Graduate Artificial Intelligence 15-780

Homework 1: *Logic and Proofs*

Out on January 18
Due on February 1

Problem 1: Murder at the Chauvinistic Adventurer's Club

Detective d'Civet tapped his cup of coffee against his saucer, calling the room to attention. "We know this much for certain: that the killer is still at large," he announced while smoothing his mustache. There are three members of the Adventurer's Club sitting in the room: Professor Agouti, Colonel Bandicoot, and Lord Crabb. "Let us review the facts of the case," he continued:

a) Each member of the Adventurer's Club likes to sail, mountain climb, or spelunk (explore caves);
b) Each member of the Adventurer's Club likes to smoke pipes or cigars;
c) Each member of the Adventurer's Club likes to drink gin or rum;
d) No one both climbs and spelunks;
e) No mountaineer likes to drink gin, but all sailors enjoy a good rum;
f) Every spelunkers smokes cigars, while no sailor smokes pipes;
g) Agouti likes rum and is a climber;
h) Crabb only smokes the finest Cuban cigars and is a spelunker;
i) Crabb and Agouti are old enemies: Crabb dislikes everything that Agouti likes, and likes whatever Agouti dislikes;
j) Bandicoot spelunks and does either mountain climbing or sailing;
k) The murderer likes both rum and cigars
l) Exactly one of Agouti, Bandicoot, and Crabb is the murderer.

"Unfortunately," concluded the Detective with a sad sip of his coffee, "I am still unable to conclude the identity of the killer." *Can you? Answer the following questions to find out.*

1) Write each sentence out as a formula of first-order logic without equality. (*Hint: consider the predicates* $Likes(x, y)$ *and* $IsMurderer(x)$.)

2) Convert each formula to CNF.

3) Suppose you suspect that Bandicoot did it. Show this by doing the following three inferences:

   - Could Agouti be the killer? (Does $(a) \wedge \ldots \wedge (l) \models \neg IsMurderer(agouti)$?)
   - Could Crabb be the killer?
   - Is Bandicoot the killer? (Does $(a) \wedge \ldots \wedge (l) \models IsMurderer(bandicoot)$?)

   Show a resolution proof for each of the three inferences. (*Hint:* do the proofs for Agouti and Crabb first.)

Problem 2: Cryptarithms, SAT, and CSPs

A cryptarithm is a simple arithmetic problem, but the number have been replaced with letters. For example, the following is a classic example (borrowed from Wikipedia).

	S	E	N	D
+	M	O	R	E
M	O	N	E	Y

Table 1: The SEND MORE MONEY Cryptarithm.

A solution to this puzzle is O = 0, M = 1, Y = 2, E = 5, N = 6, D = 7, R = 8, and S = 9.

Consider the general problem of cryptarithms. Suppose that there are up to k string that are no more than m characters long, and a final string that is no more than $m + \lceil \log_n(k) \rceil$ characters long. The strings represent base

n numbers, so each letter could represent a number in $[0, n-1]$. We are looking for an assignment of numbers to letters so that the first k strings add up to the last string.

1. Describe a CSP that captures this problem. Try to use as few variables and constraints as possible. For this problem, please restrict yourself to simple linear constraints (constraints using $+, =, \leq, \ldots$) that are easy to explain and only reference small number of variables.
   - What are the variables? What do they represent? What are their domains?
   - What are the constraints? Explain what they do. How many of them (asymptotically) are there?
   - Explicitly write out the CSP for the SEND MORE MONEY cryptarithm (Table 1). Use $k = 2$, $m = 4$, and $n = 10$.
2. SAT solvers are extremely useful for solving problems like these. Please take your general CSP formulation that you wrote in part (1) of this question and convert it into a general SAT formula.
   - What are the variables?
   - What are the constraints? How many of them (asymptotically) are there?
3. Looking at these two formulations
   - What advantages does the CSP formulation have over the SAT formulation?
   - What advantages does the SAT formulation have over the CSP formulation?

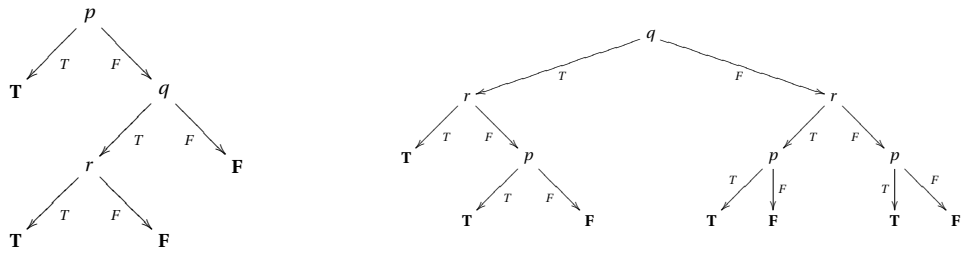
Problem 3: A Tree Representation

A decision tree is a way to represent a propositional formula. Each branch in a tree represents assigning a proposition like p both of its possible values. The leaves of the tree are either *true* or *false*, depending on whether the path to them represents a satisfying or unsatisfying assignment.

