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:

\[
\text{sentence} ::= \text{boolean} \mid \text{symbol} \mid (\text{not} \ \text{sentence}) \mid (\text{and} \ \text{sentence} \ \text{sentence}) \mid (\text{or} \ \text{sentence} \ \text{sentence}) \mid (\Rightarrow \ \text{sentence} \ \text{sentence}) \mid (\Leftrightarrow \ \text{sentence} \ \text{sentence})
\]

For example, the Scheme representation of the sentence \( P \leftrightarrow Q \) would be given by

\[
> \text{(list '=> 'P 'Q)}
\text{(=> P Q)}
\]

or more simply

\[
> (\Leftrightarrow P Q)
\text{(=> P Q)}
\]

while the sentence \( P \lor (Q \land R) \) could be expressed as

\[
> (\text{list 'or 'P (list 'and 'Q 'R)})
\text{(or P (and Q R))}
\]

or more directly

\[
> (\text{or P (and Q R)})
\text{(or P (and Q R))}
\]

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

\[
> \text{(list (cons 'P #f) (cons 'Q #t))}
\text{((P . #f) (Q . #t))}
\]
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