Nanyang Technological University

Lab 2 Report: Prolog

CZ3005 ARTIFICIAL INTELLIGENCE

Name: Eng Yong Peng
Lab Group: SSP1

Exercise 1: The Smart Phone Rivalry

1) First Order Logic (FOL)

Constant: SumSum, Appy, Galactica-S3, Stevey, Boss, Smartphone

Predicate	**Definition**
Competitor(X,Y)	X is a competitor of Y.
Developed(X,Y)	X developed Y, where X is a company and Y is a smartphone.
Steal(X,Y,Z)	X steals Y from Z, where Y is a smartphone developed by Z company.
Boss(X)	X is a boss.
Unethical(X)	X is unethical.
Rival(X)	X is a rival.
Company(X)	X is a company.
Smartphone(X)	X is a smartphone.
Business(X)	X is a business.

Translate natural language to FOL

S/N	**Natural Language**	**First Order Logic**	**Rule/Fact**
1	SumSum, a competitor of Appy	Competitor(SumSum, Appy) Company(SumSum) Company(Appy)	Fact
2	SumSum, a competitor of Appy, developed some nice smart phone technology called	∀x Smartphone(x) ∧ Developed(SumSum, x) → Steal(Stevey,x, SumSum)	Rule
3	Galactica-S3, all of which was stolen by Stevey, who is a Boss.	Smartphone(Galactica-S3) Developed(SumSum, Galactica-S3)	Fact
4	Stevey, who is a Boss.	Boss(Stevey)	Fact
5	It is unethical for a Boss to steal business from rival companies.	∀x,y,z Boss(x) ∧ Business(y) ∧ Rival(z) ∧ Company(z) ∧ Steal(x,y,z) → Unethical(x)	Rule
6	A competitor of Appy is a rival.	∀x Competitor(x, Apply) → Rival(x)	Rule
7	Smart phone technology is a business.	∀x Smartphone(x) → Business(x)	Rule

2) Prolog Clauses

S/N	FOL	Prolog Clauses	Rule/Fact
1	Competitor(SumSum, Appy) Company(SumSum) Company(Appy)	competitor(sumsum,appy). company(sumsum). company(appy).	Fact
2	∀x Smartphone(x) ∧ Developed(SumSum, x) → Steal(Stevey,x, SumSum)	steal(stevey,X,sumsum) :- smartphone(X),developed(sumsum,X).	Rule
3	Smartphone(Galactica-S3) Developed(SumSum, Galactica-S3)	smartphone(galactica-s3). developed(sumsum,galactica-s3).	Fact
4	Boss(Stevey)	boss(stevey).	Fact
5	∀x,y,z Boss(x) ∧ Business(y) ∧ Rival(z) ∧ Company(z) ∧ Steal(x,y,z) → Unethical(x)	unethical(X) :- boss(X),business(Y),rival(Z),company(Z),steal(X,Y,Z).	Rule
6	∀x Competitor(x, Apply) → Rival(x)	rival(X) :- competitor(X,appy).	Rule
7	∀x Smartphone(x) → Business(x)	business(X) :- smartphone(X).	Rule

Below is the Screenshot of "smartPhoneRivalry.pl"

```
competitor(sumsum,appy).
company(sumsum).
company(appy).
smartphone(galactica-s3).
developed(sumsum,galactica-s3).
boss(stevey).

steal(stevey,X,sumsum) :- smartphone(X),developed(sumsum,X).
unethical(X) :- boss(X),business(Y),rival(Z),company(Z),steal(X,Y,Z).
rival(X) :- competitor(X,appy).
business(X) :- smartphone(X).
```

