CSE 531, HW #8 notes

Dec, 2000

1 Problem 1

1.1 2^{n^k} is not polynomial in 2^n, suprisingly

1.1.1 Proof

Suppose to the contrary that it was. There there would be some constant c s.t. $(2^n)^c \in \Theta(2^{n^k})$ (recall that $f(x) \in \Theta(g(x)) \Leftrightarrow f(x) \in O(g(x))$ and $g(x) \in O(f(x))$. So intuitively Θ means equivalent with respect to big-O concepts.). Therefore, we'd have $2^{n^k} \in O((2^n)^c)$.

But, $(2^n)^c = 2^{cn}$. So, for example, if $k = 2$, $2^{n^2} \notin O(2^{cn})$ for any c (which you can prove using the fact that $f(x) \notin O(g(x))$ iff $\lim_{x\to\infty} \frac{g(x)}{f(x)} = \infty$ and since 2^n is a super-linear function).

1.1.2 Your faithful TA made a similar mistake

On some of your hws I think I may have said something like "No! Counter-intuitively, 2^{n^k} is still exponential in 2^n". If I did (and I think I did), it would seem that I stepped over the banana peel of saying that they're polynomially related – right into the open manhole of saying it's exponential. In other words, I goofed too.

If 2^{n^k} were exponential in 2^n, then we'd have $c^{2^n} \in O(2^{n^k})$ for some c. This is false. Intuitively, c^{2^n} is doubly-exponential, and here our intuition is correct[1] (the proof having the same idea as the one that they're not poly-related).

1.1.3 Speaking of misperceptions, there's functions between poly and exponential

Last year in CSE 521 I learned that there are functions that are between polynomial and exponential (okay, I know this will be late-breaking news for many of you). Examples are $n^{\log n}$, $(\log n)^{\log n}$ and $n^{\sqrt{n}}$.

[1] I hope

1.1.4 How to avoid this kind of mistake in the future.

I have three suggestions:

- Be more sceptical. Be suspicious of intuitive arguments like 2^{n^k} is poly in 2^n since they're both exponential. Prove it.
- Side-step tricky proofs where possible. If you have created the padded language like so: $A' = \{w\#^{2^{n^k}} \mid w \in A\}$, clearly 2^{n^k} (well $2^{n^k} + n$) is poly in 2^{n^k} – in fact it's linear. No need to bother with a tricky proof (of something that's actually false!), and no need to do some kind of "minimal" padding. Just spew out the extra #s and have an easier proof.
- Be perfect. If you were perfect, you would avoid these kinds of mistakes. Okay, okay, so maybe this advice is somewhat impractical.

1.2 Checking the #s is not linear

Checking the #s is not, I think, linear in the size of the input, at least not with a 1-tape Turing machine. It is polynomial, however. Very minor point, of course.

1.3 Assumptions

There seemed to be a tendency in solutions to be vague about what we're assuming (for the purposes of obtaining a contradiction). Stylistically, I think it makes sense to assume EXPTIME≠NEXPTIME and P=NP right at the start of the proof. Then show the contradiction with these two assumptions, and conclude that the implication must be false.

2 Problem 2

I have many things to say about this.

2.1 Style recommendation in describing alternating algs

For describing alternating algs: after reading through everyone's solutions, I think it's most natural to describe alternating algs using regular pseudocode plus "universally select" and "existentially select" (or universally/existentially do). Drawing trees is sometimes helpful to explain what the algorithm does, but doesn't seem to be that good at explaining the algorithm in the first place.

So, for example:

```
1: Existentially select an intermediate state, $M$
2: Unversally do either CANYIELD($M_0, M, t/2$) or CANYIELD($M, M_1, t/2$).
```

(note: this algorithm has many flaws; it's just an example of the style)

As an aside, observe that "Unversally do (one of two choices)" is the same as "Universally select a boolean value, if it's 1 do X, if it's 0 do Y".

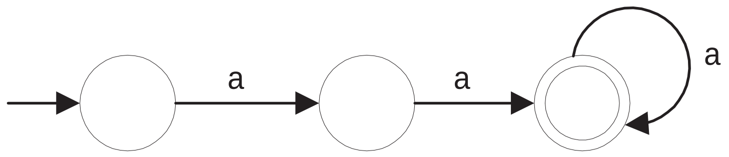

Figure 1: An NFA that requires checking for strings shorter than $2^{|Q|}$

This NFA runs over the alphabet $\Sigma = \{a\}$ (I didn't label the edges with Σ since Visio has problems with Greek letters). Clearly it will accept all strings over $\{a\}$ of length $2^{|Q|} = 2^3 = 8$, but won't accept strings of length 0 or 1. So, it's insufficient to only check strings of length $2^{|Q|}$.

