CSE 547T Automata Theory April 13, 2017

Solutions for Homework 4

1. Let M be a TM deciding L in time $t(n)$. We use the following scheme to compress M's input and speed up its computation by a factor of c, producing a new TM M^*.

   For any $d > 1$, we may define a new set of symbols $\Gamma' = \Gamma^d$. Each symbol of Γ' corresponds to a string of d symbols from Γ. Given an input string w, M^* compresses w by first making a single pass over the input, reading each successive d symbols and writing the corresponding symbol from Γ' in place of the last one, until it has moved past the end of input. (The last symbol may include up to $d-1$ trailing blanks.) Then, the TM compacts the symbols from Γ' into a contiguous string at the beginning of its tape and replaces all other non-blank cells by blanks. For an input of size n, this compression takes $O(n)$ time to compute and write the symbols from Γ' in place, plus $O(n^2)$ time to read each compressed symbol and move it back to its spot at the beginning of the tape. If $t(n) = \omega(n^2)$, then all this work takes time $o(t(n))$.

   We now describe how M^* efficiently simulates M's computation on the compressed string, assuming a compression ratio of d. M^* tracks not only its own head position, which indicates which compressed symbol $A = \langle a_1 \dots a_d \rangle$ it is over, but also the "virtual" head position of M within that symbol, corresponding to one of the d cells comprising A. The virtual head position has only d possibilities and so can be maintained in M^*'s finite control. Initially, this virtual position is 1, since M's head starts at the left end of the first compressed cell.

   At the beginning of a move of M^*, its head is over some compressed symbol A. M^* first moves left, reads the symbol A_ℓ to the left of A, then moves right two cells, reads the symbol A_r to the right of A, and finally moves back to A.

   At this point, M^* knows window (A_ℓ, A, A_r) corresponding to $3d$ cells of M's tape, and it knows M's head position in this window, which is inside A. Observe that after the next d moves of M, *its head must still be within this window*, since it cannot move past either A_ℓ or A_r in this many moves starting from within A. Moreover, we can precompute what the configuration of M will be after d moves given its starting configuration. This precomputation can be described by a function δ_c with a generic entry like

   $$\delta_c(q, A_\ell, A, A_r, j) = (p, B_\ell, B, B_r, k),$$

   meaning "If M starts in state q at the jth position of window (A_ℓ, A, A_r), then after d moves it will be in state p at position k within the window, and the window's contents will be (B_ℓ, B, B_r)." The function δ_c is part of M^*'s finite control.

   To complete d moves of M, M^* computes δ_c for M's current configuration as above, then overwrites the current window with (B_ℓ, B, B_r) in such a way that its head ends over the compressed symbol containing the new head position of M. This can always be done in at most four moves. If M halted during the last d moves, M^* halts as well.

   We conclude that M^* can simulate d moves of M in at most 8 of its own moves. If $T(n)$ is the running time of M on an input of size n, then the running time of M^* is given by

   $$\begin{aligned} T^*(n) &= 8\lceil T(n)/d \rceil + o(n^2) \\ &\leq 8T(n)/d + o(n^2) \end{aligned}$$

(where the extra constant added by removing the ceiling has been absorbed into the lower-order term). If we set $d \geq 16c$, then we have that

$$T^*(n) \leq 1/2T(n)/c + o(n^2),$$

which means that for some n_0 and all $n > n_0$,

$$T^*(n) \leq T(n)/c.$$

For the finitely many inputs of size $n \leq n_0$, M^* can remember M's result for each such input in its finite control and so can dispose of such inputs in n moves.