3) Prove that Stevey is unethical

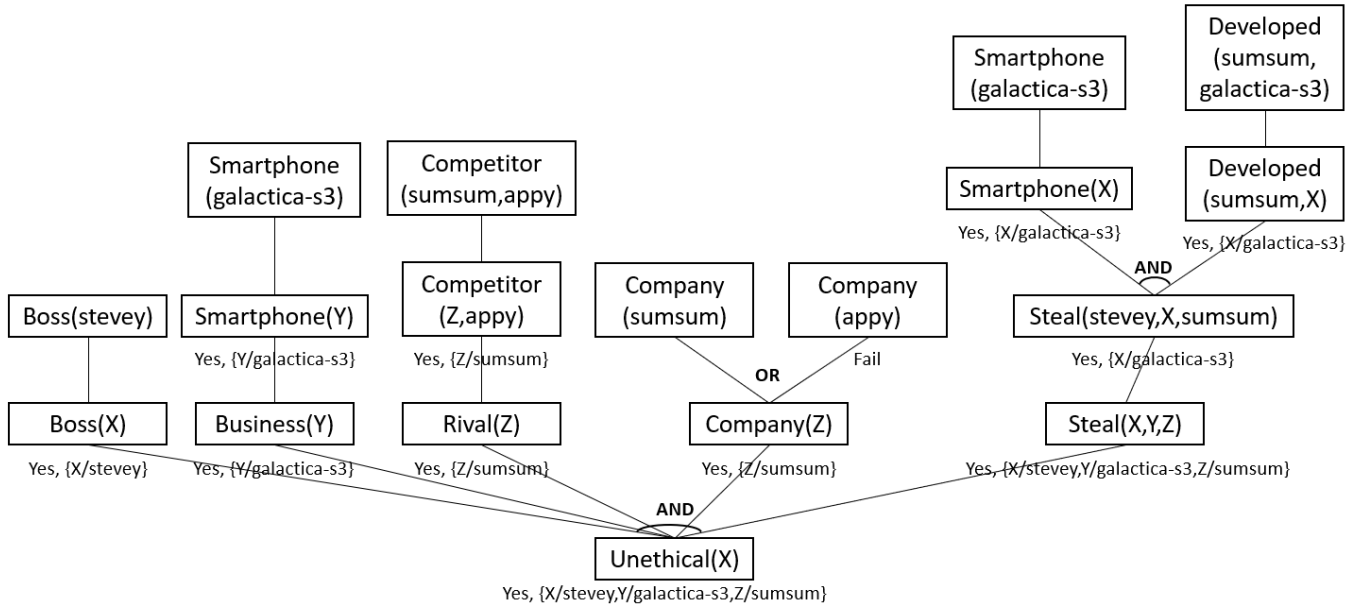
```
1 ?- trace.
true.

[trace] 1 ?- unethical(X).
   Call: (6) unethical(_G2443) ? creep
   Call: (7) boss(_G2443) ? creep
   Exit: (7) boss(stevey) ? creep
   Call: (7) business(_G2514) ? creep
   Call: (8) smartphone(_G2514) ? creep
   Exit: (8) smartphone(galactica-s3) ? creep
   Exit: (7) business(galactica-s3) ? creep
   Call: (7) rival(_G2517) ? creep
   Call: (8) competitor(_G2517, appy) ? creep
   Exit: (8) competitor(sumsum, appy) ? creep
   Exit: (7) rival(sumsum) ? creep
   Call: (7) company(sumsum) ? creep
   Exit: (7) company(sumsum) ? creep
   Call: (7) steal(stevey, galactica-s3, sumsum) ? creep
   Call: (8) smartphone(galactica-s3) ? creep
   Exit: (8) smartphone(galactica-s3) ? creep
   Call: (8) developed(sumsum, galactica-s3) ? creep
   Exit: (8) developed(sumsum, galactica-s3) ? creep
   Exit: (7) steal(stevey, galactica-s3, sumsum) ? creep
   Exit: (6) unethical(stevey) ? creep
X = stevey.

[trace] 2 ?-
```

The search process of the prolog is done by backward chaining. First the prolog will look at "unethical(X) :- boss(X),business(Y),rival(Z),company(Z),steal(X,Y,Z)." rule and traverse the all the predicates from left to right until all the predicate are satisfied. For each predicate, it will look from top to bottom of the knowledge base to search for fact that matches. Finally, when all the predicate is true, it will return the result of unethical(X) where X is stevey.

Below is the Screenshot of AND-OR Proof Tree for Unethical(X)



Exercise 2: The royal Family

1) Define old royal succession rule

Constant: Elizabeth, Charles, Ann, Andrew, Edward

Predicate	Definition
Monarch(X)	X is a competitor of the United Kingdom.
Male(X)	X is a male.
Female(X)	X is a female.
Mother(X,Y)	X is the mother of Y.
Brother(X,Y)	X is the brother of Y.
Sister(X,Y)	X is the sister of Y.
Oldest(X)	X is the oldest.
Younger(X,Y)	X is younger than Y.
Succeeds(X,Y)	X succeeds Y.

