# CSC442: Intro to AI
# Project 2: Automated Reasoning

October 6, 2019

In this project, you will implement at least two inference methods for Propositional Logic and demonstrate them on some example problems.

## Part I: Representation

You are being asked to implement a propositional theorem prover. Therefore, your program will be concerned with formulas. Think carefully about how you might represent a formula as a data structure. I suggest representing them as trees.

## Part II: Basic Model Checking

The first technique you must implement is the "truth-table enumeration method" described in AIMA Figure 7.10. The algorithm is conceptually straightforward—you may not really even need their pseudo-code. We also discussed this in class, and it is presented in the lecture slides.

You will need to represent a "model" (a.k.a. possible world, or assignment of truth values). And then you need to use that representation in the implementation of the algorithm. This is a very important design point. I suggest that you do something clear and correct but not necessarily efficient first, then improve it as time permits and your problems demand. If you use a language like Java, C++, or Python you may be particularly well served by the Set, Vector, or Hash-table/Dictionary data structures.

You must test your implementation on at least four of the problems described below.

# Part III: Advanced Propositional Inference

For the second part of this assignment, you must implement one of the other techniques for Propositional Inference described in class and in the textbook. Your choices are:

- The DPLL algorithm for checking satisfiability (AIMA Figure 7.17)
- The WalkSAT algorithm, also for SAT checking (AIMA Figure 7.18)
- A resolution-based theorem prover (AIMA Figure 7.12)
- A theorem prover based on other inference rules (if you do this, be sure to explain what you're doing in detail in your writeup)

You can use the same classes for representing Propositional Logic sentences as you did for Part I.

As with Part I, you must test your implementation on at least four of the problems described below. Use the same problems and compare the techniques in your writeup. If you can't use the same problems, explain why in your writeup.

# Data Structures for Reasoning

You should think about how you would represent Propositional Logic sentences in your program. This is one of the things that makes AI programming interesting. It's not rocket science, but it does force you to think clearly about different levels of representation: Java/C++ as programming language, Propositional Logic as "programming language", the abstraction hierarchy of your classes, etc.. Doing it relatively efficiently is an additional challenge.

It is also possible to write parsers that read various textual representations of Propositional Logic sentences and create the internal representations (data structures) used by your programs. These include, for example, a bracketed, prefix Lisp- or Scheme-like notation, an infix notation like we've used in class.

This is useful but not necessary for this project. For this project you may create the knowledge bases yourself in code. In particular, when applying your functions to the problems at the end, you are allowed to hard-code the representations.

Several of these techniques require the knowledge base to be converted to clauses (conjunctive normal form, CNF). You can hand-engineer your knowledge bases to be in CNF.

# Selected Problems

The following problems are all amenable to solution using Propositional Logic using both theorem proving and satisfiability testing approaches. You must demonstrate the power of your theorem prover by applying it to each problem below. For each problem, you must demonstrate the correctness of both model checking and resolution. Note that each inference mechanism logically corresponds to a function which returns TRUE, FALSE, or MAYBE. You must write a driver program (or function) which prints the name of the problem, states the axioms of the problem, and then answers each question.

For example, a transcript of the first sample problem might look like:

```
Modus Ponens test.
Knowledge base:
P => Q
P
Query: P
Ans with model checking: TRUE
Ans with resolution: TRUE
```

The four selected problems are listed below.

1. **Modus Ponens**: Show that $\{P, P \Rightarrow Q\} \vDash Q$.

2. **Wumpus World (Simple)**: Consider the following facts using the

Wumpus World predicates from AIMA:

$$\begin{aligned}
&\neg P_{1,1} \\
&B_{1,1} \iff (P_{1,2} \vee P_{2,1}) \\
&B_{2,1} \iff (P_{1,1} \vee P_{2,2} \vee P_{3,1}) \\
&\neg B_{1,1} \\
&B_{2,1}
\end{aligned}$$

Prove whether $P_{1,2}$ is true or not (that is, whether there is a pit at location [1,2]).

