**Major Course Output 2**
**ChatBot**

**Release Date:** October 6, 2023
**Due Date:** November 19, 2023 (Sun), 11:59 PM

# 1 Project Description

In this project, students will apply the concepts and theories they have learned in the course to develop a chatbot that can understand family relationships. Through this project, students should be able to demonstrate the following learning outcomes:

- **LO3.** Collaboratively build and evaluate an expert system prototype that addresses a need of a particular user group, community, or organization.
- **LO5.** Articulate ideas and present results in correct technical written and oral English.

# 2 Project Specifications

This section contains the specifications for this major course output.

## 2.1 Overview

In this project, students will make use of a logic-based inference system as the backbone of a simple conversational chatbot that can understand family relationships and answer queries about them.

## 2.2 Logic-Based Inference System

For this project, students are expected to write their code in the **Python 3** programming language. You are provided with a module called **logic.py**[1]. This module contains a working implementation of a first-order logic inference system which makes use of resolution inference. The module maintains a knowledge base which consists of a set of formulas expressed in first-order logic. It also provides an interface to dynamically add formulas into the knowledge base, and to verify the truthfulness of a given formula based on the existing facts in the knowledge base.

To use the module, you must first write the following import statement:

```
from logic import *
```

The following table shows how first-order logic formulas are represented within this system:

| **Name** | **Formula** | **Code** |
|---|---|---|
| Constant symbol | Elijah | `"elijah"` (must be lowercase) |
| Variable symbol | $x$ | `"$x"` (must be lowercase preceded by $) |
| Atomic Formula (atom) | Rain | `Atom("Rain")` (predicate must start with uppercase) |
| Atomic Formula (atom) | Likes(Elijah, $X$) | `Atom("Likes", "elijah", $x)` (predicate must start with uppercase) |
| Negation | ¬Rain | `Not(Atom("rain"))` |
| Conjunction | Rain $\bigwedge$ Snow | `And(Atom("rain"), Atom("snow"))` |
| Disjunction | Rain $\bigvee$ Snow | `Or(Atom("rain"), Atom("snow"))` |
| Implication | Rain $\rightarrow$ Wet | `Implies(Atom("rain"), Atom("wet"))` |
| Equivalence | Rain $\leftrightarrow$ Wet | `Equiv(Atom("rain"), Atom("wet"))` |
| Existential Quantification | $\exists X$ Likes(Elijah, $X$) | `Exists("$x", Atom("Likes", "elijah", "$x"))` |
| Universal Quantification | $\forall X$ Likes(Elijah, $X$) | `Forall("$x", Atom("Likes", "elijah", "$x"))` |

Complex formulas can be written by combining the above codes. For example, to write the formula:

$\forall X$ Student($X$) $\rightarrow$ Smart($X$) $\bigwedge$ Young($X$),

We can use:

```
Forall("$x", Implies(Atom("Student", "$x"), And(Atom("Smart", "$x"), Atom("Young", "$x"))))
```

[1]Written Percy Liang for the Stanford CS221 course, with some modifications by Thomas Tiam-Lee

Alternatively, you can create variables and functions to make it more readable:

```
def Student(x):
  return Atom("Student", x)

def Smart(x):
  return Atom("Smart", x)

def Young(x):
  return Atom("Young", x)

X = "$x"

Forall(X, Implies(Student(X), And(Smart(X), Young(X))))
```

There is also a special function `Equals` which can be used to impose that two variables are the same. For example, the formula $\exists X \exists Y$ Likes($X$, $Y$), $X \neq Y$, which can be written as:

```
  Exists("$x", Exists("$y", And(Atom("Likes", "$x", "$y), Not(Equals("$x", "$y")))))
```

means that there exists some $x$ and $y$ such that $x$ likes $y$, but they are different persons. Without the "not equals" condition, the formula consider the scenario that a single person likes himself as a possibility.

The module allows for two main operations: `tell` and `ask`.

The `tell` operation accepts a formula and adds it into the knowledge base (if there is no contradiction). The `ask` operation accepts a formula and attempts to verify its truthfulness based on the facts in the knowledge base. If the formula contains variables, it **returns a dictionary of substitutions that can be made and the corresponding response**.

