Massachusetts Institute of Technology

16.410-13 Principles of Autonomy and Decision Making

Assignment #6, tutorial. Due: LEC #11

Problem PS.6.1.1: Propositional Sentences and Models

Consider a vocabulary with only four propositions: A, B, C, and D. We will investigate a number of sentences containing some or all of these propositions, and the models for which these sentences are true. Sentences will be expressed in a Scheme-like prefix form, but this is easy to convert to and from the infix form used in the Russell and Norvig text.

Part 1: First sentence

Given the four propositions (A, B, C, and D), consider the simple sentence (and A B). The truth table for this is:

```
A  B  C  D  |  (and A B)
-----------------------
F  F  F  F  |  F
F  F  F  T  |  F
F  F  T  F  |  F
F  F  T  T  |  F
F  T  F  F  |  F
F  T  F  T  |  F
F  T  T  F  |  F
F  T  T  T  |  F
T  F  F  F  |  F
T  F  F  T  |  F
T  F  T  F  |  F
T  F  T  T  |  F
T  T  F  F  |  T
T  T  F  T  |  T
T  T  T  F  |  T
T  T  T  T  |  T
```

Indicate which of the following models are true for the sentence:

(or (and A B) (and C D))

```
   A B C D
```

1. F F F F
2. F F F T
3. F F T F
4. F F T T
5. F T F F
6. F T F T

7. F T T F
8. F T T T
9. T F F F
10. T F F T
11. T F T F
12. T F T T
13. T T F F
14. T T F T
15. T T T F
16. T T T T

Part 2: Second sentence

Indicate which of the following models are true for the sentence:

(or A B)

A B C D

1. F F F F
2. F F F T
3. F F T F
4. F F T T
5. F T F F
6. F T F T
7. F T T F
8. F T T T
9. T F F F
10. T F F T
11. T F T F
12. T F T T
13. T T F F
14. T T F T
15. T T T F
16. T T T T

Part 3: Third sentence

Indicate which of the following models are true for the sentence:

(A <-> B) <-> C

(<-> means implication)

A B C D

1. F F F F
2. F F F T
3. F F T F
4. F F T T
5. F T F F
6. F T F T
7. F T T F
8. F T T T
9. T F F F
10. T F F T
11. T F T F
12. T F T T
13. T T F F
14. T T F T
15. T T T F
16. T T T T

Problem PS.6.2.1: DPLL Predicates

For this problem, you will implement a few predicates needed for the DPLL algorithm as described in the Russell and Norvig text, Chapter 7. First, consider how the principle data structures should be represented. The dpll algorithm, as described in Russell and Norvig, is a recursive procedure that takes three arguments: symbols, sentence, and model. Symbols should be a list of symbols, for example:

```
(a foo bar d)
```

Sentence (called clauses in the text) is assumed to be in CNF form, and should be a list of lists, where each nested list represents a disjunctive clause, and the overall list represents the conjunction of these disjunctions. Each disjunctive clause is a list of propositions, where a proposition is either a symbol, or its negation, which is represented using the not operator. For example, the negation of the proposition a is (not a). Thus, the example sentence

```
((a b) ((not c) d))
```

means (and (or a b) (or (not c) d)).

Model should be an alist with elements of the form (symbol . truth-value). For example,

```
((a . #t) (b . #f))
```

means that a is true in this model, and b is false.

Using the code in ps6, you will be asked to fill in the implementation of various functions. Feel free to add any auxilliary functions you think you might need. Take a look at the file useful.scm, which has a number of utility functions you may want to use in your solutions. You may find the functions `any?` and `every?` especially helpful.

Please also hand in a paper copy of your code as part of this problem set.

```
;; return true if _any_ of the propositions is true in model.
(define (clause-true? clause model)
  ;; YOUR CODE HERE
  #f)

;; return true if clause is false - which requires that all of its
;; disjuncts (propositions) be false.
(define (clause-false? clause model)
  ;; YOUR CODE HERE
  #f)
(define (sentence-true? sentence model)
  ;; YOUR CODE HERE
  #f)

(define (sentence-false? sentence model)
  ;; YOUR CODE HERE
  #f)
```

Problem PS.6.3.1: DPLL find-unit-clause

For this problem, you will implement `find-unit-clause`. A unit clause is a clause with all but one of its propositions definitely false (and one remaining proposition with an unassigned variable). Return a dotted pair of the unbound symbol and required truth value. Return #f if no unit clause is found. In order to automatically test your implementation, you need to return unit clause variable assigments in the same order as the canonical implemenation. To do this, you must implement find-unit-clause so that it finds **all** of the unit clauses in a given sentence and returns the binding for the least (according to `symbol?`) symbol. Look at the implementation of `find-pure-symbol` to see how it uses bsort to sort the possible returns and select the 1st pair of the sorted list. Before returning a (symbol . value) pair from `find-unit-clause`, cons the pair onto `*unit-clauses*`, which will used to check your answer. In other words, `*unit-clauses*` should contain a list of (symbol . value) pairs, in reverse order of when they were found by `find-unit-clause`.

```
(define (find-unit-clause sentence model)

  ;; YOUR CODE HERE
```

```
  ;; ... (cons var-symbol-pair *unit-clauses*) ...

)
```

Problem PS.6.4.1: Hours

We want to understand how much time it took students to answer the questions on the problem sets.

> Approximately how many hours did you spend really working on this problem set?