Below is the Screenshot of "royal part 1.pl"

```
male(charles).
male(andrew).
male(edward).
female(elizabeth).
female(ann).

monarch(elizabeth).
oldest(charles).

mother(elizabeth,charles).
mother(elizabeth,ann).
mother(elizabeth,andrew).
mother(elizabeth,edward).

brother(andrew,charles).
brother(edward,andrew).
sister(ann,edward).

younger(andrew,charles).
younger(edward,andrew).

succeeds(X,Y) :- male(X),oldest(X),monarch(Y),mother(Y,X).
succeeds(X,Y) :- brother(X,Y),younger(X,Y),monarch(Z),mother(Z,X),mother(Z,Y).
succeeds(X,Y) :- sister(X,Y),monarch(Z),mother(Z,X),mother(Z,Y).
```

As the old succession rule suggests, the next member who will inherit monarch is oldest male and if there is no more male, the next oldest female will succeed. There are 3 rules I created for this scenario. First, it will retrieve the oldest male. If the 1st rule does not satisfy, it will move to the 2nd rule, which is to look for younger child and must be a brother of the previous child (older) and both of the child must have their mother as monarch of the United Kingdom. After exhausted all the search for brothers, it will proceed to the 3rd rule, which is to find female successor with the same mother as monarch of the United Kingdom. The facts in the knowledge base are ordered in order of their birth. Mother(X,Y) is ordered

from oldest to youngest child and as well as younger(X,Y). Note that I did not add Ann to younger(X,Y) is because there is only female child which can be handled by 3rd rule.

Below screenshot shows the result of old succession rule

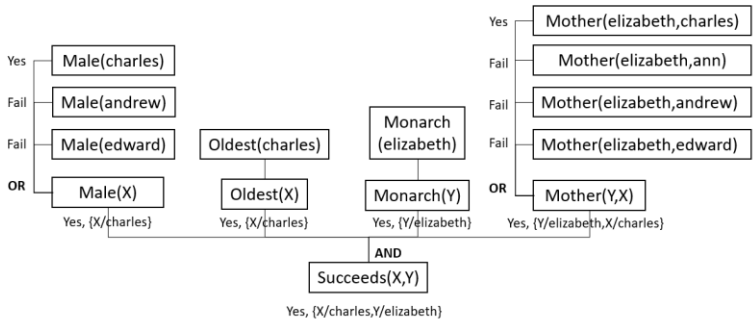
```
1 ?- succeeds(X,Y).
X = charles,
Y = elizabeth ;
X = andrew,
Y = charles ;
X = edward,
Y = andrew ;
X = ann,
Y = edward ;
false.

2 ?-
```

Below screenshot shows the trace of old succession rule

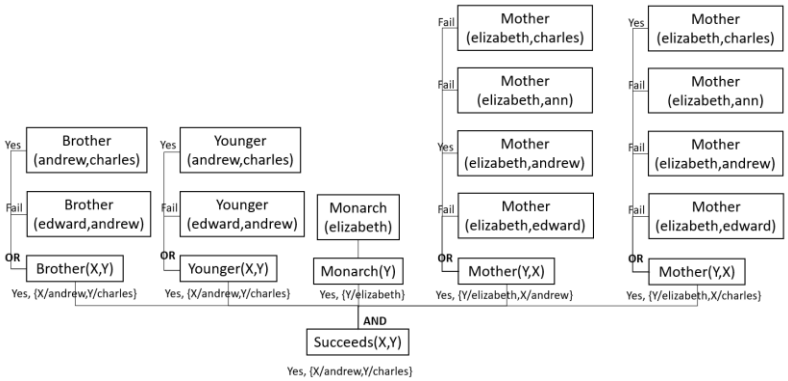
```
1 ?- trace.
true.

[trace] 1 ?- succeeds(X,Y).
   Call: (6) succeeds(_G2449, _G2450) ? creep
   Call: (7) male(_G2449) ? creep
   Exit: (7) male(charles) ? creep
   Call: (7) oldest(charles) ? creep
   Exit: (7) oldest(charles) ? creep
   Call: (7) monarch(_G2450) ? creep
   Exit: (7) monarch(elizabeth) ? creep
   Call: (7) mother(elizabeth, charles) ? creep
   Exit: (7) mother(elizabeth, charles) ? creep
   Exit: (6) succeeds(charles, elizabeth) ? creep
X = charles,
Y = elizabeth ;
```



