeye, qiuhua) ? creep
Call: (9) parent_of(kather, qiuhua) ? creep
Fail: (9) parent_of(kather, qiuhua) ? creep
Redo: (9) parent_of(yeye, _4230) ? creep
Exit: (9) parent_of(yeye, qinhua) ? creep
Call: (9) parent_of(kather, qinhua) ? creep
Fail: (9) parent_of(kather, qinhua) ? creep
Redo: (9) female(_3976) ? creep
Exit: (9) female(maryalice) ? creep
Call: (9) parent_of(yeye, _4230) ? creep
Exit: (9) parent_of(yeye, xiaochun) ? creep
Call: (9) parent_of(maryalice, xiaochun) ? creep
Fail: (9) parent_of(maryalice, xiaochun) ? creep
Redo: (9) parent_of(yeye, _4230) ? creep
Exit: (9) parent_of(yeye, qiuhua) ? creep
Call: (9) parent_of(maryalice, qiuhua) ? creep
Fail: (9) parent_of(maryalice, qiuhua) ? creep
Redo: (9) parent_of(yeye, _4230) ? creep
Exit: (9) parent_of(yeye, qinhua) ? creep

Call: (9) parent_of(maryalice, qinhua) ? creep
Fail: (9) parent_of(maryalice, qinhua) ? creep
Redo: (9) female(_3976) ? creep
Exit: (9) female(ann) ? creep
Call: (9) parent_of(yeye, _4230) ? creep
Exit: (9) parent_of(yeye, xiaochun) ? creep
Call: (9) parent_of(ann, xiaochun) ? creep
Fail: (9) parent_of(ann, xiaochun) ? creep
Redo: (9) parent_of(yeye, _4230) ? creep
Exit: (9) parent_of(yeye, qiuhua) ? creep
Call: (9) parent_of(ann, qiuhua) ? creep
Fail: (9) parent_of(ann, qiuhua) ? creep
Redo: (9) parent_of(yeye, _4230) ? creep
Exit: (9) parent_of(yeye, qinhua) ? creep
Call: (9) parent_of(ann, qinhua) ? creep
Fail: (9) parent_of(ann, qinhua) ? creep
Redo: (9) male(_3974) ? creep
Exit: (9) male(xiaochun) ? creep
Call: (9) female(_3976) ? creep
Exit: (9) female(biqun) ? creep
Call: (9) parent_of(xiaochun, _4230) ? creep
Exit: (9) parent_of(xiaochun, zhiwei) ? creep
Call: (9) parent_of(biqun, zhiwei) ? creep
Exit: (9) parent_of(biqun, zhiwei) ? creep
Exit: (8) spouse(xiaochun, biqun) ? creep
X = xiaochun,
Y = biqun ;
Redo: (9) female(_3976) ? creep
Exit: (9) female(qiuhua) ? creep
Call: (9) parent_of(xiaochun, _4230) ? creep
Exit: (9) parent_of(xiaochun, zhiwei) ? creep
Call: (9) parent_of(qiuhua, zhiwei) ? creep
Fail: (9) parent_of(qiuhua, zhiwei) ? creep
Redo: (9) female(_3976) ? creep
Exit: (9) female(qinhua) ? creep
Call: (9) parent_of(xiaochun, _4230) ? creep
Exit: (9) parent_of(xiaochun, zhiwei) ? creep
Call: (9) parent_of(qinhua, zhiwei) ? creep
Fail: (9) parent_of(qinhua, zhiwei) ? creep
Redo: (9) female(_3976) ? creep
Exit: (9) female(kather) ? creep
Call: (9) parent_of(xiaochun, _4230) ? creep
Exit: (9) parent_of(xiaochun, zhiwei) ? creep
Call: (9) parent_of(kather, zhiwei) ? creep
Fail: (9) parent_of(kather, zhiwei) ? creep
Redo: (9) female(_3976) ? creep
Exit: (9) female(maryalice) ? creep
Call: (9) parent_of(xiaochun, _4230) ? creep
Exit: (9) parent_of(xiaochun, zhiwei) ? creep
Call: (9) parent_of(maryalice, zhiwei) ? creep
Fail: (9) parent_of(maryalice, zhiwei) ? creep
Redo: (9) female(_3976) ? creep
Exit: (9) female(ann) ? creep
Call: (9) parent_of(xiaochun, _4230) ? creep
Exit: (9) parent_of(xiaochun, zhiwei) ? creep
Call: (9) parent_of(ann, zhiwei) ? creep
Fail: (9) parent_of(ann, zhiwei) ? creep
Redo: (9) male(_3974) ? creep
Exit: (9) male(zhiwei) ? creep
Call: (9) female(_3976) ? creep

