Assigned: Tuesday 27 September 2011

Due: Monday 3 October 2011 11:59 pm

Objectives:

- Understand a propositional logic representation (implementation)
- Reinforce the distinction and relation between sentences and models

Collaboration: This laboratory will be completed in pairs of your choice. If you do not have a partner, one will be assigned.

1 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.

A **sentence** is 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, a sentence is defined 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 of its elements) are symbols and whose values (cdr of its elements) are booleans. For example, a 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))
```

2 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.

3 Implementing enumeration

Problem 1: PL-True?

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`*.

Problem 2: TT-Check-All

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`.

Problem 3: TT-Entails?

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."

What to turn in

Your submission should include the following

- Your completed `enumeration.scm` file
- A short 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