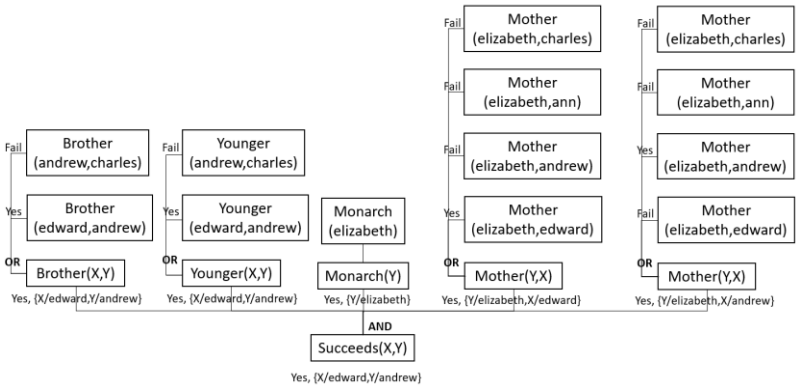
First it will look at the 1st succeeds rule "succeeds(X,Y) :- male(X),oldest(X),monarch(Y),mother(Y,X)." by traversing the predicate from left to right. Find the 1st male from the top of the knowledge base which is Charles and it satisfied and return value of Charles as X. By substituting X with Charles where oldest(X) and there IS a fact oldest(Charles). This will continue to check until all the predicate is true and return the result to succeeds(X,Y).

```
X = charles,
Y = elizabeth ;
   Redo: (7) mother(elizabeth, charles) ? creep
   Fail: (7) mother(elizabeth, charles) ? creep
   Redo: (7) male(_G2449) ? creep
   Exit: (7) male(andrew) ? creep
   Call: (7) oldest(andrew) ? creep
   Fail: (7) oldest(andrew) ? creep
   Redo: (7) male(_G2449) ? creep
   Exit: (7) male(edward) ? creep
   Call: (7) oldest(edward) ? creep
   Fail: (7) oldest(edward) ? creep
   Redo: (6) succeeds(_G2449, _G2450) ? creep
   Call: (7) brother(_G2449, _G2450) ? creep
   Exit: (7) brother(andrew, charles) ? creep
   Call: (7) younger(andrew, charles) ? creep
   Exit: (7) younger(andrew, charles) ? creep
   Call: (7) monarch(_G2534) ? creep
   Exit: (7) monarch(elizabeth) ? creep
   Call: (7) mother(elizabeth, andrew) ? creep
   Exit: (7) mother(elizabeth, andrew) ? creep
   Call: (7) mother(elizabeth, charles) ? creep
   Exit: (7) mother(elizabeth, charles) ? creep
   Exit: (6) succeeds(andrew, charles) ? creep
X = andrew,
Y = charles ;
```



Now we do not want Charles as the successor, we can continue by pressing ";" to continue searching for next successor. The knowledge base will continue to check if any of the facts satisfy the 1st rule. Since there is only 1 fact of oldest(x), the 1st rule will be invalid and proceed to 2nd rule. The 2nd rule find the younger brother of the oldest male and return the result as Andrew.

