# Lab 7 First-Order Logic

SUSTC

# Coding Example – PL Forward Chaining Revisit

- Example of PL-FC inference:

```
horn_clauses_KB = PropDefiniteKB()
for s in "P==>Q; (L&M)==>P; (B&L)==>M; (A&P)==>L; (A&B)==>L; A;B".split(';'):
    horn_clauses_KB.tell(expr(s))
print('Can we conclude Q?')
print(pl_fc_entails(horn_clauses_KB, expr('Q')))
```

- Lab 7-1, we have the following chemical reaction:
  - $Fe_2O_3+H_2SO_4=Fe2(SO_4)_3+H_2O$
  - $SO_2+O_2=SO_3$
  - $FeS_2+O_2=Fe_2O_3+SO_2$
  - $SO_3+H_2O=H_2SO_4$
- Can we generate $Fe_2(SO_4)_3$ given $FeS_2$, $O_2$, $H_2O$?

# Coding Example – PL Forward Chaining

- Use PL-FC inference

```
from logic import *
hkb = PropDefiniteKB()

mkb="
      Fill in knowledge
      ".split(';')
for s in mkb:
        hkb.tell(expr(s))

for s in "FS;O;H2O".split(';'):
    hkb.tell(expr(s))

print(hkb.clauses)

s=expr('FSO4')
myans=pl_fc_entails(hkb, s)
print(myans)
```

Can we generate $Fe_2(SO_4)_3$ given $FeS_2$, $O_2$, $H_2O$?

# First-Order Logic KB

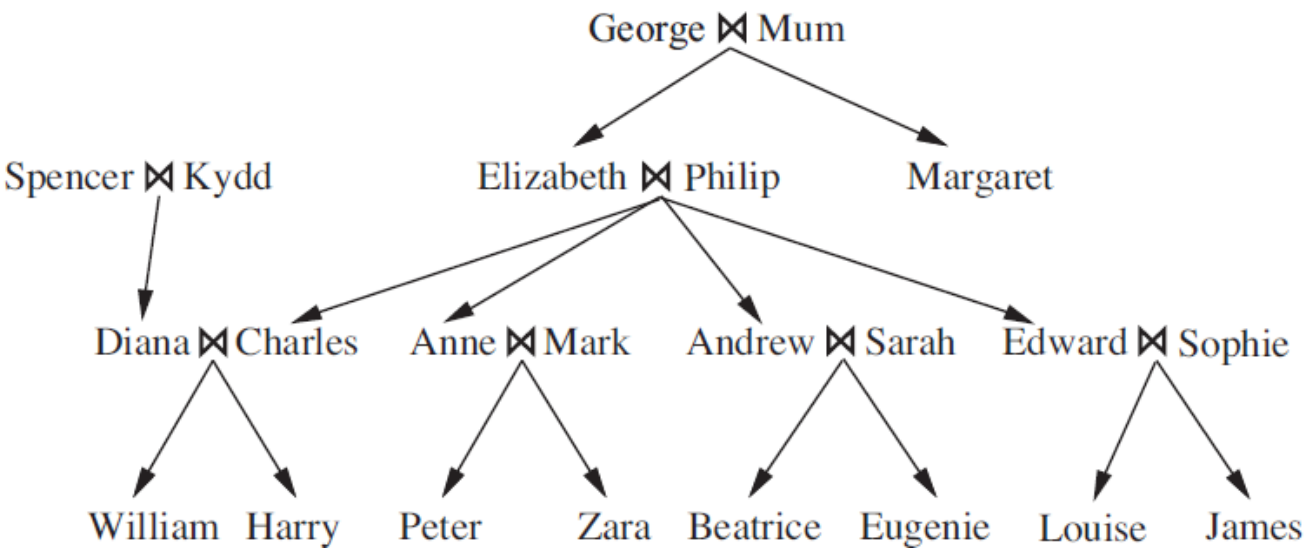
```
class FolKB(KB):

    def __init__(self, initial_clauses=[]):
        self.clauses = []  # inefficient: no indexing
        for clause in initial_clauses:
            self.tell(clause)

    def tell(self, sentence):
        if is_definite_clause(sentence):
            self.clauses.append(sentence)
        else:
            raise Exception("Not a definite clause: {}".format(sentence))

    def ask_generator(self, query):
        return fol_bc_ask(self, query)

    def retract(self, sentence):
        self.clauses.remove(sentence)

    def fetch_rules_for_goal(self, goal):
        return self.clauses
```