For example, consider the formula

$$p \vee (q \wedge r).$$

This formula could be represented by either decision tree:



We will restrict our attention to trees that follow a *variable ordering*. This means that every path from the root of the tree assigns the variables in the same order. For example the left subtree follows the ordering $p > q > r$ since all p-nodes appear before q nodes, and all q-nodes appear before r nodes. The right subtree follows the ordering $q > r > p$. The variable ordering is important to the size of the tree—the left tree is much more compact than the right tree.

A decision tree can either be *reduced* or not. A reduced decision tree does not have any branching nodes that are redundant in the sense that both if its subtrees are identical. For example, in both trees above, several branching nodes are omitted because they are not necessary—after assigning $p = \mathbf{T}$ the formula is satisfied regardless of the values assigned to q and r. A general decision tree may have redundant branching nodes.

1. Consider the formula $(p \rightarrow q) \wedge (r \rightarrow p)$. Draw the reduced decision tree for the ordering $p > q > r$.
2. If you were handed a formula with n propositions, and an ordering, what is the worst-case (asymptotic) size of a reduced decision tree?
3. If someone gave you a general (not necessarily reduced) decision tree for a formula that contained n propositions, how quickly could you tell if the formula is:
   - satisfiable?
   - valid?

   For each give a worst-case analysis which depends only on n, as well as a worst-case analysis which depends on the number of nodes m in the tree. What if the tree was a reduced decision tree?
4. Give a sequence of formulas $\langle F_1, F_2, \ldots \rangle$ such that F_n has n distinct propositions, along with two corresponding sequences of orderings $\langle A_1, A_2, \ldots \rangle$ and $\langle B_1, B_2, \ldots \rangle$ such that the reduced decision tree for F_n under B_n is exponentially larger than the reduced decision tree for F_n under A_n.
5. Suppose that someone told you that they had an alternative representation for a formula with n propositions that never required more than a polynomial number of bits (in n) to express. Do you believe them? Justify your answer.

Problem 4: Inductive Structures and Recursive Functions in FOL

Terms—combinations of functions and constants—can represent a number of interesting data structures in FOL. For example, we can express natural numbers with the constant $\mathbf{z}$ and the 1-ary successor function $s(x)$. *E.g.* $\mathbf{z}$ represents 0, $s(\mathbf{z})$ represents 1 and $s(s(\mathbf{z}))$ represents 2. We use the boldface font for $\mathbf{z}$ to distinguish it.

As a more advanced example, consider the 2-ary construction function $cons(x, y)$ and constants a, b, nil. We can build strings on a, b by combining constants with the *cons* function like so:

$$cons(a, cons(b, cons(b, cons(a, nil)))).$$

This example term represents the string 'abba'. Notice that first argument of *cons* is always a constant and the second argument is either a *cons* statement or *nil*. We will call such terms *well-formed strings*. An example of non-well-formed strings would be $cons(nil, a)$ or $cons(b, a)$ or $cons(cons(b, a), nil)$.

Furthermore, we can define predicates that do useful things for us:

1. Define two predicates for natural numbers:
   - *plus*, where the first two numbers must sum to form the last number *e.g.* $plus(s(\mathbf{z}), s(\mathbf{z}), s(s(\mathbf{z})))$. Feel free to use the notation $s^n(\mathbf{z}) \equiv \underbrace{s(\ldots s}_{n}(\mathbf{z})\ldots)$.
   - *times*, where the product of the first two numbers is the last number *e.g.* $times(s(\mathbf{z}), s(s(\mathbf{z})), s(s(\mathbf{z})))$.
2. In words, what does $foo(nil, \mathbf{z}) \wedge \left[\forall w, x, y,\ foo(w, x) \rightarrow foo(cons(y, w), s(x))\right]$ do?
3. In this question we will extend well-formed string to be any string where the first argument of *cons* is a natural number, *e.g.* $cons(s(\mathbf{z}), cons(s(s(\mathbf{z})), cons(s(s(\mathbf{z})), cons(s(\mathbf{z}), nil))))$ is a well-formed string of natural numbers.

   Define a predicate $sum(N, S)$ that takes in a well-formed string of natural numbers S and a number N. It must be true if and only if N is the sum of numbers in the string. For example:

   $$sum(s^6(\mathbf{z}),\ cons(s(\mathbf{z}), cons(s(s(\mathbf{z})), cons(s(s(\mathbf{z})), cons(s(\mathbf{z}), nil)))))$$

   is true since $1 + 2 + 2 + 1 = 6$.

4. How would we describe a binary tree-like structure?
5. For your above formulation of a binary tree, define two predicates:
    - The first is true if and only if d is the *depth* of the binary tree
    - The second is true if and only if n is the *size* (number of nodes) of the binary tree