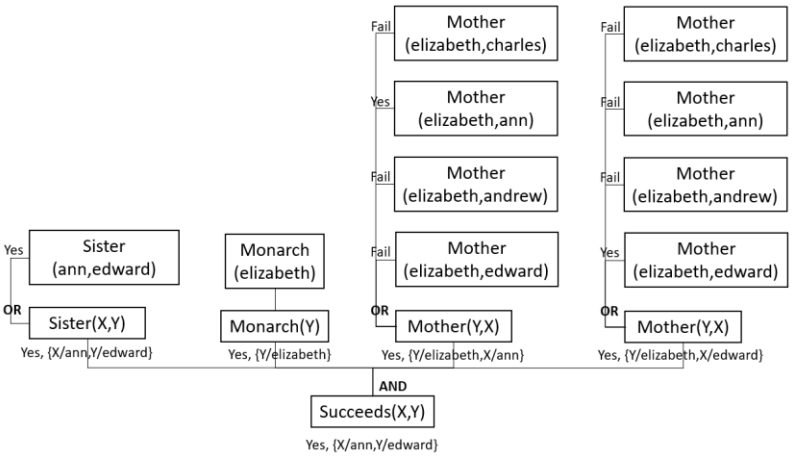
```
X = andrew,
Y = charles ;
   Redo: (7) mother(elizabeth, charles) ? creep
   Fail: (7) mother(elizabeth, charles) ? creep
   Redo: (7) mother(elizabeth, andrew) ? creep
   Fail: (7) mother(elizabeth, andrew) ? creep
   Redo: (7) brother(_G2449, _G2450) ? creep
   Exit: (7) brother(edward, andrew) ? creep
   Call: (7) younger(edward, andrew) ? creep
   Exit: (7) younger(edward, andrew) ? creep
   Call: (7) monarch(_G2534) ? creep
   Exit: (7) monarch(elizabeth) ? creep
   Call: (7) mother(elizabeth, edward) ? creep
   Exit: (7) mother(elizabeth, edward) ? creep
   Call: (7) mother(elizabeth, andrew) ? creep
   Exit: (7) mother(elizabeth, andrew) ? creep
   Exit: (6) succeeds(edward, andrew) ? creep
X = edward,
Y = andrew
```



Now we do not want Andrew as the successor. As there is 1 more brother fact in the knowledge base, it will continue using the 2nd rule and returns Edward.

```
X = edward,
Y = andrew ;
   Redo: (7) mother(elizabeth, andrew) ? creep
   Fail: (7) mother(elizabeth, andrew) ? creep
   Redo: (6) succeeds(_G2449, _G2450) ? creep
   Call: (7) sister(_G2449, _G2450) ? creep
   Exit: (7) sister(ann, edward) ? creep
   Call: (7) monarch(_G2534) ? creep
   Exit: (7) monarch(elizabeth) ? creep
   Call: (7) mother(elizabeth, ann) ? creep
   Exit: (7) mother(elizabeth, ann) ? creep
   Call: (7) mother(elizabeth, edward) ? creep
   Exit: (7) mother(elizabeth, edward) ? creep
   Exit: (6) succeeds(ann, edward) ? creep
X = ann,
Y = edward ;
   Redo: (7) mother(elizabeth, ann) ? creep
   Fail: (7) mother(elizabeth, ann) ? creep
   Fail: (6) succeeds(_G2449, _G2450) ? creep
false.

[trace] 2 ?-
|
```



Now we do not want Edward as the successor. This time round, we had exhausted all the brother and younger facts. It will proceed with the last rule to find a sister to become the next successor and returns Ann.

2) Define new royal succession rule

Below is the Screenshot of "royal part 2.pl"

```
male(charles).
male(andrew).
male(edward).
female(elizabeth).
female(ann).

monarch(elizabeth).
oldest(charles).

mother(elizabeth,charles).
mother(elizabeth,ann).
mother(elizabeth,andrew).
mother(elizabeth,edward).

younger(ann,charles).
younger(andrew,ann).
younger(edward,andrew).

succeeds(X,Y) :- oldest(X),monarch(Y),mother(Y,X).
succeeds(X,Y) :- younger(X,Y),monarch(Z),mother(Z,X),mother(Z,Y).
```