2. Let ϕ be a Boolean formula. The following exponential-time strategy suffices to determine whether ϕ is satisfiable. Order the variables of ϕ as $x_1 \ldots x_n$. Let eval(ψ, $x_i = v$) be a function that partially evaluates formula ψ by setting x_i to the truth value v and simplifying. (Clearly, eval runs in time polynomial in $|\psi|$). Then we can successively generate all 2^n partial evaluations of ϕ as follows:

```
$S_0 = \{\phi\}$
**for** $i$ from 1 to $n$ **do**
    $S_i = \emptyset$
    **for all** $\psi \in S_{i-1}$ **do**
        $\psi_0 = $ eval($\psi$, $x_i = 0$)
        $\psi_1 = $ eval($\psi$, $x_i = 1$)
        $S_i = S_i \cup \{\psi_0, \psi_1\}$
**return** true iff $S_n$ contains "true"
```

We now augment this strategy using A to obtain a polynomial-time algorithm for SAT. Let f be a polynomial-time transducer mapping instances of SAT to instances of A. Crucially, we know that ϕ is satisfiable iff $f(\phi)$ is a true instance of A. Moreover, because f runs in time polynomial in $|\phi|$, it produces a string of zeros of length poly($|\phi|$). Hence, *there are only poly($|\phi|$) distinct values $f(\psi)$ among all formulas ψ such that $|\psi| \leq |\phi|$.* This includes all formulas that arise as partial evaluations of ϕ.

We modify our enumeration procedure above to merge any two formulas $\psi, \psi' \in S_i$ such that $f(\psi) = f(\psi')$. We know that one such formula is satisfiable iff the other is. Here's the modified procedure:

```
$S_0 = \{\phi\}$
**for** $i$ from 1 to $n$ **do**
    $S_i = \emptyset$
    **for all** $\psi \in S_{i-1}$ **do**
        $\psi_0 = $ eval($\psi$, $x_i = 0$)
        $\psi_1 = $ eval($\psi$, $x_i = 1$)
        $S_i = S_i \cup \{\psi_0, \psi_1\}$
    **for all** $\psi, \psi' \in S_i$ **do**
        **if** $f(\psi) = f(\psi')$
            $S_i = S_i - \{\psi'\}$
**return** true iff $S_n$ contains "true"
```

We first observe that the revised procedure is still correct. Indeed, ϕ is satisfiable iff, at each stage of the original procedure, at least one formula $\psi \in S_i$ is satisfiable, since partial evaluation cannot change satisfiability of a formula. For the revised procedure, we never merge a satisfiable with an unsatisfiable formula, so the above observation still holds. Conclude that ϕ is satisfiable iff S_n contains the only satisfiable formula on zero variables, namely "true".

We now argue that the revised procedure runs in time polynomial in $|\phi|$. At the end of each outer loop iteration, S_i contains at most as many formulas as there are distinct values of f on formulas of size $\leq |\phi|$. But there are only poly($|\phi|$) such values, so S_i has polynomial size. Inside iteration i, we generate up to double the number of formulas in S_{i-1} and then compare them all pairwise using f to merge formulas that map to an equivalent value. Hence, we do only $O(|S_{i-1}|)$ partial evals and $O(|S_{i-1}|^2)$ merges, each in time polynomial in $|\phi|$. Conclude that the entire procedure runs in time $O(\text{poly}(|\phi|))$.

Since we have solved SAT in polynomial time, the existence of A implies that $P = NP$.

3. It is straightforward to show that LPFP is in *NP*. A certificate is a path through G (i.e. an ordered list of vertices). The verifier checks that the path is indeed present in G, that it has length at least k, and that none of the forbidden pairs are present on the path.

   To show that LPFP is NP-hard, we reduce from 3SAT. Let ϕ be a 3-CNF formula with m clauses. We construct a graph G with a source s, and sink t, and m "levels," each containing three vertices. The vertices on level j correspond to the three literals in clause C_j; we label them ℓ_j^1, ℓ_j^2, and ℓ_j^3. We add edges from s to each vertex at level 1; from each vertex at level m to t, and from each vertex at level j to every vertex at level $j+1$. Finally, we add as forbidden pairs any two vertices ℓ, ℓ' that are logical negations of each other. This construction can be done in time at most quadratic in the formula size.

   Claim: ϕ is satisfiable iff there exists a path of length $m+2$ in G with no forbidden pairs.

   Pf: Suppose ϕ has a satisfying assignment A. Let ℓ_j^* be a literal in clause C_j that is made true by A. Consider the path p that starts at s, goes through each ℓ_j^* for $1 \leq j \leq m$, and ends at t. This path has length $m+2$. Moreover, since every literal corresponding to an intermediate vertex on the path is simultaneously true, no literal is the logical negation of any other literal on the path, and so the path is free of forbidden pairs.