Both the **tell** and **ask** operations return a `KBResponse` object. The object has a property called `status`, which can either be `"CONTINGENT"`, `"ENTAILMENT"`, or `"CONTRADICTION"`. This table shows each status means in the context of "tell" and "ask":

| **Status** | **Tell** | **Ask** |
|---|---|---|
| `"CONTINGENT"` | learned something new, formula added to KB | Not sure (may be true or false) |
| `"ENTAILMENT"` | already knew that | Yes / true |
| `"CONTRADICTION"` | impossible, formula not added to KB | No / false |

You may refer to this sample Python program for an example usage of the module:

```
from logic import *
Rain = Atom("Rain")           # Shortcut
Wet = Atom("Wet")             # Shortcut
kb = createResolutionKB()     # Create the knowledge base
kb.ask(Wet)                   # Result: not sure
kb.ask(Not(Wet))              # Result: not sure
kb.tell(Implies(Rain, Wet))   # Result: learned something new, added to KB
kb.ask(Wet)                   # Result: not sure
kb.tell(Rain)                 # Result: learned something new, added to KB
kb.tell(Wet)                  # Result: already knew that
kb.ask(Wet)                   # Result: yes
kb.ask(Not(Wet))              # Result: no
kb.tell(Not(Wet))             # Result: impossible, not added to KB
```

Finally, the knowledge base object has a helper function `getAllConstants`, which simply returns all the constants that are currently in the knowledge base. This function may be helpful in answering some of the question prompts for the chatbot. Here is an example usage of the said function:

```
from logic import *
kb = createResolutionKB()                   # Create the knowledge base
kb.tell(Atom("Likes", "romeo", "juliet"))   # Tell operation
kb.tell(Atom("Likes", "romeo", "mary"))     # Tell operation
kb.tell(Atom("Likes", "mary", "bob"))       # Tell operation
kb.getAllConstants()                        # Returns ["juliet", "mary", "bob", "romeo"]
```

## 2.3 The Chatbot

For this project, you are to develop a conversational chatbot that can understand a set of statements and sentence patterns. The user should be able to communicate with the chatbot by giving it a prompt. After which, the chatbot responds accordingly. There should be two kinds of prompts: a statement or a question. Statements allow the user to tell the chatbot new pieces of information. On the other hand, questions allow the user to ask the chatbot certain questions. **To get full credit, make sure that you follow the sentence patterns below exactly.**

The boxes in the sentence patterns below can be replaced with the name of a person. For simplicity, in this project, we will only consider names that do not have spaces and only contain letters. You may also assume that names will always be written such that the first letter is capitalized, while the rest is lowercase. Furthermore, you may assume that names are unique. No two people will have the same name.

### 2.3.1 Statement prompts

Statement prompts can be one of the following sentence patterns:

| **Sentence Pattern** | |
|---|---|
| [ ] and [ ] are siblings.[2] | [ ] is a brother of [ ]. |
| [ ] is a sister of [ ]. | [ ] is the father of [ ]. |
| [ ] is the mother of [ ]. | [ ] and [ ] are the parents of [ ]. |
| [ ] is a grandmother of [ ]. | [ ] is a grandfather of [ ]. |
| [ ] is a child of [ ]. | [ ], [ ] and [ ] are children of [ ].[3] |
| [ ] is a daughter of [ ]. | [ ] is a son of [ ]. |
| [ ] is an uncle of [ ]. | [ ] is an aunt of [ ]. |

The chatbot should respond to a statement prompt as follows. If the information is valid and does not contradict with the known facts, the chatbot should acknowledge that it learned something and add the new piece of information in the knowledge base. If the information contradicts with the known facts, the chatbot should tell the user that it couldn't add that the knowledge base due to the contradiction.

### 2.3.2 Question Prompts

Question prompts always end with a question mark (?) and can be one of the following sentence patterns:

| **Sentence Pattern** | |
|---|---|
| Are [ ] and [ ] siblings? | Who are the siblings of [ ]? |
| Is [ ] a sister of [ ]? | Who are the sisters of [ ]? |
| Is [ ] a brother of [ ]? | Who are the brothers of [ ]? |
| Is [ ] the mother of [ ]? | Who is the mother of [ ]? |
| Is [ ] the father of [ ]? | Who is the father of [ ]? |
| Are [ ] and [ ] the parents of [ ]? | Who are the parents of [ ]? |
| Is [ ] a grandmother of [ ]? | Is [ ] a grandfather of [ ]? |
| Is [ ] a daughter of [ ]? | Who are the daughters of [ ]? |
| Is [ ] a son of [ ]? | Who are the sons of [ ]? |
| Is [ ] a child of [ ]? | Who are the children of [ ]? |
| Are [ ], [ ] and [ ] children of [ ]? | Is [ ] an aunt of [ ]? |
| Is [ ] an uncle of [ ]? | Are [ ] and [ ] relatives? |

The chatbot should respond to a question prompt by providing the appropriate answer. For yes/no questions, the chatbot should answer either "yes", "no", or "not sure". Note that there is a difference between "no" and "not sure". The former means it is impossible for the query to be true, while the latter means it may be true or false.