Exit: (9) female(biqun) ? creep
Call: (9) parent_of(zhiwei, _4230) ? creep
Fail: (9) parent_of(zhiwei, _4230) ? creep
Redo: (9) female(_3976) ? creep
Exit: (9) female(qiuhua) ? creep
Call: (9) parent_of(zhiwei, _4230) ? creep
Fail: (9) parent_of(zhiwei, _4230) ? creep
Redo: (9) female(_3976) ? creep
Exit: (9) female(qinhua) ? creep
Call: (9) parent_of(zhiwei, _4230) ? creep
Fail: (9) parent_of(zhiwei, _4230) ? creep
Redo: (9) female(_3976) ? creep
Exit: (9) female(kather) ? creep
Call: (9) parent_of(zhiwei, _4230) ? creep
Fail: (9) parent_of(zhiwei, _4230) ? creep
Redo: (9) female(_3976) ? creep
Exit: (9) female(maryalice) ? creep
Call: (9) parent_of(zhiwei, _4230) ? creep
Fail: (9) parent_of(zhiwei, _4230) ? creep
Redo: (9) female(_3976) ? creep
Exit: (9) female(ann) ? creep
Call: (9) parent_of(zhiwei, _4230) ? creep
Fail: (9) parent_of(zhiwei, _4230) ? creep
Redo: (9) male(_3974) ? creep
Exit: (9) male(linfeng) ? creep
Call: (9) female(_3976) ? creep
Exit: (9) female(biqun) ? creep
Call: (9) parent_of(linfeng, _4230) ? creep
Fail: (9) parent_of(linfeng, _4230) ? creep
Redo: (9) female(_3976) ? creep
Exit: (9) female(qiuhua) ? creep
Call: (9) parent_of(linfeng, _4230) ? creep
Fail: (9) parent_of(linfeng, _4230) ? creep
Redo: (9) female(_3976) ? creep
Exit: (9) female(qinhua) ? creep
Call: (9) parent_of(linfeng, _4230) ? creep
Fail: (9) parent_of(linfeng, _4230) ? creep
Redo: (9) female(_3976) ? creep
Exit: (9) female(kather) ? creep
Call: (9) parent_of(linfeng, _4230) ? creep
Fail: (9) parent_of(linfeng, _4230) ? creep
Redo: (9) female(_3976) ? creep
Exit: (9) female(maryalice) ? creep
Call: (9) parent_of(linfeng, _4230) ? creep
Fail: (9) parent_of(linfeng, _4230) ? creep
Redo: (9) female(_3976) ? creep
Exit: (9) female(ann) ? creep
Call: (9) parent_of(linfeng, _4230) ? creep
Fail: (9) parent_of(linfeng, _4230) ? creep
Redo: (9) male(_3974) ? creep
Exit: (9) male(william) ? creep
Call: (9) female(_3976) ? creep
Exit: (9) female(biqun) ? creep
Call: (9) parent_of(william, _4230) ? creep
Exit: (9) parent_of(william, zhihong) ? creep
Call: (9) parent_of(biqun, zhihong) ? creep
Fail: (9) parent_of(biqun, zhihong) ? creep
Redo: (9) female(_3976) ? creep
Exit: (9) female(qiuhua) ? creep
Call: (9) parent_of(william, _4230) ? creep

Exit: (9) parent_of(william, zhihong) ? creep
Call: (9) parent_of(qiuhua, zhihong) ? creep
Exit: (9) parent_of(qiuhua, zhihong) ? creep
Exit: (8) spouse(william, qiuhua) ? creep
X = william,
Y = qiuhua .

1.2 Results for uncle(X,myself).
Prolog Rule:

uncle(Uncle, Child) :-
(male(Uncle), sibling(Uncle,Parent), parent_of(Parent,Child));
(male(Uncle), spouse(Uncle,Sister), sibling(Sister, Parent), parent_of(Parent,Child)).

Output of 'uncle(X, zhiwei)':
X = william;

Trace:
[trace] ?- uncle(X,zhiwei).
Call: (8) uncle(_4002, zhiwei) ? creep
Call: (9) male(_4002) ? creep
Exit: (9) male(yeye) ? creep
Call: (9) sibling(yeye, _4224) ? creep
Call: (10) parent_of(_4222, yeye) ? creep
Fail: (10) parent_of(_4222, yeye) ? creep
Fail: (9) sibling(yeye, _4224) ? creep
Redo: (9) male(_4002) ? creep
Exit: (9) male(xiaochun) ? creep
Call: (9) sibling(xiaochun, _4224) ? creep
Call: (10) parent_of(_4222, xiaochun) ? creep
Exit: (10) parent_of(yeye, xiaochun) ? creep
Call: (10) parent_of(yeye, _4224) ? creep
Exit: (10) parent_of(yeye, xiaochun) ? creep
Call: (10) xiaochun\=xiaochun ? creep
Fail: (10) xiaochun\=xiaochun ? creep
Redo: (10) parent_of(yeye, _4224) ? creep
Exit: (10) parent_of(yeye, qiuhua) ? creep
Call: (10) xiaochun\=qiuhua ? creep
Exit: (10) xiaochun\=qiuhua ? creep
Exit: (9) sibling(xiaochun, qiuhua) ? creep
Call: (9) parent_of(qiuhua, zhiwei) ? creep
Fail: (9) parent_of(qiuhua, zhiwei) ? creep
Redo: (10) parent_of(yeye, _4224) ? creep
Exit: (10) parent_of(yeye, qinhua) ? creep
Call: (10) xiaochun\=qinhua ? creep
Exit: (10) xiaochun\=qinhua ? creep
Exit: (9) sibling(xiaochun, qinhua) ? creep
Call: (9) parent_of(qinhua, zhiwei) ? creep
Fail: (9) parent_of(qinhua, zhiwei) ? creep
Redo: (9) male(_4002) ? creep
Exit: (9) male(zhiwei) ? creep
Call: (9) sibling(zhiwei, _4224) ? creep
Call: (10) parent_of(_4222, zhiwei) ? creep
Exit: (10) parent_of(xiaochun, zhiwei) ? creep
Call: (10) parent_of(xiaochun, _4224) ? creep
Exit: (10) parent_of(xiaochun, zhiwei) ? creep
Call: (10) zhiwei\=zhiwei ? creep
Fail: (10) zhiwei\=zhiwei ? creep
Redo: (10) parent_of(_4222, zhiwei) ? creep
Exit: (10) parent_of(biqun, zhiwei) ? creep
Call: (10) parent_of(biqun, _4224) ? creep