   Conversely, suppose that the desired path p exists in G. Because p has length at least $m+2$, it must start at s, end at t, and touch every level of G once. Let ℓ_j be the literal at level j on path p. We know that p does not contain any forbidden pairs, so no literal and its negation are both present in p. Consider a truth assignment A for ϕ that is chosen so as to make each ℓ_j on p true. A is feasible because of the lack of forbidden pairs on p, and it makes one literal of every clause true and so satisfies ϕ.

4. Recall that the logical implication $x \implies y$ is equivalent to the assertion $\neg x \vee y$. Hence, a 2-clause of the form $\ell \vee \ell'$ is equivalent to the logical implication $\neg \ell \to \ell'$, and to its contrapositive $\ell' \to \neg \ell$.

   We can represent ϕ as a directed graph G on the literals $x_1 \ldots x_n$ and $\neg x_1 \ldots \neg x_n$ of ϕ. If ϕ contains the clause $\ell \vee \ell'$, then G contains edges $\ell \to \neg \ell'$ and $\ell' \to \neg \ell$. Note that a path from ℓ to ℓ' exists in G iff ℓ logically implies ℓ' given ϕ.

   Claim: ϕ is satisfiable iff G does not contain paths from any vertex ℓ to $\neg \ell$ and from $\neg \ell$ to ℓ.

   Pf: Suppose that G contains the two paths above. Consider any truth assignment that makes ℓ true (and hence makes $\neg \ell$ false). Then there must be some edge $a \to b$ in the path from ℓ to $\neg \ell$ whose tail is true and whose head is false. But this edge corresponds to a clause $\neg a \vee b$ in ϕ, which is falsified by the truth assignment, and so the assignment does not satisfy ϕ.

Consider instead an assignment that makes ℓ false. The assignment also makes $\neg\ell$ true, so we can make the same argument as above to show that it does not satisfy ϕ. Conclude that no truth assignment can satisfy ϕ.

Suppose conversely that ϕ does not contain paths from ℓ to $\neg\ell$ and $\neg\ell$ to ℓ. Then we construct a satisfying assignment for ϕ as follows.

(a) First, for every vertex ℓ that has a path to its logical inverse, set its variable so as to make ℓ false.

(b) For all directed paths into literals that were just set "false", propagate the "false" assignment backward to all literals along each path until a previously set literal is found. Set the inverse of each newly set literal vertex to "true".

(c) If no new literals had their truth values set "true" in the previous step, pick an arbitrary vertex with no truth assignment and set its variable so that its literal is true. If no such vertex exists, stop.

(d) For all directed paths out of literals that were just set "true", propagate the "true" assignment forward to all literals along each path until a previously set literal is found. Set the inverse of each newly set literal vertex to "false".

(e) Go to step 2.

We claim that the above process never yields an inconsistent assignment, i.e. one in which the head of some edge is true and the tail is false.

Consider the first time step 2 is executed (i.e. right after step 1). No vertices were previously set, so this step could fail (i.e. create an edge from "true" to "false") only if

- some literal ℓ was set "false" because it has a path to $\neg\ell$;
- there exist paths to ℓ from some literal m and from $\neg m$.

But then there must also be paths of contrapositive implications from $\neg ell$ to $\neg m$ and to m, and these literals have paths to ℓ, contradicting our intial assumption.

Now consider any later execution of step 2 or 4. We describe step 4; the argument for step 2 is similar. If an execution of step 4 creates an edge from "true" to "false", then setting some literal ℓ "true" and propagating this value led to a vertex m that had already been set "false". m cannot have been set in a previous stage of execution, or its value would have been propagated back to ℓ. Hence, the current stage also propagated "true" to literal $\neg m$. But then there are paths from ℓ to m and $\neg m$ and hence contrapositive paths from m to $\neg\ell$ and $\neg m$ to $\neg\ell$. Conclude that ℓ would have been set false in the first execution of step 2, so this bad case is impossible.

Conclude that all implications, and hence all clauses of ϕ, can simultaneously be satisfied.