Assigned: Tuesday 1 October 2013

Due: Monday 7 October 2013, 11:59 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 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 symbol (representing a variable), or a list. When the sentence is a list, the first item must be one of the 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 |
            (not sentence) |
            (and sentence sentence) |
            (or 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 'or 'P (list 'and 'Q 'R))
(or P (and Q R))
```

or more directly

```
> '(or P (and Q R))
(or P (and Q R))
```

A **model** is an association list whose keys (`car`s of its elements) are symbols and whose values (`cdr`s 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.scm` provides a method `(get-symbols` *`sentence`*`)` that produces a list of all the symbols in a sentence of the form given above. In addition, `(list-union` *`list1 list2`*`)` 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.scm`. 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.scm`.

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 `list-union` to get "a list of the proposition symbols in KB and α."

Part D: Testing [8 points]

Write a driver program that tests the functionality of all your procedures. Your tests should indicate what is being tested, compare or verify the expected result, and cover all paths through your procedures.

Problem 2: Using enumeration [24 points]

You may use your own implementation of `tt-entails?` for problem 2, or you may use a compiled version from `~weinman/courses/CSC261/code/enumeration/compiled`, which you'll copy during the lab prep. Add `(require "weinman-enumeration.scm")` to a Scheme file in your `propositional` directory to access the precompiled version of `tt-entails?`.

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 (individually) the unicorn is mythical, magical, or horned.[1]

Part B: Portia's Caskets [8 points]

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

> In Shakespeare's *Merchant of Venice* Portia had three caskets—gold, silver, and lead—inside one of which was Portia' s portrait. The suitor was to choose one of the caskets, and if he was lucky enough (or wise enough) to choose the one with the portrait, then he could claim Portia as his bride. On the lid of each casket was an inscription to help the suitor choose wisely. Now, suppose Portia wished to choose her husband not on the basis of virtue, but simply on the basis of intelligence. She had the following inscriptions put on the caskets.
>
> **Gold:** The portrait is in this casket
> **Silver:** The portrait is not in this casket
> **Lead:** The portrait is not in the gold casket
>
> Portia explained to the suitor that of the three statements, at most one was true.[2]

Using this knowledge base and `tt-entails?`, determine which casket should the suitor should choose.

Part C: Knights, Knaves, and Werewolves [8 points]

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

> There are many situations in life in which it is good to have one' s wits about one. So I shall now give you detailed, step-by-step instructions which will teach you ... how to avoid werewolves in the forest.
>
> Of course, I cannot absolutely guarantee that you will actually meet with any of these situations, but as the White Knight wisely explained to Alice, it is well to be provided for *everything.*
>
> Suppose you are visiting a forest in which every inhabitant is either a knight or a knave. (We recall that knights always tell the truth and knaves always lie.) In addition, some of the inhabitants are werewolves and have the annoying habit of sometimes turning into wolves at night and devouring people. A werewolf can be either a knight or a knave.
>
> Again, each of A,B, C is a knight or a knave and exactly one of them is a werewolf. They make the following statements:
>
> A: I am a werewolf.
> B: I am a werewolf.
> C: At most one of us is a knight.[3]

[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 Mystery of Portia's Caskets", *What is the name of this book?*, Prentice Hall (1978), p. 55.

[3] From R. Smullyan, "How to Avoid Wherewolves–and Other Bits of Practical Advice", *What is the name of this book?*, Prentice Hall (1978), pp 82-83.

Using this knowledge base and `tt-entails?`, give a complete classification of A, B, and C as either knight or knave and werewolf or human.

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) the unicorn is horned.

What to turn in

Programming Assignment

Your joint submission should include the following

- Your completed `enumeration.scm` file
- A `.scm` file with a test driver program that demonstrates your enumeration procedures are correct
- A single PDF containing (merged)
  - Your Scheme files
  - A transcript of your test driver program's output

Application Assignment

Your individual submission should include the following

- A `.scm` file with your Problem 2 knowledge bases defined and output indicating `tt-entails?` based proofs
- A single PDF containing (merged)
  - Your Scheme file
  - A transcript of your Scheme knowledge-base program's output
  - Solutions of problem 3, parts A and B