- (QiuXin Hui) 7.2 Given the following, can you prove that the unicorn is mythical? How about magical? Horned? Decide what you think the right answer is yourself, then show how to get the answer using both (1) model-checking and (2) inference rule methods. “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.”

7.2 As human reasoners, we can see from the first two statements, that if it is mythical, then it is immortal; otherwise it is a mammal. So it must be either immortal or a mammal, and thus horned. That means it is also magical. However, we can't deduce anything about whether it is mythical. To proide a formal answer, we can enumerate the possible worlds ($2^5 = 32$ of them with 5 proposition symbols), mark those in which all the assertions are true, and see which conclusions hold in all of those. Or, we can let the machine do the work—in this case, the Lisp code for propositional reasoning:

```
> (setf kb (make-prop-kb))
#S(PROP-KB SENTENCE (AND))
> (tell kb "Mythical => Immortal")
T
> (tell kb "~Mythical => ~Immortal ^ Mammal")
T
> (tell kb "Immortal | Mammal => Horned")
T
> (tell kb "Horned => Magical")
T
> (ask kb "Mythical")
NIL
> (ask kb "~Mythical")
NIL
> (ask kb "Magical")
T
> (ask kb "Horned")
T
```

- (Matthew Nguyen) 7.19 A sentence is in **disjunctive normal form** (DNF) if is is the disjunction of conjunctions of literals. For example, the sentence $(A \wedge B \wedge \neg C) \vee (\neg A \wedge C) \vee (B \wedge \neg C)$ is in DNF.
  - (a) Any propositional logic sentence is logically equivalent to the assertion that some possible world in which it would be true is in fact the case. From this observation, prove that any sentence can be written in DNF.
  - (b) Construct an algorithm that converts any sentence into DNF. (Hint: the algorithm is similar to the one that converts to CNF.)
  - (c) Construct a simple algorithm that takes as input a sentence in DNF and returns a satisfying assignment if one exists, or reports that no satisfying assignment exists.
  - (d) Apply the algorithms in (b) and (c) to the following sentence:
    - $(A \Rightarrow B) \wedge (B \Rightarrow C) \wedge (C \Rightarrow \neg A)$

7.19

a. Each possible world can be expressed as the conjunction of all the literals that hold in the model. The sentence is then equivalent to the disjunction of all these conjunctions, i.e., a DNF expression.

b. A trivial conversion algorithm would enumerate all possible models and include terms corresponding to those in which the sentence is true; but this is necessarily exponential-time. We can convert to DNF using the same algorithm as for CNF except that we distribute $\wedge$ over $\vee$ at the end instead of the other way round.

c. A DNF expression is satisfiable if it contains at least one term that has no contradictory literals. This can be checked in linear time, or even during the conversion process. Any completion of that term, filling in missing literals, is a model.

d. The first steps give

$$(\neg A \vee B) \wedge (\neg B \vee C) \wedge (\neg C \vee \neg A) \ .$$

Converting to DNF means taking one literal from each clause, in all possible ways, to generate the terms (8 in all). Choosing each literal corresponds to choosing the truth value of each variable, so the process is very like enumerating all possible models. Here, the first term is $(\neg A \wedge \neg B \wedge \neg C)$, which is clearly satisfiable.

e. The problem is that the final step typically results in DNF expressions of exponential size, so we require both exponential time AND exponential space.

- (Matthew Tseng) 7.17 A propositional *2-CNF* expression is a conjunction of clauses, each containing exactly 2 literals; for example:
  - $(A \vee B) \wedge (\neg A \vee C) \wedge (\neg B \vee D) \wedge (\neg C \vee G) \wedge (\neg D \vee G)$
  - (a) Prove using resolution that the above sentence entails *G*.
- (b) Two clauses are *semantically distinct* if they are not logically equivalent. How many semantically distinct 2-CNF clauses can be constructed from *n* propositional symbols?
- (c) Using you answer to (b), prove that propositional resolution always terminates in time polynomial in *n* given a 2-CNF sentence containing no more than *n* distinct symbols.
- (d) Explain why the argument in (c) does not apply to 3-CNF.