Exit: (10) parent_of(biqun, zhiwei) ? creep
Call: (10) zhiwei\=zhiwei ? creep
Fail: (10) zhiwei\=zhiwei ? creep
Fail: (9) sibling(zhiwei, _4224) ? creep
Redo: (9) male(_4002) ? creep
Exit: (9) male(linfeng) ? creep
Call: (9) sibling(linfeng, _4224) ? creep
Call: (10) parent_of(_4222, linfeng) ? creep
Exit: (10) parent_of(qinhua, linfeng) ? creep
Call: (10) parent_of(qinhua, _4224) ? creep
Exit: (10) parent_of(qinhua, linfeng) ? creep
Call: (10) linfeng\=linfeng ? creep
Fail: (10) linfeng\=linfeng ? creep
Fail: (9) sibling(linfeng, _4224) ? creep
Redo: (9) male(_4002) ? creep
Exit: (9) male(william) ? creep
Call: (9) sibling(william, _4224) ? creep
Call: (10) parent_of(_4222, william) ? creep
Fail: (10) parent_of(_4222, william) ? creep
Fail: (9) sibling(william, _4224) ? creep
Redo: (8) uncle(_4002, zhiwei) ? creep
Call: (9) male(_4002) ? creep
Exit: (9) male(yeye) ? creep
Call: (9) spouse(yeye, _4224) ? creep
Call: (10) male(yeye) ? creep
Exit: (10) male(yeye) ? creep
Call: (10) female(_4222) ? creep
Exit: (10) female(biqun) ? creep
Call: (10) parent_of(yeye, _4224) ? creep
Exit: (10) parent_of(yeye, xiaochun) ? creep
Call: (10) parent_of(biqun, xiaochun) ? creep
Fail: (10) parent_of(biqun, xiaochun) ? creep
Redo: (10) parent_of(yeye, _4224) ? creep
Exit: (10) parent_of(yeye, qiuhua) ? creep
Call: (10) parent_of(biqun, qiuhua) ? creep
Fail: (10) parent_of(biqun, qiuhua) ? creep
Redo: (10) parent_of(yeye, _4224) ? creep
Exit: (10) parent_of(yeye, qinhua) ? creep
Call: (10) parent_of(biqun, qinhua) ? creep
Fail: (10) parent_of(biqun, qinhua) ? creep
Redo: (10) female(_4222) ? creep
Exit: (10) female(qiuhua) ? creep
Call: (10) parent_of(yeye, _4224) ? creep
Exit: (10) parent_of(yeye, xiaochun) ? creep
Call: (10) parent_of(qiuhua, xiaochun) ? creep
Fail: (10) parent_of(qiuhua, xiaochun) ? creep
Redo: (10) parent_of(yeye, _4224) ? creep
Exit: (10) parent_of(yeye, qiuhua) ? creep
Call: (10) parent_of(qiuhua, qiuhua) ? creep
Fail: (10) parent_of(qiuhua, qiuhua) ? creep
Redo: (10) parent_of(yeye, _4224) ? creep
Exit: (10) parent_of(yeye, qinhua) ? creep
Call: (10) parent_of(qiuhua, qinhua) ? creep
Fail: (10) parent_of(qiuhua, qinhua) ? creep
Redo: (10) female(_4222) ? creep
Exit: (10) female(qinhua) ? creep
Call: (10) parent_of(yeye, _4224) ? creep
Exit: (10) parent_of(yeye, xiaochun) ? creep
Call: (10) parent_of(qinhua, xiaochun) ? creep
Fail: (10) parent_of(qinhua, xiaochun) ? creep

Redo: (10) parent_of(yeye, _4224) ? creep
Exit: (10) parent_of(yeye, qiuhua) ? creep
Call: (10) parent_of(qinhua, qiuhua) ? creep
Fail: (10) parent_of(qinhua, qiuhua) ? creep
Redo: (10) parent_of(yeye, _4224) ? creep
Exit: (10) parent_of(yeye, qinhua) ? creep
Call: (10) parent_of(qinhua, qinhua) ? creep
Fail: (10) parent_of(qinhua, qinhua) ? creep
Fail: (9) spouse(yeye, _4224) ? creep
Redo: (9) male(_4002) ? creep
Exit: (9) male(xiaochun) ? creep
Call: (9) spouse(xiaochun, _4224) ? creep
Call: (10) male(xiaochun) ? creep
Exit: (10) male(xiaochun) ? creep
Call: (10) female(_4222) ? creep
Exit: (10) female(biqun) ? creep
Call: (10) parent_of(xiaochun, _4224) ? creep
Exit: (10) parent_of(xiaochun, zhiwei) ? creep
Call: (10) parent_of(biqun, zhiwei) ? creep
Exit: (10) parent_of(biqun, zhiwei) ? creep
Exit: (9) spouse(xiaochun, biqun) ? creep
Call: (9) sibling(biqun, _4224) ? creep
Call: (10) parent_of(_4222, biqun) ? creep
Fail: (10) parent_of(_4222, biqun) ? creep
Fail: (9) sibling(biqun, _4224) ? creep
Redo: (10) female(_4222) ? creep
Exit: (10) female(qiuhua) ? creep
Call: (10) parent_of(xiaochun, _4224) ? creep
Exit: (10) parent_of(xiaochun, zhiwei) ? creep
Call: (10) parent_of(qiuhua, zhiwei) ? creep
Fail: (10) parent_of(qiuhua, zhiwei) ? creep
Redo: (10) female(_4222) ? creep
Exit: (10) female(qinhua) ? creep
Call: (10) parent_of(xiaochun, _4224) ? creep
Exit: (10) parent_of(xiaochun, zhiwei) ? creep
Call: (10) parent_of(qinhua, zhiwei) ? creep
Fail: (10) parent_of(qinhua, zhiwei) ? creep
Fail: (9) spouse(xiaochun, _4224) ? creep
Redo: (9) male(_4002) ? creep
Exit: (9) male(zhiwei) ? creep
Call: (9) spouse(zhiwei, _4224) ? creep
Call: (10) male(zhiwei) ? creep
Exit: (10) male(zhiwei) ? creep
Call: (10) female(_4222) ? creep
Exit: (10) female(biqun) ? creep
Call: (10) parent_of(zhiwei, _4224) ? creep
Fail: (10) parent_of(zhiwei, _4224) ? creep
Redo: (10) female(_4222) ? creep
Exit: (10) female(qiuhua) ? creep
Call: (10) parent_of(zhiwei, _4224) ? creep
Fail: (10) parent_of(zhiwei, _4224) ? creep
Redo: (10) female(_4222) ? creep
Exit: (10) female(qinhua) ? creep
Call: (10) parent_of(zhiwei, _4224) ? creep
Fail: (10) parent_of(zhiwei, _4224) ? creep
Fail: (9) spouse(zhiwei, _4224) ? creep
Redo: (9) male(_4002) ? creep
Exit: (9) male(linfeng) ? creep
Call: (9) spouse(linfeng, _4224) ? creep
Call: (10) male(linfeng) ? creep

