**CSC 261** **Lab 5: Propositional Logic** **Spring 2020**

**Assigned:** Tuesday 18 February

**Due:** Wednesday 26 February, 10:30 p.m.

**Objectives:**

- Understand a propositional logic **representation** (implementation)
- Reinforce the distinction and relation between **sentences** and **models**
- Practice **translating** statements into sentences of propositional logic
- Learn how to use enumeration and resolution to **prove** propositions

**Collaboration:**

- Problem 1 (programming) will be completed in pairs assigned by the instructor.
- Problems 2 and 3 (inference) must be completed individually.

# Introduction

One simple (if inefficient) way to determine entailment in propositional logic is by enumerating all possible models to verify the truth value of a proposition under all of them. In this lab, we'll be implementing this technique in Scheme. Therefore, we will need a way to represent both sentences of propositional logic and models.

In Scheme, we will represent a **sentence** as either a boolean value, a Scheme symbol (representing a propositional variable), or a list. When the sentence is a list, the first item must be one of the Scheme symbols `not`, `and`, `or`, `=>`, or `<=>`. If the symbol is `not`, the list must have only a second element (another sentence). Otherwise the list must have two subsequent elements, both sentences, that represent the arguments to the specified logical connective.

More formally, we define a sentence recursively as follows:

```
sentence := boolean |
            symbol |
            (! sentence) |
            (^ sentence sentence) |
            (v sentence sentence) |
            (=> sentence sentence) |
            (<=> sentence sentence)
```

For example, the Scheme representation of the sentence $P \Leftrightarrow Q$ would be given by

```
> (list '<=> 'P 'Q)
'(<=> P Q)
```

or more simply

```
> '(<=> P Q)
'(<=> P Q)
```

while the sentence $P \vee (Q \wedge R)$ could be expressed as

```
> (list 'v 'P (list '^ 'Q 'R))
'(v P (^ Q R))
```

or more directly

```
> '(v P (^ Q R))
'(v P (^ Q R))
```

A **model** is an association list whose keys (`cars` of its elements) are symbols and whose values (`cdrs` of its elements) are booleans. For example, the model for the second line in AIMA Figure 7.8 (p. 246) would be given by

```
>  (list (cons 'P #f) (cons 'Q #t))
'((P . #f) (Q . #t))
```

# Code and environment

The starter code for this assignment may be found in

```
~weinman/courses/CSC261/code/enumeration
```

In particular the file `logic.rkt` provides a method `(get-symbols` *`sentence`*`)` that produces a list of all the symbols in a sentence of the form given above. In addition, `(set-union` *`list1 list2`*`)` from `racket/set` can be used to merge lists of symbols (without repetition).

# Assignment

## Problem 1: Implementing enumeration [40 points]

In addition to the problems below, 12 points will be allotted for program style (comments, readability, formatting and organization, variable names, etc.).

### Part A: PL-True? [8 points]

Implement the Scheme version of PL-True?, `(pl-true?` *`sentence model`*`)` as documented in the starter file `enumeration.rkt`. Your implementation should recursively determine whether *`sentence`* is true in *`model`*. As a precondition, every symbol in *`sentence`* must have an entry in *`model`*.

### Part B: TT-Check-All [8 points]

Implement `(tt-check-all` *`knowledge-base query symbols model`*`)`, the Scheme version of TT-Check-All of AIMA Fig. 7.10 (p. 248), as documented in `enumeration.rkt`.

### Part C: TT-Entails? [4 points]

Implement `(tt-entails?` *`knowledge-base query`*`)`, the Scheme version of TT-Entails? of AIMA Fig. 7.10 (p. 248), as documented in `enumeration.scm`. Note that you will need to use `get-symbols` and `set-union` to get "a list of the proposition symbols in $KB$ and $\alpha$."

### Part D: Testing [8 points]

Write a test driver program `driver.rkt` that tests the functionality of all your procedures. Your tests should cover all paths through your procedures.