7.17

a. The negated goal is $\neg G$. Resolve witht he last two clauses to produce $\neg C$ and $\neg D$. Resolve with the second and third clauses to produce $\neg A$ and $\neg B$. Resolve these successively against the first clause to produce the empty clause.

b. This can be answered with or without $True$ and $False$ symbols; we'll omit them for simplicity. First, each 2-CNF clause has two places to put literals. There are $2n$ distinct literals, so there are $(2n)^2$ syntactically distinct clauses. Now, many of these clauses are semantically identical. Let us handle them in groups. There are $C(2n, 2) = (2n)(2n - 1)/2 = 2n^2 - n$ clauses with two different literals, if we ignore ordering. All these

clauses are semantically distinct except those that are equivalent to $True$ (e.g., $(A \vee \neg A)$), of which there are n, so that makes $2n^2 - 2n + 1$ clauses with distinct literals. There are $2n$ clauses with repeated literals, all distinct. So there are $2n^2 + 1$ distinct clauses in all.

c. Resolving two 2-CNF clauses cannot increase the clause size; therefore, resolution can generate only $O(n^2)$ distinct clauses before it must terminate.

d. First, note that the number of 3-CNF clauses is $O(n^3)$, so we cannot argue for nonpolynomial complexity on the basis of the number of different clauses! The key observation is that resolving two 3-CNF clauses can *increase* the clause size to 4, and so on, so clause size can grow to $O(n)$, giving $O(2^n)$ possible clauses.

- (Braden Hiebert) 7.8 We have defined four binary logical connectives.
  - Are there any others that might be useful?
  - How many binary connectives can there be?
  - Why are some of them not very useful?

7.8 A binary logical connective is defined by a truth table with 4 rows. Each of the four rows may be true or false, so there are $2^4 = 16$ possible truth tables, and thus 16 possible connectives. Six of these are trivial ones that ignore one or both inputs; they correspond to $True, False, P, Q, \neg P$ and $\neg Q$. Four of them we have already studied: $\wedge, \vee, \Rightarrow, \Leftrightarrow$. The remaining six are potentially useful. One of them is reverse implication ($\Leftarrow$ instead of $\Rightarrow$), and the other five are the negations of $\wedge, \vee, \Leftrightarrow, \Rightarrow$ and $\Leftarrow$. The first three of these are sometimes called *nand*, *nor*, and *xor*.

- (Wang Zhu) 7.4 Which of the following are correct?
  - $False \models True$
  - $True \models False$
  - $(A \wedge B) \models (A \Leftrightarrow B)$
  - $A \Leftrightarrow B \models A \vee B$
  - $A \Leftrightarrow B \models \neg A \vee B$
  - $(A \wedge B) \Rightarrow C \models (A \Rightarrow B) \vee (B \Rightarrow C)$
  - $(C \vee (\neg A \wedge \neg B)) \equiv ((A \Rightarrow C) \wedge (B \Rightarrow C))$
  - $(A \vee B) \wedge (\neg C \vee \neg D \vee E) \models (A \vee B)$
  - $(A \vee B) \wedge (\neg C \vee \neg D \vee E) \models (A \vee B) \wedge (\neg D \vee E)$
  - $(A \vee B) \wedge \neg(A \Rightarrow B)$ is satisfiable.
  - $(A \Leftrightarrow B) \wedge (\neg A \vee B)$ is satisfiable.
  - $(A \Leftrightarrow B) \Leftrightarrow C$ has the same number of models as $A \Leftrightarrow B$ for a fixed set of symbols $\{A,B,C\}$.

7.4 In all cases, the question can be resolved easily by referring to the definition of entailment.

a. *False* $\models$ *True* is true because *False* has no models and hence entails every sentence AND because *True* is true in all models and hence is entailed by every sentence.

b. *True* $\models$ *False* is false.

c. $(A \wedge B) \models (A \Leftrightarrow B)$ is true because the left-hand side has exactly one model that is one of the two models of the right-hand side.

d. $A \Leftrightarrow B \models A \vee B$ is false because one of the models of $A \Leftrightarrow B$ has both A and B false, which does not satisfy $A \vee B$.

e. $A \Leftrightarrow B \models \neg A \vee B$ is true because the RHS is $A \Rightarrow B$, one of the conjuncts in the definition of $A \Leftrightarrow B$.

f. $(A \wedge B) \Rightarrow C \models (A \Rightarrow C) \vee (B \Rightarrow C)$ is true because the RHS is false only when both disjuncts are false, i.e., when A and B are true and C is false, in which case the LHS is also false. This may seem counterintuitive, and would not hold if $\Rightarrow$ is interpreted as "causes."

g. $(C \vee (\neg A \wedge \neg B)) \equiv ((A \Rightarrow C) \wedge (B \Rightarrow C))$ is true; proof by truth table enumeration, or by application of distributivity (Fig 7.11).

h. $(A \vee B) \wedge (\neg C \vee \neg D \vee E) \models (A \vee B)$ is true; removing a conjunct only allows more models.

i. $(A \vee B) \wedge (\neg C \vee \neg D \vee E) \models (A \vee B) \wedge (\neg D \vee E)$ is false; removing a disjunct allows fewer models.

j. $(A \vee B) \wedge \neg(A \Rightarrow B)$ *is* satisfiable; model has A and $\neg B$.

k. $(A \Leftrightarrow B) \wedge (\neg A \vee B)$ *is* satisfiable; RHS is entailed by LHS so models are those of $A \Leftrightarrow B$.

l. $(A \Leftrightarrow B) \Leftrightarrow C$ does have the same number of models as $(A \Leftrightarrow B)$; half the models of $(A \Leftrightarrow B)$ satisfy $(A \Leftrightarrow B) \Leftrightarrow C$, as do half the non-models, and there are the same numbers of models and non-models.