Exit: (10) male(linfeng) ? creep
Call: (10) female(_4222) ? creep
Exit: (10) female(biqun) ? creep
Call: (10) parent_of(linfeng, _4224) ? creep
Fail: (10) parent_of(linfeng, _4224) ? creep
Redo: (10) female(_4222) ? creep
Exit: (10) female(qiuhua) ? creep
Call: (10) parent_of(linfeng, _4224) ? creep
Fail: (10) parent_of(linfeng, _4224) ? creep
Redo: (10) female(_4222) ? creep
Exit: (10) female(qinhua) ? creep
Call: (10) parent_of(linfeng, _4224) ? creep
Fail: (10) parent_of(linfeng, _4224) ? creep
Fail: (9) spouse(linfeng, _4224) ? creep
Redo: (9) male(_4002) ? creep
Exit: (9) male(william) ? creep
Call: (9) spouse(william, _4224) ? creep
Call: (10) male(william) ? creep
Exit: (10) male(william) ? creep
Call: (10) female(_4222) ? creep
Exit: (10) female(biqun) ? creep
Call: (10) parent_of(william, _4224) ? creep
Exit: (10) parent_of(william, zhihong) ? creep
Call: (10) parent_of(biqun, zhihong) ? creep
Fail: (10) parent_of(biqun, zhihong) ? creep
Redo: (10) female(_4222) ? creep
Exit: (10) female(qiuhua) ? creep
Call: (10) parent_of(william, _4224) ? creep
Exit: (10) parent_of(william, zhihong) ? creep
Call: (10) parent_of(qiuhua, zhihong) ? creep
Exit: (10) parent_of(qiuhua, zhihong) ? creep
Exit: (9) spouse(william, qiuhua) ? creep
Call: (9) sibling(qiuhua, _4224) ? creep
Call: (10) parent_of(_4222, qiuhua) ? creep
Exit: (10) parent_of(yeye, qiuhua) ? creep
Call: (10) parent_of(yeye, _4224) ? creep
Exit: (10) parent_of(yeye, xiaochun) ? creep
Call: (10) qiuhua\=xiaochun ? creep
Exit: (10) qiuhua\=xiaochun ? creep
Exit: (9) sibling(qiuhua, xiaochun) ? creep
Call: (9) parent_of(xiaochun, zhiwei) ? creep
Exit: (9) parent_of(xiaochun, zhiwei) ? creep
Exit: (8) uncle(william, zhiwei) ? creep
X = william .

2.

2a) Output of 'parent_of(X,Y).' before reordering	**2b) Output of 'parent_of(X,Y).' after reordering**
X = yeye, Y = xiaochun ;	X = yeye, Y = xiaochun ;
X = yeye, Y = qiuhua ;	X = yeye, Y = qiuhua ;
X = yeye, Y = qinhua ;	X = yeye, Y = qinhua ;
X = xiaochun, Y = zhiwei ;	X = xiaochun, Y = zhiwei ;
X = biqun, Y = zhiwei ;	X = biqun, Y = zhiwei ;

2a) Output of 'parent_of(X,Y).' before reordering	2b) Output of 'parent_of(X,Y).' after reordering
X = qinhua, Y = linfeng ;	X = qinhua, Y = linfeng ;
X = qiuhua, Y = zhihong ;	X = qiuhua, Y = zhihong ;
X = william, Y = zhihong ;	X = william, Y = zhihong ;
X = warren, Y = jerry ;	**X = maryalice, Y = jerry ;**
X = maryalice, Y = jerry.	**X = warren, Y = jerry.**

The results for 2a) and 2b) are different after reordering the facts. The difference is bolded in the above table. The reason is because of the order in which the facts are defined will affect the order in which the AND-OR tree searches for answers. The facts that got defined first will be searched and returned first. In 2a), parent_of(warren,jerry) is defined before parent_of(maryalice,jerry), while in 2b) these 2 pieces of facts are defined in the reverse order. This thereby resulted in the difference in the output for 2a) and 2b)

2a) Trace for 'parent_of(X,Y).':

Trace:

```
[trace]  ?- parent_of(X,Y).
   Call: (8) parent_of(_4368, _4370) ? creep
   Exit: (8) parent_of(yeye, xiaochun) ? creep
X = yeye,
Y = xiaochun ;
   Redo: (8) parent_of(_4368, _4370) ? creep
   Exit: (8) parent_of(yeye, qiuhua) ? creep
X = yeye,
Y = qiuhua ;
   Redo: (8) parent_of(_4368, _4370) ? creep
   Exit: (8) parent_of(yeye, qinhua) ? creep
X = yeye,
Y = qinhua ;
   Redo: (8) parent_of(_4368, _4370) ? creep
   Exit: (8) parent_of(xiaochun, zhiwei) ? creep
X = xiaochun,
Y = zhiwei ;
   Redo: (8) parent_of(_4368, _4370) ? creep
   Exit: (8) parent_of(biqun, zhiwei) ? creep
X = biqun,
Y = zhiwei ;
   Redo: (8) parent_of(_4368, _4370) ? creep
   Exit: (8) parent_of(qinhua, linfeng) ? creep
X = qinhua,
Y = linfeng ;
   Redo: (8) parent_of(_4368, _4370) ? creep
   Exit: (8) parent_of(qiuhua, zhihong) ? creep
X = qiuhua,
Y = zhihong ;
   Redo: (8) parent_of(_4368, _4370) ? creep
   Exit: (8) parent_of(william, zhihong) ? creep
X = william,
Y = zhihong ;
   Redo: (8) parent_of(_4368, _4370) ? creep
   Exit: (8) parent_of(warren, jerry) ? creep
X = warren,
Y = jerry ;
   Redo: (8) parent_of(_4368, _4370) ? creep
```

```
   Exit: (8) parent_of(maryalice, jerry) ? creep
X = maryalice,
Y = jerry.
```

2b) Trace for parent_of(X,Y). after reordering.