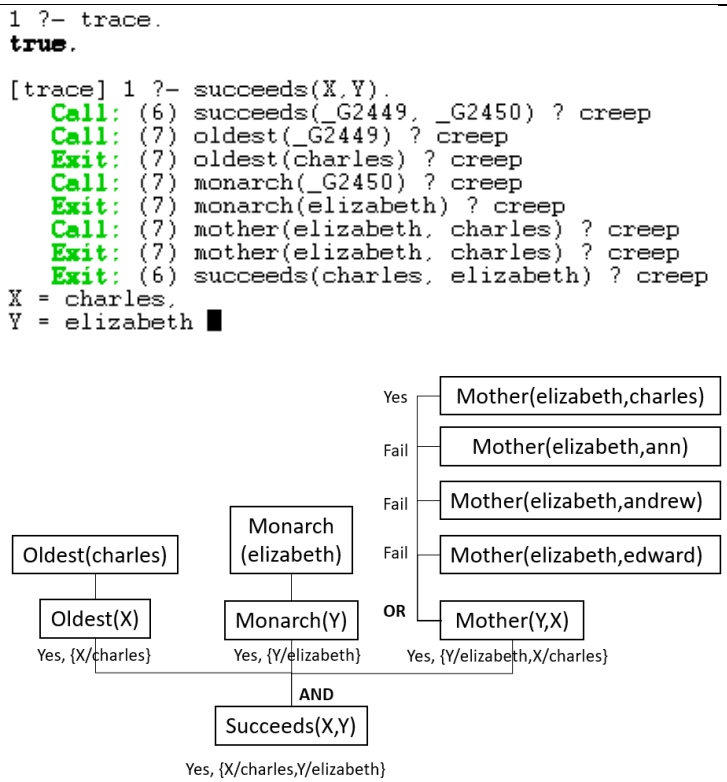
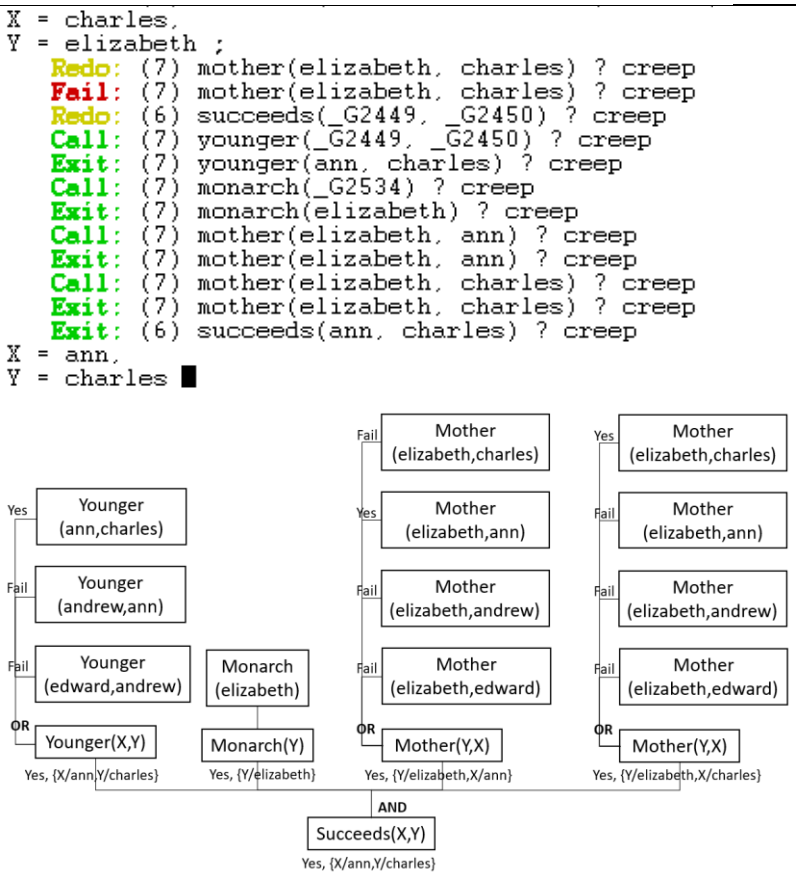
As the new succession rule suggests, the next member who will inherit monarch is oldest child and followed by younger child. Since male and female successor does not apply anymore, the 3rd rule, brother and sister facts can be removed. Male predicate of 1st rule will also be removed. With addition to younger facts to include Ann in the hierarchy. Younger(X,Y) is now determines the order of birth.

Below screenshot shows the result of new succession rule

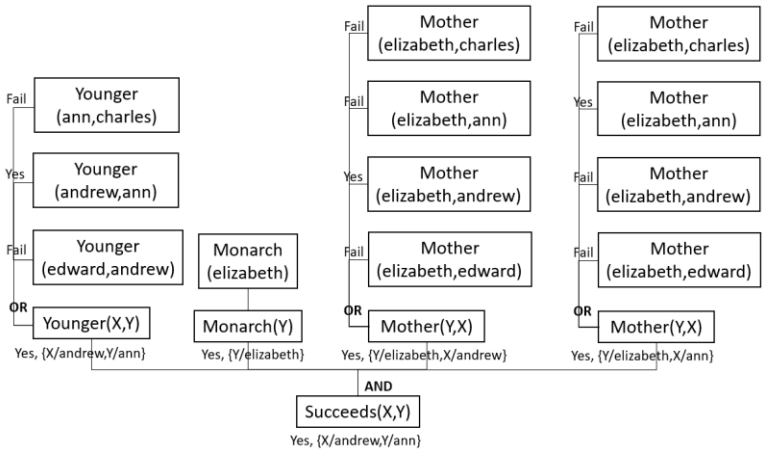
```
1 ?- succeeds(X,Y).
X = charles,
Y = elizabeth ;
X = ann,
Y = charles ;
X = andrew,
Y = ann ;
X = edward,
Y = andrew ;
false.

2 ?-
```

