# CSCE 625 Homework #3 (2024 Spring)

# First Order Logic (Part II)

# Total: 100 points (written + coding).

See Canvas for due date.

See the Canvas assignment description for submission files and requirements.

## 1 First-Order Logic

**Important:** In this section, assume that $w, x, y, z$ are variables; $A, B, C, D$ are constants; $f(\cdot), g(\cdot), h(\cdot)$ are functions; and $P(\cdot), Q(\cdot), R(\cdot)$ are predicates.

**Problem 1 (30 pts):** Proof by resolution.

(1) Translate the following into first-order logic, (2) convert the resulting formulas into a canonical form, and (3) prove the theorem using resolution.

Given:

1. All Aggies are college students
2. All Aggies are diligent
3. For all college students there is some Starbucks that they have tea at.
4. For all Starbucks where there is someone who is diligent and has tea there, that Starbucks is in a library.
5. Everyone who has tea in a library, they are brilliant.

Show that the following is a logical consequence of the above:

6. (conclusion) All Aggies are brilliant.

Note: The predicates you need to use are as follows:

- $Aggie(x)$: $x$ is an Aggie.

- $College(x)$: $x$ is a college student.
- $Tea(x, y)$: $x$ has tea in $y$.
- $Diligent(x)$: $x$ is diligent.
- $Starbucks(x)$: $x$ is a Starbucks.
- $Library(x)$: $x$ is inside a library.
- $Brilliant(x)$: $x$ is brilliant.

You may use $A(\cdot)$, $C(\cdot)$, $T(\cdot)$, $D(\cdot)$, $S(\cdot)$, $L(\cdot)$, $B(\cdot)$ as a shorthand, respectively.

The problem above is pretty straight forward, so you may have no problem translating to FOL. If you are curious how it is done in general, see `https://www.cs.utexas.edu/users/novak/reso.html` for hints on translating a story into FOL and proving it using resolution.

## 2 Programming : Propositional Logic Resolution Theorem Prover

**Problem 2 (40 pts):** Resolution theorem prover in Python.

For this programming component, you will be implementing a simple resolution-based theorem prover in Python. This will be for propositional logic.

- Download the `resolution.ipynb` file from Canvas→Files→Assignment→hw3.
- Then upload to `https://research.colab.google.com` (use your TAMU NetID to login), and run it.
- Look at the test code to get yourself familiarized with the provided facilities, such as `resolve()`.
- You can create new code cells to experiment.

The only part you need to implement is the `prover()` function. You will be implementing the *two-pointer method* (same as level saturation), using the *set-of-support strategy*.

Here's a description of the algorithm[1]:

1. Initialize the two pointers (simply, the clause index) so that you will be using the set-of-support strategy. This can be done in the following way.
   - Set the "inner loop" pointer to the front of the list of clauses.
   - Set the "outer loop" pointer to the first clause resulting from the negated conclusion.
   - Go to Step 2.
2. Resolve:
   - If the clauses pointed by the two pointers can be resolved, produce the resolvent.

[1] `https://www.cs.utexas.edu/users/novak/asg-reso.html`

- Add the resolvent to the end of the list of clauses and print it out.
- If the resolvent is empty (empty clause), stop; the theorem is proved. Return ”success”.
- Otherwise, go to Step 3.

3. Advance Pointers:
   - Move the ”inner loop” pointer forward one clause.
   - If the ”outer loop” pointer goes beyond the last clause in the list, stop; the theorem cannot be proved. Return ”failure”.
   - If the ”inner loop” pointer has not reached the ”outer loop” pointer, go to the Resolve step (Step 2).
   - Otherwise, reset the ”inner loop” pointer to the front of the list of clauses and move the ”outer loop” pointer forward one clause, then go to the Resolve step (Step 2).

Testing requirement:

1. Test the `prover()` function on the default theorem in the skeleton code.
2. Enter the Unicorn theorem in slide03-logic.pdf (copied below) into your code as a new theorem. Test all three possible outcomes ($M$ or $G$ or $H$: try one conclusion at a time), given the premises (under ”Given”). Don’t forget to negate the conclusion.
3. In all example cases, report how many new clauses were generated before deriving False (empty clause), or failing.