Trace:

```
[trace]  ?- parent_of(X,Y).
   Call: (8) parent_of(_3804, _3806) ? creep
   Exit: (8) parent_of(yeye, xiaochun) ? creep
X = yeye,
Y = xiaochun ;
   Redo: (8) parent_of(_3804, _3806) ? creep
   Exit: (8) parent_of(yeye, qiuhua) ? creep
X = yeye,
Y = qiuhua ;
   Redo: (8) parent_of(_3804, _3806) ? creep
   Exit: (8) parent_of(yeye, qinhua) ? creep
X = yeye,
Y = qinhua ;
   Redo: (8) parent_of(_3804, _3806) ? creep
   Exit: (8) parent_of(xiaochun, zhiwei) ? creep
X = xiaochun,
Y = zhiwei ;
   Redo: (8) parent_of(_3804, _3806) ? creep
   Exit: (8) parent_of(biqun, zhiwei) ? creep
X = biqun,
Y = zhiwei ;
   Redo: (8) parent_of(_3804, _3806) ? creep
   Exit: (8) parent_of(qinhua, linfeng) ? creep
X = qinhua,
Y = linfeng ;
   Redo: (8) parent_of(_3804, _3806) ? creep
   Exit: (8) parent_of(qiuhua, zhihong) ? creep
X = qiuhua,
Y = zhihong ;
   Redo: (8) parent_of(_3804, _3806) ? creep
   Exit: (8) parent_of(william, zhihong) ? creep
X = william,
Y = zhihong ;
   Redo: (8) parent_of(_3804, _3806) ? creep
   Exit: (8) parent_of(maryalice, jerry) ? creep
X = maryalice,
Y = jerry ;
   Redo: (8) parent_of(_3804, _3806) ? creep
   Exit: (8) parent_of(warren, jerry) ? creep
X = warren,
Y = jerry.
```

Example 2 The Smart Phone Rivalry

1. Translate the natural language statements above describing the dealing within the Smart Phone industry in to First Order Logic, (FOL).

Natural Language	FOL	Remarks
SumSum, a competitor of Appy.	Competitor(SumSum, Appy)	
SumSum developed some nice smart phone technology called Galactica-S3.	Developed(SumSum, Galactica-S3)	
A smart phone technology called Galactica-S3.	SmartphoneTech(Galactica-S3)	
Smartphone technology is a business.	∀X SmartphoneTech(X) => Business(X)	For all X, if X is a smartphone technology, then X is a business.
Galactica-S3 was stolen by Stevey.	Steal(Stevey, Galactica-S3)	
Stevey is a boss.	Boss(Appy, Stevey)	It's assumed that Stevey is the boss of Appy.
A competitor of Appy is a rival.	∀X∀Y Competitor(X,Y) => Rival(X,Y)	For all X, if X is a competitor of Y, then X is a rival of Y. The natural language stating "a competitor of Appy is a rival" is a specific case of the FOL defined.
It is unethical for a Boss to steal business from from rival companies.	∀W∀X∀Y∀Z (Boss(X, W) ∧ Steal(W, Y) ∧ Rival(X, Z) ∧ Developed(Z, Y) ∧ Business(Y)) => Unethical(W)	For all W and for all X and for all Y and for all Z, - if W is the boss of X, and - W stole Y, and - X and Z are rival, and - Z developed Y, and - Y is a business, then W is unethical.

2. Write the FOLs as Prolog Clauses

competitor(sumsum,appy).

developed(sumsum,galacticas3).

smartphonetech(galacticas3).

steal(stevey,galacticas3).

boss(appy,stevey).

```
business(X) :- smartphonetech(X).
rival(X,Y) :- competitor(X,Y); competitor(Y,X).
unethical(W) :- boss(X,W), steal(W,Y), rival(X,Z), developed(Z,Y), business(Y).
```

---

3. Using the prolog search engine, prove that Stevey is unethical. Show a trace of your proof.

Output of 'unethical(X).':

```
X = stevey.
```

Trace:

```
[trace]  ?- unethical(X).
   Call: (8) unethical(_13078) ? creep
   Call: (9) boss(_13292, _13078) ? creep
   Exit: (9) boss(appy, stevey) ? creep
   Call: (9) steal(stevey, _13294) ? creep
   Exit: (9) steal(stevey, galacticas3) ? creep
   Call: (9) rival(appy, _13294) ? creep
   Call: (10) competitor(appy, _13294) ? creep
   Fail: (10) competitor(appy, _13294) ? creep
   Redo: (9) rival(appy, _13294) ? creep
   Call: (10) competitor(_13292, appy) ? creep
   Exit: (10) competitor(sumsum, appy) ? creep
   Exit: (9) rival(appy, sumsum) ? creep
   Call: (9) developed(sumsum, galacticas3) ? creep
   Exit: (9) developed(sumsum, galacticas3) ? creep
   Call: (9) business(galacticas3) ? creep
   Call: (10) smartphonetech(galacticas3) ? creep
   Exit: (10) smartphonetech(galacticas3) ? creep
   Exit: (9) business(galacticas3) ? creep
   Exit: (8) unethical(stevey) ? creep
X = stevey.
```

Example 3 Royal Family

---

1. Define their relations and rules in a prolog rule base. Hence, define the old Royal succession rule. Using this old succession rule determine the line of succession based on the information given. Do a trace to show your results.