The textbook gives several other examples of knowledge bases and queries using the Wumpus World that you can also try.

3. **Horn Clauses** (Russell & Norvig) If the unicorn is mythical, then it is immortal, but if it is not mythical, then it is a mortal mammal. If the unicorn is either immortal or a mammal, then it is horned. The unicorn is magical if it is horned.

   (a) Can we prove that the unicorn is mythical?

   (b) Can we prove that the unicorn is magical?

   (c) Can we prove that the unicorn is horned?

4. **The Doors of Enlightenment** (from CRUX 357): There are four doors $X$, $Y$, $Z$, and $W$ leading out of the Middle Sanctum. At least one of them leads to the Inner Sanctum. If you enter a wrong door, you will be devoured by a fierce dragon. Well, there were eight priests $A$, $B$, $C$, $D$, $E$, $F$, $G$, and $H$, each of whom is either a knight or a knave. (Knights always tell the truth and knaves always lie.) They made the following statements to the philosopher:

   - $A$: $X$ is a good door.
   - $B$: At least one of the doors $Y$ or $Z$ is good.
   - $C$: $A$ and $B$ are both knights.
   - $D$: $X$ and $Y$ are both good doors.
   - $E$: $X$ and $Z$ are both good doors.

- $F$: Either $D$ or $E$ is a knight.
- $G$: If $C$ is a knight, so is $F$.
- $H$: If $G$ and $I$ (meaning $H$) are knights, so is $A$.

(a) Smullyan's problem: Which door should the philosopher choose?

(b) Liu's problem: The philosopher lacked concentration. All he heard was the first statement ($A$'s) and the last statement ($H$'s) plus two fragments:

- $C$: $A$ and ... are both knights.
- $G$: If $C$ is a knight, ...

Prove that he had heard enough to make a decision.

# Extra Credit

There are two opportunities for extra credit for this project. You may do zero, either, or both. The maximum grade you can receive on this project is 120%.

- Implement a parser for WFFs. You must write a module which accepts strings as input (e.g., "`P ^ Q) =>` R v !S)" and parses them to build (and return) a tree-structured representation of the formula. If you do this, then your program will be capable of operating using *ask* and *tell*, as discussed in class. You can receive up to 10% extra credit for a wff parser.

- Implement CNF conversino for WFFs. You must write a module which automatically converts an underlying knowledge base into conjunctive normal form. Note that you can do this either by modifying the KB, or by directly extracting a set of clauses. If you do this, then you must augment your resolution module to use your automatically constructed CNF representation (as opposed to a by-hand initialization). You can receive up to 10% extra credit for implementing automatic CNF conversion.

## Writeup

Lastly, you must produce a short PDF writeup with at least the following four sections:

1. Key Datastructures (Describe how you represented formulas, models, and clauses)

2. Performance (Include your programs answers to the four questions. Include results for model checking and resolution).

3. Member contributions (describe each team member's contributions to the project)

## Submission Instructions

Upload a ZIP archive with everything (source code and writeup) to BlackBoard before the deadline.

Place all your source code into a directory called `src`. You must also include a plain-text `README` file in your src directory which describes how to build and run your program, and includes the names and UR NetID's for each contributing team member. Zip your `src` directory, `README`, and PDF writup into a single file and upload to blackboard.

Submissions received by 11:59PM Sunday, October 27th will receive a flat 5% bonus credit. Submissions will be accepted until 11:59PM, Wednesday, October 30th.

# Grading

| Amount | Category | Subcat |
|---|---|---|
| 20 | | Writeup |
| 40 | | Model Checking |
| | 10 | Permutations |
| | 15 | Model relation |
| | 15 | Correctness of entailment |
| 40 | | Resolution |
| | 10 | Refutation Completeness |
| | 10 | Correct resolvents |
| | 10 | Factoring |
| | 10 | Correctness of entailment |
| +10 | | Formula Parsing |
| +10 | | CNF Conversion |