Given:

1. $M \rightarrow I$
2. $\neg M \rightarrow (\neg I \wedge L)$
3. $(I \vee L) \rightarrow H$
4. $H \rightarrow G$

Prove: (conclusion) $H$

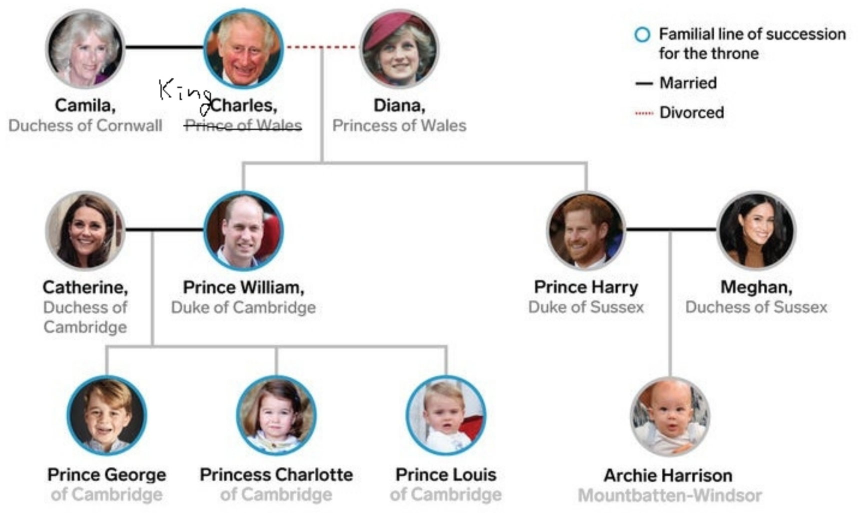
Make sure you turn the above into conjunctive normal form.

## 3 Programming : First-Order Logic with Prolog

See Canvas $\rightarrow$ Quick links and info $\rightarrow$ Homeworks: $\rightarrow$ Homework 3 extras

**Problem 3 (30 pts):** Simple knowledge base in Prolog.

Consider the following family tree (British royal family).



Use Prolog to enter the facts and rules. First, read the Prolog Tutorial in Canvas.

Prolog access:

- Login to `sun.cs.tamu.edu` using putty or some other terminal emulator: run as `gprolog`. (1) Save your code as `family.pl`. (2) Run `gprolog` from the unix prompt. (3) Enter `[family].` (don't forget that "." at the end). (4) Enter your queries.
- Or, use `http://tau-prolog.org/sandbox/` : Prolog sandbox. (1) Enter your code in the window to the left, then click on [Consult Program] (or [Reconsult Program]). (2) Enter your query in the text entry box at the top right. Keep on pressing [Enter] key to retrieve multiple answers.
- Or, install and use GNU Prolog `http://www.gprolog.org/`

**Task 1: define the facts**

**IMPORTANT NOTE:** In prolog, variables are upper case, and predicates, functions, and constants are lower case.

The facts should only include `male(), female(), spouse()`, and `parent()`, directly from the family tree.

Note:

- Facts: For `parent()`, only include biological parent.
- Facts: For `spouse()`, include both current and divorced.
- Rules: For `grandparent()`, define rule to track only biological ancestry.
- Only use the constructs shown below. Do not use advanced Prolog syntax such as `dynamic`, `assertz`, `retract`, etc.
- Do not define unnecessary predicates such as `divorced`.

```
% For constants, use the black bold text in the family tree (first line).
%
% males
male(charles).
male(princewilliam).
...

% females
female(diana).
female(catherine).
...

% spouse
spouse(charles,diana).
spouse(diana,charles).
...

% parent(X,Y) : X is a parent of Y.
parent(charles,princewilliam).
parent(charles,princeharry).
...
```

**Task 2: define the rules**

Define the following rules.

```
% In all predicates below, relation(X,Y) means X is a "relation" of Y.
% For example, child(X,Y) = X is a child of Y.
%
% In some of the rules, you may need X \\== Y (X is not the same as Y).
% - As you go along, use rules defined above it as much as possible.
%
% NOTE: in prolog, variables are uppercase:  X, Y, Z, ...
% constants are lowercase:  charles, diana, ...

mother(X,Y) :-
father(X,Y) :-
brother(X,Y) :-
sister(X,Y) :-
sibling(X,Y) :-
nephew(X,Y) :-
niece(X,Y) :-
grandparent(X,Y) :-
stepmother(X,Y) :-
cousin(X,Y) :-
uncle(X,Y) :-
aunt(X,Y) :-
```

**Task 3: Test your KB**

After you're done entering the facts and rules into a file: `family.pl`, run `gprolog` and enter `[family].`. Once you've loaded the KB, answer these questions, only using the predicates defined in the KB.

1. Who are the grandchildren of Charles?
2. Who is married to Prince William's brother?
3. Who is the uncle of Princess Charlotte?
4. Who is the aunt of Archie Harrison?
5. Who are the nephews of Prince Harry?
6. Who is the stepmother of Prince William?
7. Who has siblings?
8. Who has a female cousin?
9. Who has a niece?

Include the following in the zip file:

- `family.pl` : your KB.
- `family-results.txt` : Your queries and the results.