Prolog Rule Base	**Remarks**
monarch(elizabeth).	Fact
prince(charles).	Fact
prince(andrew).	Fact
prince(edward).	Fact
princess(ann).	Fact
older(charles,ann).	Fact
older(ann,andrew).	Fact
older(andrew,edward).	Fact
elder(X,Y) :- older(X,Y); (older(X,_),older(_,Y)).	Rule. If X is older than Y; OR X is older than someone and that someone is older than Y, then X is elder than Y.
old_rank(X,Y) :- (prince(X),princess(Y)); (prince(X),prince(Y),elder(X,Y)); (princess(X),princess(Y),elder(X,Y)).	Rule. If X is a prince and Y is a princess; OR X is a prince and Y is a prince and X is elder than Y; OR X is a princess and Y is a princess and X is elder than Y, then X ranks higher in the succession line than Y.
%% quick sort pivoting(H,[],[],[]). pivoting(H,[X|T],[X|L],G) :- **\+old_rank(X,H)**, pivoting(H,T,L,G). pivoting(H,[X|T],L,[X|G]) :- **old_rank(X,H)**, pivoting(H,T,L,G). quick_sort(List,Sorted) :- q_sort(List,[],Sorted). q_sort([],Acc,Acc). q_sort([H|T],Acc,Sorted) :- pivoting(H,T,L1,L2), q_sort(L1,Acc,Sorted1),q_sort(L2,[H|Sorted1],Sorted).	This quick sort implementation in prolog will be used to sort the succession line.

Prolog Rule Base	Remarks
old_succession(Candidates,Sorted_Candidates) :- quick_sort(Candidates,Sorted_Candidates).	This line of prolog calls the quick sort prolog implementation, so as the produce the sorted succession line.

Output of 'old_succession([andrew,charles,edward,ann],Succession).':

Succession = [charles, andrew, edward, ann] .

Trace:

```
[trace]  ?- old_succession([andrew,charles,edward,ann],Succession).
   Call: (8) old_succession([andrew, charles, edward, ann], _3324) ? creep
   Call: (9) quick_sort([andrew, charles, edward, ann], _3324) ? creep
   Call: (10) q_sort([andrew, charles, edward, ann], [], _3324) ? creep
   Call: (11) pivoting(andrew, [charles, edward, ann], _3578, _3580) ? creep
   Call: (12) old_rank(charles, andrew) ? creep
   Call: (13) prince(charles) ? creep
   Exit: (13) prince(charles) ? creep
   Call: (13) princess(andrew) ? creep
   Fail: (13) princess(andrew) ? creep
   Redo: (12) old_rank(charles, andrew) ? creep
   Call: (13) prince(charles) ? creep
   Exit: (13) prince(charles) ? creep
   Call: (13) prince(andrew) ? creep
   Exit: (13) prince(andrew) ? creep
   Call: (13) elder(charles, andrew) ? creep
   Call: (14) older(charles, andrew) ? creep
   Fail: (14) older(charles, andrew) ? creep
   Redo: (13) elder(charles, andrew) ? creep
   Call: (14) older(charles, _3582) ? creep
   Exit: (14) older(charles, ann) ? creep
   Call: (14) older(_3580, andrew) ? creep
   Exit: (14) older(ann, andrew) ? creep
   Exit: (13) elder(charles, andrew) ? creep
   Exit: (12) old_rank(charles, andrew) ? creep
   Call: (12) old_rank(charles, andrew) ? creep
   Call: (13) prince(charles) ? creep
   Exit: (13) prince(charles) ? creep
   Call: (13) princess(andrew) ? creep
   Fail: (13) princess(andrew) ? creep
   Redo: (12) old_rank(charles, andrew) ? creep
   Call: (13) prince(charles) ? creep
   Exit: (13) prince(charles) ? creep
   Call: (13) prince(andrew) ? creep
   Exit: (13) prince(andrew) ? creep
   Call: (13) elder(charles, andrew) ? creep
   Call: (14) older(charles, andrew) ? creep
   Fail: (14) older(charles, andrew) ? creep
   Redo: (13) elder(charles, andrew) ? creep
   Call: (14) older(charles, _3582) ? creep
   Exit: (14) older(charles, ann) ? creep
   Call: (14) older(_3580, andrew) ? creep
   Exit: (14) older(ann, andrew) ? creep
   Exit: (13) elder(charles, andrew) ? creep
   Exit: (12) old_rank(charles, andrew) ? creep
   Call: (12) pivoting(andrew, [edward, ann], _3584, _3562) ? creep
   Call: (13) old_rank(edward, andrew) ? creep
```