- (Will Brady) 7.15 This question considers representing satisfiability (SAT) problems as CSPs.
  - (a) Draw the constraint graph corresponding to the following SAT problem for the case when $n = 5$: $(\neg X_1 \vee X_2) \wedge \ldots \wedge (\neg X_{n-1} \vee X_n)$.
  - (b) How many solutions are there for this general SAT problem as a function of n?
  - (c) Suppose we apply Backtracking-Search to find all solutions to a SAT CSP of the type given in (a). Assume the variables are ordered, and *False* is ordered before *True.* How much time will the algorithm take to terminate? (Use O() notation).

7.15

a. The graph is simply a connected chain of 5 nodes, one per variable.

b. $n + 1$ solutions. Once any X_i is true, all subsequent X_js must be true. Hence the solutions are i falses followed by $n - i$ trues, for $i = 0, \ldots, n$.

c. The complexity is $O(n^2)$. This is somewhat tricky. Consider what part of the complete binary tree is explored by the search. The algorithm must follow all solution sequences, which themselves cover a quadratic-sized portion of the tree. Failing branches are all those trying a $false$ after the preceding variable is assigned $true$. Such conflicts are detected immediately, so they do not change the quadratic cost.

d. These facts are not obviously connected. Horn-form logical inference problems need not have tree-structured constraint graphs; the linear complexity comes from the nature of the constraint (implication) not the structure of the problem.

- (Freddy Kooliyath) 6.3 Consider the problem of constructing (not solving) crossword puzzles: fitting words into a rectangular grid. The grid, which is given as a part o the problem, specifies which squares are blank and which are shaded. A list of words is provided and the task is to fill in the blank squares by using any subset of the list. Formulate this problem precisely in two ways:
  - As a general search problem. Choose an appropriate search algorithm and specify a heuristic function. Is it better to fill in blanks one letter at a time or one word at a time?
  - As a constraint satisfaction problem. Should the variables be words or letters?
- Which formulation do you think will be better. Why?

6.3 **a.** Crossword puzzle construction can be solved many ways. One simple choice is depth-first search. Each successor fills in a word in the puzzle with one of the words in the dictionary. It is better to go one word at a time, to minimize the number of steps.

b. As a CSP, there are even more choices. You could have a variable for each box in the crossword puzzle; in this case the value of each variable is a letter, and the constraints are

that the letters must make words. This approach is feasible with a most-constraining value heuristic. Alternately, we could have each string of consecutive horizontal or vertical boxes be a single variable, and the domain of the variables be words in the dictionary of the right length. The constraints would say that two intersecting words must have the same letter in the intersecting box. Solving a problem in this formulation requires fewer steps, but the domains are larger (assuming a big dictionary) and there are fewer constraints. Both formulations are feasible.

- (Haipeng Li) 6.6 In this problem we will show how any CSP can be transformed into a CSP with only binary constraints.
  - (a) Show how a single ternary constraint such as “A + B = C” can be turned into three binary constraints by using an auxiliary variable. You may assume finite domains.
  - (b) Show how constraints with more than tree variables can be treated similarly.
  - (c) Show how unary constraints can be eliminated by altering the domains of variables.

6.6 The problem statement sets out the solution fairly completely. To express the ternary constraint on A, B and C that $A + B = C$, we first introduce a new variable, AB. If the domain of A and B is the set of numbers N, then the domain of AB is the set of pairs of numbers from N, i.e. $N \times N$. Now there are three binary constraints, one between A and AB saying that the value of A must be equal to the first element of the pair-value of AB; one between B and AB saying that the value of B must equal the second element of the value of AB; and finally one that says that the sum of the pair of numbers that is the value of AB must equal the value of C. All other ternary constraints can be handled similarly.

Now that we can reduce a ternary constraint into binary constraints, we can reduce a 4-ary constraint on variables A, B, C, D by first reducing A, B, C to binary constraints as shown above, then adding back D in a ternary constraint with AB and C, and then reducing this ternary constraint to binary by introducing CD.

By induction, we can reduce any n-ary constraint to an $(n-1)$-ary constraint. We can stop at binary, because any unary constraint can be dropped, simply by moving the effects of the constraint into the domain of the variable.