## Problem 2: Using enumeration [32 points]

You may NOT use your own implementation of `tt-entails?` for problem 2; instead, use a compiled version from `~weinman/courses/CSC261/code/enumeration/compiled`, which you'll copy during the lab prep. To access the precompiled version of `tt-entails?`, add

```
(require "weinman-enumeration.rkt")
```

to a Racket file `problems.rkt` in your `propositional/` directory.

In all the problems below, **use clear symbol names**, just as you would variable names in programs, and **add comments** to document the meaning/interpretation of your knowledge-base components.

### Part A: Unicorns! [8 points]

Translate the following paragraph into a (Scheme-based) knowledge base of propositional logic (i.e., using the form above) and call it `unicorn-kb`.

> 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.

Using this knowledge base and `tt-entails?`, determine whether you can prove the unicorn is mythical, magical, or horned.[1] (That is, prove each proposition individually, not their disjunction).

### Part B: Tarts without Salt [10 points]

Translate the following information into a knowledge-base called `salt-kb`.

> "Well," said the King, "here is your sugar, so you can make me the tarts."
>
> "Without salt?" asked the Queen.
>
> So! The salt had also been stolen! Well, this time it was found that the culprit was either the Caterpillar, Bill the Lizard, or the Cheshire Cat. (One of them had come into the kitchen and eaten up all the salt; the container wasn't missing.) The three were tried and made the following statements in court:
>
> > CATERPILLAR: Bill the Lizard ate the salt.
> > BILL THE LIZARD: That is true!
> > CHESHIRE CAT: I never ate the salt!
>
> As it happened, at least one of them lied and at least one told the truth.[2]

Using this knowledge base and `tt-entails?`, prove who stole the salt and the others' innocence.

### Part C: The Inner Sanctum [14 points]

**i.** Translate the following information into a knowledge-base called `doors-kb`.

> 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[3] are knights, so is $A$.[4]

[1] From J. Barwise and J. Etchemendy, *The Language of First-Order Logic*, Third Edition, CSLI, 1983, as given in S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach,* Third Edition, Prentice Hall, Exercise 7.2, p. 280.

[2] From R. Smullyan, "The Fifth Tale", *Alice in Puzzle-Land*, Penguin Books (1982), pp. 12-13.

[3] "I" means $H$ referring to himself.

[4] From R. Smullyan, "The Island of Baal", *What is the name of this book?*, Prentice Hall (1978), p. 141.

Using this knowledge base and `tt-entails?`, determine which door(s) the philosopher should choose to reach the Inner Sanctum.

**ii.** It turns out the philosopher had poor concentration. All he heard was the first statement (from $A$) and the last statement (from $H$) along with two snippets:

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

Encode this more limited information as `doors-2-kb` and prove the philosopher was still fortunate to have heard enough to make a decision.[5]

## Problem 3: Alternative approaches [12 points]

### Part A: CNF

Re-write your propositional knowledge base represented by `unicorn-kb` in CNF. Use the typical propositional format, rather than our Scheme form.

### Part B: Resolution

Use resolution to prove (or disprove) that the unicorn is horned. Include the resolvents of each step.

# What to turn in

### Programming Assignment

In addition to `references.txt`, your joint submission should include the following:

- Your completed `enumeration.rkt` file
- A `driver.rkt` file with a test program demonstrating your enumeration procedures are correct
- A transcript `output.txt` of your test driver program's output

### Application Assignment

In addition to `references.txt`, your individual submission should include the following:

- File `problems.rkt` with your Problem 2 knowledge bases defined and commands for output indicating `tt-entails?`-based proofs
- A transcript `output.txt` of your Scheme knowledge-base program's output
- A single PDF `logic.pdf` containing solutions of problem 3, parts A and B



[5] From Crux Mathematicorum 357, as given in T.W. Neller, Z. Markov, and I. Russel, "Clue deduction: an introduction to satisfiability reasoning", 2005, p. 13.