Call: (14) prince(edward) ? creep
Exit: (14) prince(edward) ? creep
Call: (14) princess(andrew) ? creep
Fail: (14) princess(andrew) ? creep
Redo: (13) old_rank(edward, andrew) ? creep
Call: (14) prince(edward) ? creep
Exit: (14) prince(edward) ? creep
Call: (14) prince(andrew) ? creep
Exit: (14) prince(andrew) ? creep
Call: (14) elder(edward, andrew) ? creep
Call: (15) older(edward, andrew) ? creep
Fail: (15) older(edward, andrew) ? creep
Redo: (14) elder(edward, andrew) ? creep
Call: (15) older(edward, _3588) ? creep
Fail: (15) older(edward, _3588) ? creep
Fail: (14) elder(edward, andrew) ? creep
Redo: (13) old_rank(edward, andrew) ? creep
Call: (14) princess(edward) ? creep
Fail: (14) princess(edward) ? creep
Fail: (13) old_rank(edward, andrew) ? creep
Redo: (12) pivoting(andrew, [edward, ann], [edward|_3568], _3562) ? creep
Call: (13) pivoting(andrew, [ann], _3568, _3562) ? creep
Call: (14) old_rank(ann, andrew) ? creep
Call: (15) prince(ann) ? creep
Fail: (15) prince(ann) ? creep
Redo: (14) old_rank(ann, andrew) ? creep
Call: (15) prince(ann) ? creep
Fail: (15) prince(ann) ? creep
Redo: (14) old_rank(ann, andrew) ? creep
Call: (15) princess(ann) ? creep
Exit: (15) princess(ann) ? creep
Call: (15) princess(andrew) ? creep
Fail: (15) princess(andrew) ? creep
Fail: (14) old_rank(ann, andrew) ? creep
Redo: (13) pivoting(andrew, [ann], [ann|_3574], _3562) ? creep
Call: (14) pivoting(andrew, [], _3574, _3562) ? creep
Exit: (14) pivoting(andrew, [], [], []) ? creep
Exit: (13) pivoting(andrew, [ann], [ann], []) ? creep
Exit: (12) pivoting(andrew, [edward, ann], [edward, ann], []) ? creep
Exit: (11) pivoting(andrew, [charles, edward, ann], [edward, ann], [charles]) ? creep
Call: (11) q_sort([edward, ann], [], _3596) ? creep
Call: (12) pivoting(edward, [ann], _3596, _3598) ? creep
Call: (13) old_rank(ann, edward) ? creep
Call: (14) prince(ann) ? creep
Fail: (14) prince(ann) ? creep
Redo: (13) old_rank(ann, edward) ? creep
Call: (14) prince(ann) ? creep
Fail: (14) prince(ann) ? creep
Redo: (13) old_rank(ann, edward) ? creep
Call: (14) princess(ann) ? creep
Exit: (14) princess(ann) ? creep
Call: (14) princess(edward) ? creep
Fail: (14) princess(edward) ? creep
Fail: (13) old_rank(ann, edward) ? creep
Redo: (12) pivoting(edward, [ann], [ann|_3580], _3604) ? creep
Call: (13) pivoting(edward, [], _3580, _3604) ? creep
Exit: (13) pivoting(edward, [], [], []) ? creep
Exit: (12) pivoting(edward, [ann], [ann], []) ? creep
Call: (12) q_sort([ann], [], _3602) ? creep
Call: (13) pivoting(ann, [], _3602, _3604) ? creep

```
   Exit: (13) pivoting(ann, [], [], []) ? creep
   Call: (13) q_sort([], [], _3602) ? creep
   Exit: (13) q_sort([], [], []) ? creep
   Call: (13) q_sort([], [ann], _3608) ? creep
   Exit: (13) q_sort([], [ann], [ann]) ? creep
   Exit: (12) q_sort([ann], [], [ann]) ? creep
   Call: (12) q_sort([], [edward, ann], _3614) ? creep
   Exit: (12) q_sort([], [edward, ann], [edward, ann]) ? creep
   Exit: (11) q_sort([edward, ann], [], [edward, ann]) ? creep
   Call: (11) q_sort([charles], [andrew, edward, ann], _3324) ? creep
   Call: (12) pivoting(charles, [], _3620, _3622) ? creep
   Exit: (12) pivoting(charles, [], [], []) ? creep
   Call: (12) q_sort([], [andrew, edward, ann], _3620) ? creep
   Exit: (12) q_sort([], [andrew, edward, ann], [andrew, edward, ann]) ? creep
   Call: (12) q_sort([], [charles, andrew, edward, ann], _3324) ? creep
   Exit: (12) q_sort([], [charles, andrew, edward, ann], [charles, andrew, edward, ann]) ?
creep
   Exit: (11) q_sort([charles], [andrew, edward, ann], [charles, andrew, edward, ann]) ? creep
   Exit: (10) q_sort([andrew, charles, edward, ann], [], [charles, andrew, edward, ann]) ?
creep
   Exit: (9) quick_sort([andrew, charles, edward, ann], [charles, andrew, edward, ann]) ?
creep
   Exit: (8) old_succession([andrew, charles, edward, ann], [charles, andrew, edward, ann]) ?
creep
Succession = [charles, andrew, edward, ann] .
```

---

2. Recently, the Royal succession rule has been modified. The throne is now passed down according to the order of birth irrespective of gender. Modify your rules and prolog knowledge base to handle the new succession rule. Explain the necessary changes to the knowledge needed to represent the new information. Use this new succession rule to determine the new line of succession based on the same knowledge given. Show your results using a trace.

Prolog Rule Base	**Remarks**
new_pivoting(H,[],[],[]). new_pivoting(H,[X|T],[X|L],G) :- **\+elder(X,H)**, new_pivoting(H,T,L,G). new_pivoting(H,[X|T],L,[X|G]) :- **elder(X,H)**, new_pivoting(H,T,L,G). new_quick_sort(List,Sorted) :- new_q_sort(List,[],Sorted). new_q_sort([],Acc,Acc). new_q_sort([H|T],Acc,Sorted) :- new_pivoting(H,T,L1,L2), new_q_sort(L1,Acc,Sorted1),new_q_sort(L2,[H| Sorted1],Sorted).	Since now age is the only concern in throne succession, a new quick sort implementation is defined.
new_succession(Candidates,Sorted_Candidates) :- new_quick_sort(Candidates,Sorted_Candidates).	This modified line of prolog calls the new quick sort prolog implementation, so as the produce the new sorted succession line.

Output of 'new_succession([andrew,charles,edward,ann],Succession).':

Succession = [charles, ann, andrew, edward] .

