Notes 15 for CS 170

1 Horn Formulae

You may have seen puzzles like the following.

> If Mr. Smith has a dog, then Mrs. Brown has a cat.
> If Mr. Jones has a dog, then he has a cat, too.
> If Mr. Smith has a dog and Mr. Jones has a cat, then Mrs. Peacock has a dog.
> If Mrs. Brown and Mr. Jones share a pet of the same species, then Mr. Smith has a cat.
> All the men have dogs.

What can we say about who has what kind of pet?

Well, we can encode this problem using Boolean variables. Let's write s, j, b, p for Boolean variables that are true if Smith, Jones, Brown, or Peacock (respectively) have a dog, and write S, J, B, P for Boolean variables that are true if they have a cat. (Note that any given person can have both a cat and a dog, or might have no pets whatsoever.) Then the above puzzle turns into the following logical formula:

$$\begin{array}{lcll}
(s & \Rightarrow & B) & \wedge \\
(j & \Rightarrow & J) & \wedge \\
((s \wedge J) & \Rightarrow & p) & \wedge \\
((b \wedge j) & \Rightarrow & S) & \wedge \\
((B \wedge J) & \Rightarrow & S) & \wedge \\
 & & (s) & \wedge \\
 & & (j). &
\end{array}$$

Which of the Boolean variables must be true? Evidently, s, j, p, S, B, J must be true, but the others can be false. This corresponds to the conclusion that Mr. Smith and Mr. Jones must own both a cat and a dog, Mrs. Peacock must own a dog (though we don't know whether she owns a cat), and Mrs. Brown must own a cat (though we don't know whether she owns a dog).

The above formula has a special property: it is a conjunction of implications, where no variable appears complemented (negated). This special kind of formula is known as a Horn formula. Each clause of a Horn formula takes the form

$$(x_1 \wedge x_2 \wedge \cdots \wedge x_k) \Rightarrow x_{k+1},$$

for some set of Boolean variables $x_1, \ldots, x_{k+1}$. A Horn formula is a conjunction of such clauses. In case it helps, we can notice that each such clause can also be written in the equivalent form

$$\overline{x_1} \vee \overline{x_2} \vee \cdots \vee \overline{x_k} \vee x_{k+1},$$

where $\overline{x}$ denotes the complement of x. Note that the case $k = 0$ is allowed, in which case the clause takes the form $\mathsf{True} \Rightarrow x_{k+1}$ (because an empty conjunction takes the value True, just as an empty sum takes the value 0), which is usually written as x_{k+1}.

We can now develop a greedy algorithm. The idea behind the algorithm is that we start with all variables set to false, and we only set a variables to true if an implication forces us to. Recall that an implication is currently unsatisfied if all variables to the left of the arrow are true and the one to the right is false. This algorithm is greedy, in the sense that it (greedily) tries to ensure the pure negative clauses are satisfied, and only changes a variable if absolutely forced.

GreedyHorn(ϕ): (where ϕ is a Horn formula)
1. Start with all variables set to false.
2. While there is an unsatisfied implication, do:
3. Set the implied variable to true.
4. Return the current truth assignment.

On the example above, GreedyHorn starts by setting s to true, due to the empty implication (s). Then, B gets forced to true, due to $s \Rightarrow B$. Then j becomes true, thanks to the other empty implication. The algorithm continues by setting J, then p, and finally S. At this point all the implications are satisfied, and we conclude that s, B, j, J, p, and S must all be true.

Note that we only set a variable to true if we are forced to by some chain of implications. Thus, any variable that is set to true in the output of GreedyHorn(ϕ) really has to be true in all satisfying assignments to ϕ. (Formally, we can prove this by induction. The induction statement would be the following: after GreedyHorn has flipped the value of k variables to true, any satisfying truth assignment must also have those k variable set to true.) Likewise, if v is true in all satisfying assignments to ϕ, then it must be because there is a chain of implications which will cause GreedyHorn(ϕ) to set v to true: if there is no implication forcing v to be true, then we could take any satisfying assignment and change v to false, and the resulting assignment would also satisfying ϕ.

How efficient is this algorithm? If the formula ϕ has length n, then GreedyHorn(ϕ) might require $O(n)$ iterations of the while loop, and each iteration of the while loop might take up to $O(n)$ work to scan for unsatisfied implications. Thus, GreedyHorn(ϕ) runs in $O(n^2)$ time: it is a quadratic-time algorithm.

It turns out that if we are slightly clever in how we implement the above algorithm, we can make it run in linear time. We will assume ϕ has been parsed into a conjunction of clauses, and each vertex v will have a link to all the clauses where it appears on the left-hand side of the implication. See Figure 1 for the algorithm.

Let's analyze the running time of this algorithm. If we store the left-hand side of each clause as a doubly linked list of variables, then Step 7 can be performed in $O(1)$ time. The set W will hold a worklist, and it is easy to make all operations on W run in $O(1)$ time. The running time analysis is now similar to that for depth-first or breadth-first search. Each variable enters the worklist at most once, and each iteration of the while loop removes one element from the worklist. Consequently, the number of iterations of the while loop is at most the number of variables, and so the total number of executions of Step 5 is at most $O(n)$. Each clause of the form $(v_1 \wedge \cdots \wedge v_k) \Rightarrow v_{k+1}$ is examined at most k times in Steps

FASTGREEDYHORN(ϕ):
1. Set v to False for each variable v in ϕ.
2. Set $W := \{v : v$ appears on the right-hand side of an empty implication$\}$.
3. While $W \neq \emptyset$, do:
4. Take (and delete) v from W.
5. Set v to True.
6. For each clause c where v appears on the left-hand side, do:
7. Delete v from the left-hand side of c.
8. If this makes c into an empty implication, add the variable on the right-hand side of c into W (if it is not already in W).
9. Return the current truth assignment.

Figure 1: A linear-time algorithm for Horn formulae.

7–8, hence the total number of executions of Steps 7–8 is at most $O(n)$. Adding up all the costs, we see that the total running time for FASTGREEDYHORN(ϕ) is $O(n)$, where n denotes the length of ϕ.