[2] Sibling means at least one parent must be the same.
[3] Can be two or more children

### 2.3.3 Chatbot Implementation

You should implement the chatbot as a Python program. Graphical user interface (GUI) is allowed, but not required. You are free to add as many additional files as you want, but you are **NOT** allowed to modify `logic.py`. **You must make use of the provided `logic.py` module to handle the inference needed for the program.** Do not re-implement any of the logic that is already handled by `logic.py`. In general, the main tasks in this project are as follows:

1. Encoding the relationship rules into the knowledge base as first order logic formulas
2. Parsing the input prompts and translating them into the appropriate first order logic formulas.
3. Making the appropriate queries to the knowledge base in order to update the facts or answer the questions.

If invalid prompts are provided (i.e., does not follow any of the given sentence patterns), the program should not crash and should respond instead with an error message. You are free to add additional prompts beyond the ones given, as long as they do not conflict with any prompts in the list.

You must make sure that the chatbot behaves intelligently. That is, it should be able to infer facts based on real-world concepts even if they were not explicitly stated. The following are some examples of intelligent behavior that should be handled by the chatbot:

**Example 1:**

```
> Bob is the father of Alice.
OK! I learned something.
> Alice is the mother of John.
OK! I learned something.
> Is Bob a grandfather of John?
Yes!
```

In this example, even though it wasn't explicitly stated that Bob is a grandfather of John, the chatbot inferred this based on the relationships that were given.

**Example 2:**

```
> Mark is the father of Patricia.
OK! I learned something.
> Mark is a daughter of Ann.
That's impossible!
```

In this example, Mark couldn't be a daughter of Ann, because Mark is male (since he was already established to be a father).

**Example 3:**

```
> One is the father of Two.
OK! I learned something.
> Two is the father of Three.
OK! I learned something.
> Three is the father of One.
That's impossible!
```

In this example, the given set of relationships is just not possible in real life.

**Example 4:**

```
> Lea is the mother of Lea.
That's impossible!
```

Lea couldn't have given birth to herself.

Note that there are other rules that you should capture apart from these examples. Part of this project is for you to spend some time thinking deeply on how real-world knowledge is translated into first order logic. To get full credit, your chatbot should be able to exhibit many cases of intelligent inference.

# 3 Report

In addition to the chatbot program, you are required to write a report documenting the implementation of the project. At the minimum the report should contain the following:

1. **Brief Introduction:** A short introduction describing the chatbot that was developed. Please make this section short and concise.
2. **Knowledge Base:** A section listing the formulas that were encoded into the knowledge base. You must enumerate and describe each formula. Use the standard first order logic notation when writing the formulas.
3. **Chatbot Implementation**: A section describing how the chatbot was implemented, including how it parses the prompts and how it queries the knowledge base to provide intelligent answers.
4. **Results**: A section describing the chatbot in action, highlighting its capabilities and aspects of intelligence as well as identifying its weaknesses.

5. **Limitations and Challenges**: A discussion on the limitations of the chatbot that was developed, especially when compared to large language model (LLM) based chatbots like ChatGPT. What are the challenges that make these limitations difficult to overcome?
6. **Conclusion:** A conclusion section summarizing the process of implementing the project, including reflections, realizations, and insights obtained.
7. **Table of Contributions:** A table showing the contributions of each member to the project.

There is no required format for the report, but please refrain from submitting very long reports with little substance. Please minimize copying or paraphrasing already known concepts and theories to make the report longer. The bulk of the report should contain the group's personal ideas and insights.

# 4 Deliverables

There are two deliverables for this project: a zip file containing the source files for the project, and a pdf file containing the report. The source files should include:

1. All the source code and files needed to run the project (this includes the unmodified `logic.py`).
2. An `instructions.txt` file which contains instructions on how to run the program, as well as any other non-trivial information needed to operate the program properly.

The file name of the zip file should be: `mco2_<surname 1>_<surname 2>_<surname 3>_<surname 4>.zip` with the surnames in alphabetical order. The file name of the report should be `mco2_<surname 1>_<surname 2>_<surname 3>_<surname 4>.pdf` with the surnames in alphabetical order.

# 5 Academic Honesty

Honesty policy applies. Please take note that you are **NOT** allowed to borrow and/or copy-and-paste in full or in part any existing related program code from the internet or other sources (such as printed materials like books, or source codes by other people that are not online). **Violating this policy is a serious offense in De La Salle University and will result in a grade of 0.0 for the whole course**.

Please remember that the point of this project is for all members to learn something and increase their appreciation of the concepts covered in class. Each member is expected to be able to explain the different aspects of their submitted work, whether part of their contributions or not. **Failure to do this will be interpreted as a failure of the learning goals, and will result in a grade of 0 for that member**.