Trace:

```
[trace]  ?- new_succession([andrew,charles,edward,ann],Succession).
   Call: (8) new_succession([andrew, charles, edward, ann], _3384) ? creep
   Call: (9) new_quick_sort([andrew, charles, edward, ann], _3384) ? creep
   Call: (10) new_q_sort([andrew, charles, edward, ann], [], _3384) ? creep
   Call: (11) new_pivoting(andrew, [charles, edward, ann], _3650, _3652) ? creep
   Call: (12) elder(charles, andrew) ? creep
   Call: (13) older(charles, andrew) ? creep
   Fail: (13) older(charles, andrew) ? creep
   Redo: (12) elder(charles, andrew) ? creep
   Call: (13) older(charles, _3654) ? creep
   Exit: (13) older(charles, ann) ? creep
   Call: (13) older(_3652, andrew) ? creep
   Exit: (13) older(ann, andrew) ? creep
   Exit: (12) elder(charles, andrew) ? creep
   Call: (12) elder(charles, andrew) ? creep
   Call: (13) older(charles, andrew) ? creep
   Fail: (13) older(charles, andrew) ? creep
   Redo: (12) elder(charles, andrew) ? creep
   Call: (13) older(charles, _3654) ? creep
   Exit: (13) older(charles, ann) ? creep
   Call: (13) older(_3652, andrew) ? creep
   Exit: (13) older(ann, andrew) ? creep
   Exit: (12) elder(charles, andrew) ? creep
   Call: (12) new_pivoting(andrew, [edward, ann], _3656, _3634) ? creep
   Call: (13) elder(edward, andrew) ? creep
   Call: (14) older(edward, andrew) ? creep
   Fail: (14) older(edward, andrew) ? creep
   Redo: (13) elder(edward, andrew) ? creep
   Call: (14) older(edward, _3660) ? creep
   Fail: (14) older(edward, _3660) ? creep
   Fail: (13) elder(edward, andrew) ? creep
   Redo: (12) new_pivoting(andrew, [edward, ann], [edward|_3640], _3634) ? creep
   Call: (13) new_pivoting(andrew, [ann], _3640, _3634) ? creep
   Call: (14) elder(ann, andrew) ? creep
   Call: (15) older(ann, andrew) ? creep
   Exit: (15) older(ann, andrew) ? creep
   Exit: (14) elder(ann, andrew) ? creep
   Call: (14) elder(ann, andrew) ? creep
   Call: (15) older(ann, andrew) ? creep
   Exit: (15) older(ann, andrew) ? creep
   Exit: (14) elder(ann, andrew) ? creep
   Call: (14) new_pivoting(andrew, [], _3640, _3646) ? creep
   Exit: (14) new_pivoting(andrew, [], [], []) ? creep
   Exit: (13) new_pivoting(andrew, [ann], [], [ann]) ? creep
   Exit: (12) new_pivoting(andrew, [edward, ann], [edward], [ann]) ? creep
   Exit: (11) new_pivoting(andrew, [charles, edward, ann], [edward], [charles, ann]) ? creep
   Call: (11) new_q_sort([edward], [], _3668) ? creep
   Call: (12) new_pivoting(edward, [], _3668, _3670) ? creep
   Exit: (12) new_pivoting(edward, [], [], []) ? creep
   Call: (12) new_q_sort([], [], _3668) ? creep
   Exit: (12) new_q_sort([], [], []) ? creep
   Call: (12) new_q_sort([], [edward], _3674) ? creep
   Exit: (12) new_q_sort([], [edward], [edward]) ? creep
   Exit: (11) new_q_sort([edward], [], [edward]) ? creep
   Call: (11) new_q_sort([charles, ann], [andrew, edward], _3384) ? creep
```

Call: (12) new_pivoting(charles, [ann], _3680, _3682) ? creep
Call: (13) elder(ann, charles) ? creep
Call: (14) older(ann, charles) ? creep
Fail: (14) older(ann, charles) ? creep
Redo: (13) elder(ann, charles) ? creep
Call: (14) older(ann, _3684) ? creep
Exit: (14) older(ann, andrew) ? creep
Call: (14) older(_3682, charles) ? creep
Fail: (14) older(_3682, charles) ? creep
Fail: (13) elder(ann, charles) ? creep
Redo: (12) new_pivoting(charles, [ann], [ann|_3664], _3688) ? creep
Call: (13) new_pivoting(charles, [], _3664, _3688) ? creep
Exit: (13) new_pivoting(charles, [], [], []) ? creep
Exit: (12) new_pivoting(charles, [ann], [ann], []) ? creep
Call: (12) new_q_sort([ann], [andrew, edward], _3686) ? creep
Call: (13) new_pivoting(ann, [], _3686, _3688) ? creep
Exit: (13) new_pivoting(ann, [], [], []) ? creep
Call: (13) new_q_sort([], [andrew, edward], _3686) ? creep
Exit: (13) new_q_sort([], [andrew, edward], [andrew, edward]) ? creep
Call: (13) new_q_sort([], [ann, andrew, edward], _3692) ? creep
Exit: (13) new_q_sort([], [ann, andrew, edward], [ann, andrew, edward]) ? creep
Exit: (12) new_q_sort([ann], [andrew, edward], [ann, andrew, edward]) ? creep
Call: (12) new_q_sort([], [charles, ann, andrew, edward], _3384) ? creep
Exit: (12) new_q_sort([], [charles, ann, andrew, edward], [charles, ann, andrew, edward]) ? creep
Exit: (11) new_q_sort([charles, ann], [andrew, edward], [charles, ann, andrew, edward]) ? creep
Exit: (10) new_q_sort([andrew, charles, edward, ann], [], [charles, ann, andrew, edward]) ? creep
Exit: (9) new_quick_sort([andrew, charles, edward, ann], [charles, ann, andrew, edward]) ? creep
Exit: (8) new_succession([andrew, charles, edward, ann], [charles, ann, andrew, edward]) ? creep