Below screenshot shows the trace of old succession rule

	1st rule will be used to find the oldest child as successor with mother as the monarch of United Kingdom. The predicate in the rule will be checked from left to right. Returns Charles as successor.
	Since there is only 1 oldest(X) predicate, it will not use the 1st rule again. 2nd rule will be used to determine the next younger child after the oldest child to be the next successor. Returns Ann as successor.

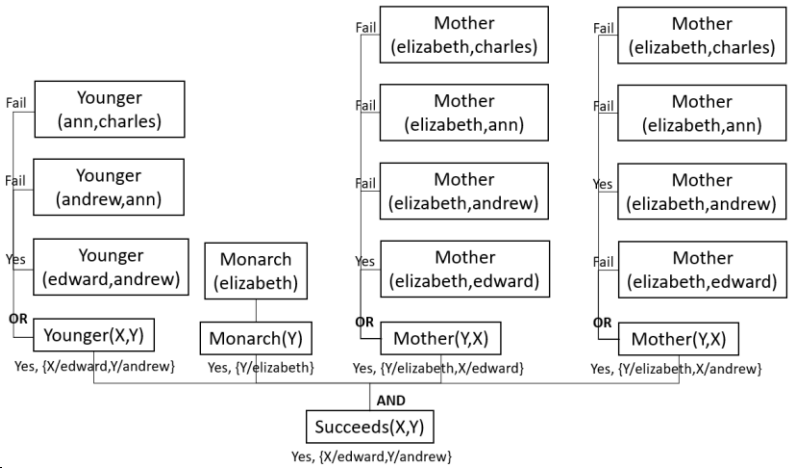
```
X = ann,
Y = charles ;
   Redo: (7) mother(elizabeth, charles) ? creep
   Fail: (7) mother(elizabeth, charles) ? creep
   Redo: (7) mother(elizabeth, ann) ? creep
   Fail: (7) mother(elizabeth, ann) ? creep
   Redo: (7) younger(_G2449, _G2450) ? creep
   Exit: (7) younger(andrew, ann) ? creep
   Call: (7) monarch(_G2534) ? creep
   Exit: (7) monarch(elizabeth) ? creep
   Call: (7) mother(elizabeth, andrew) ? creep
   Exit: (7) mother(elizabeth, andrew) ? creep
   Call: (7) mother(elizabeth, ann) ? creep
   Exit: (7) mother(elizabeth, ann) ? creep
   Exit: (6) succeeds(andrew, ann) ? creep
X = andrew,
Y = ann
```



2nd rule will be continuing to apply.
Returns Andrew as successor.

```
X = andrew,
Y = ann ;
   Redo: (7) mother(elizabeth, ann) ? creep
   Fail: (7) mother(elizabeth, ann) ? creep
   Redo: (7) mother(elizabeth, andrew) ? creep
   Fail: (7) mother(elizabeth, andrew) ? creep
   Redo: (7) younger(_G2449, _G2450) ? creep
   Exit: (7) younger(edward, andrew) ? creep
   Call: (7) monarch(_G2534) ? creep
   Exit: (7) monarch(elizabeth) ? creep
   Call: (7) mother(elizabeth, edward) ? creep
   Exit: (7) mother(elizabeth, edward) ? creep
   Call: (7) mother(elizabeth, andrew) ? creep
   Exit: (7) mother(elizabeth, andrew) ? creep
   Exit: (6) succeeds(edward, andrew) ? creep
X = edward,
Y = andrew ;
   Redo: (7) mother(elizabeth, andrew) ? creep
   Fail: (7) mother(elizabeth, andrew) ? creep
   Fail: (6) succeeds(_G2449, _G2450) ? creep
false.

[trace] 2 ?-
|
```



2nd rule will be continuing to apply until it has exhausted all the younger(X,Y) predicate.
Returns Edward as successor.