Backward Chaining

# Coding Example – FOL KB Backward Chaining

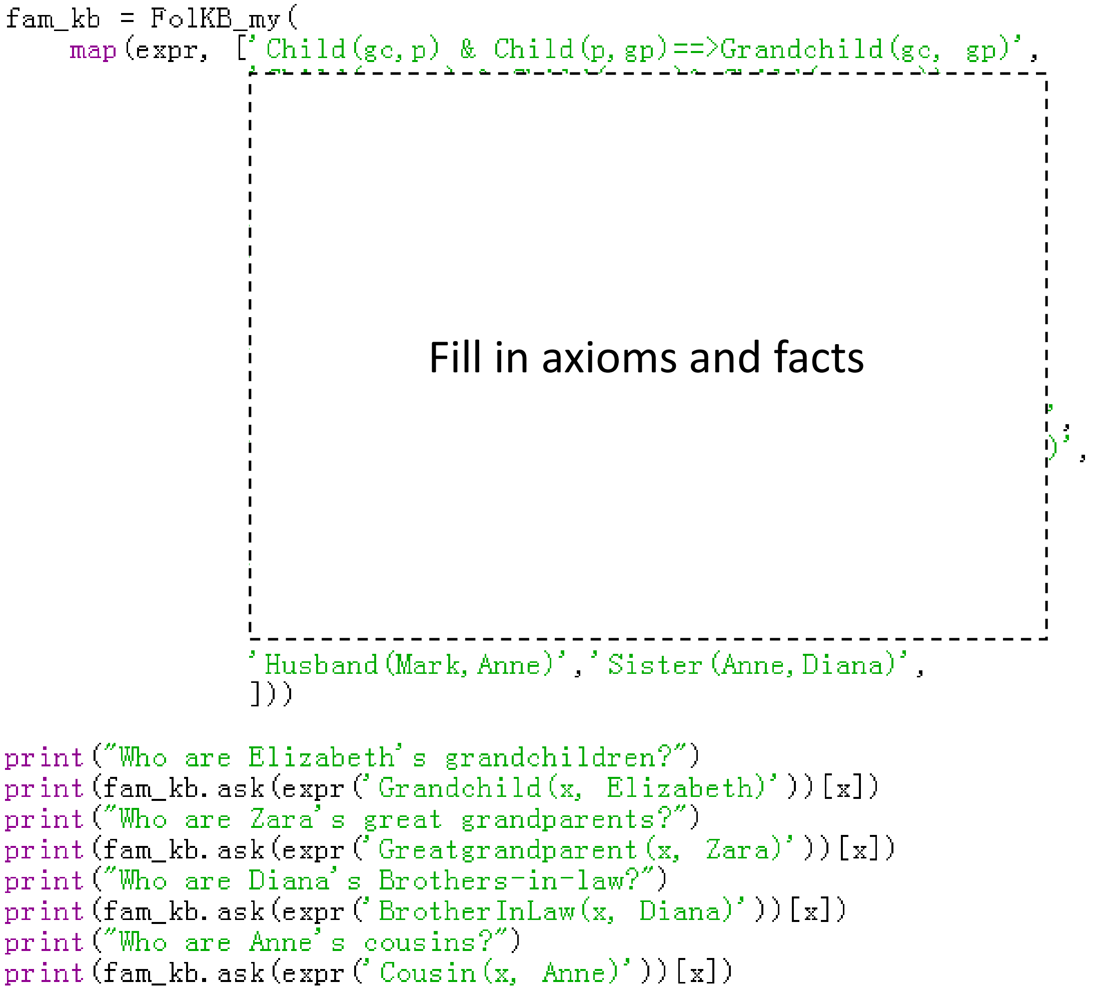
```
from logic import *

kb0 = FolKB([expr('Farmer(Mac)'), expr('Rabbit(Pete)'),\
             expr('(Rabbit(r) & Farmer(f)) ==> Hates(f, r)')])
test_kb = FolKB(
    map(expr, ['Farmer(Mac)',
               'Rabbit(Pete)',
               'Mother(MrsMac, Mac)',
               'Mother(MrsRabbit, Pete)',
               '(Rabbit(r) & Farmer(f)) ==> Hates(f, r)',
               '(Mother(m, c)) ==> Loves(m, c)',
               '(Mother(m, r) & Rabbit(r)) ==> Rabbit(m)',
               '(Farmer(f)) ==> Human(f)',
               # Note that this order of conjuncts
               # would result in infinite recursion:
               # '(Human(h) & Mother(m, h)) ==> Human(m)'
               '(Mother(m, h) & Human(h)) ==> Human(m)'
               ]))
test_kb.tell(expr('Rabbit(Flopsie)'))
test_kb.retract(expr('Rabbit(Pete)'))
print('Who does Mac hate?')
print(test_kb.ask(expr('Hates(Mac, x)'))[x])
```

# Lab 7-2: Homework 8.14




- Write axioms describing the predicates “Grandchild , Greatgrandparent, Ancestor, Brother, Sister, Daughter, Son, FirstCousin, BrotherInLaw, SisterInLaw, Aunt, and Uncle”.
- Who are (1) Elizabeth’s grandchildren, (2) Zara’s great-grandparents, (3) Diana’s brothers-in-law, and (4) Peter's cousins?

# Coding Example – FOL KB



```
fam_kb = FolKB_my(
    map(expr, ['Child(gc,p) & Child(p,gp)==>Grandchild(gc, gp)',
```

Fill in axioms and facts

```
               'Husband(Mark,Anne)','Sister(Anne,Diana)',
               ]))

print("Who are Elizabeth's grandchildren?")
print(fam_kb.ask(expr('Grandchild(x, Elizabeth)'))[x])
print("Who are Zara's great grandparents?")
print(fam_kb.ask(expr('Greatgrandparent(x, Zara)'))[x])
print("Who are Diana's Brothers-in-law?")
print(fam_kb.ask(expr('BrotherInLaw(x, Diana)'))[x])
print("Who are Anne's cousins?")
print(fam_kb.ask(expr('Cousin(x, Anne)'))[x])
```