2.2 You have to test strings of all lengths up to 2^k

I think you all know this, but it's not sufficient to just test all strings of length *exactly* 2^k (or $2^k - 1$). You also have to test strings shorter, right down to the empty string. Many (in fact, I think most) solutions didn't work in this regard.

Figure 1 (on page 3) shows an example NFA that requires this.

I think there's a couple of ways around this:

- Universally select an integer t s.t. $0 \leq t \leq 2^{|Q|}$. Then do the check for strings of this length. t has $|Q|$ bits, so you can write it down in linear time.
- Fudge your CANYIELD function s.t. it will return true if the start (set of) state is equal to the target state, regardless of how many steps you're supposed to do it in. There are other variations on this idea. In this case you can see if it rejects early (i.e. for shorter strings).

Note: adjusting the NFA s.t. each accept state has a Σ edge to itself (i.e. once it's in an accept state it's stuck there) doesn't work for this problem. This trick, however, does work for solving not-empty, where we only need to see one acceptance.

2.3 You can't just reverse the answer of an alternating machine

True confession: I fell into this trap last Friday, and had to think for quite a pregnant pause after Richard told me you couldn't do this.

Why you can't trivially do it. If I give you an alternating TM, you can't just say something like "Run Zasha's alternating TM, and reverse its answer". This is a problem for the same reason that you can't do this with a non-deterministic TM, and the reason that it's not trivial that co-NP=NP[2].

[2]For those of you who thought I e-mailed that thing about co-NP vs. NP because I wanted you to realize the same thing about alternation... well done! In fact, I was just intending to ask about co-NP, and a student really did ask me about that. I hadn't realized about the alternation issue at all. So, good job for generalizing my advice.

The idea is the same; the results of the alternating choices are at the leaves. Once the leaves are known, that's it. If you try to reverse the answer, you're just doing it at all of the leaves. But, unfortunately, true or false is the same as false or true, so reversing the leaves doesn't reverse the answer.

Also, there's no point at which you "return from a subroutine that did an alternating computation, and can modify the result", as one student in the class thought[3]. Each alternating branch of computation only sees one sequence of universal/existential choices.

But there's a straightforward way of achieving the same effect. However, things *are* better in the alternation world. In co-NP, it's possible that you could reverse the answer quickly enough if co-NP=NP, but no-one knows how to, and I think most people think it's impossible.

However, in alternation there is actually a fairly simple way of reversing the answer. It's based on the following facts in logic, given a boolean predicate $p(x)$:

- $\neg(\forall x(p(x))) \iff \exists x(\neg p(x))$. In words, if it's not true that all x satisfy $p(x)$, then there must be some x that doesn't satisfy it.

- $\neg(\exists x(p(x))) \iff \forall x(\neg p(x))$. In words, if it's not true that there's some x that satisfies $p(x)$, then it must be that all x don't satisfy it.

Notice the symmetry between these properties. This leads to the following simple algorithm to reverse the answer of an alternating TM:

- Every time the TM universally selects something, change it to an existential selection. And vice-versa.

- Every time the TM rejects, instead accept. And vice-versa.

So, you have to be careful about just reversing an alternating computation, but given an algorithm you can fairly easily make another algorithm that does the opposite of the first algorithm. And you can do this without changing your time or space bounds.

2.4 Can't write down a string of length 2^k in poly time

The following is not a valid part of a poly-time alternating algorithm:

```
1: Universally select a string of length $2^{|Q|}$.
```

You can't write down this string in poly-time. There's no magic in alternation that allows you to just do this in APTIME (although there is magic that allows you to use other Savitch-like tricks to do it in APTIME).

[3]This is true; technically I'm a student in the class.

2.5 An alternate, alternating alg for this problem

My initial solution for this problem was to string a bunch of theorems together and hammer out an algorithm. Richard wasn't very happy with this solution, since it sidesteps most of the interesting issues. So, I'm pleased that no-one suggested it; I think you would have been missing out on important concepts.

However, I think it's worth sharing, and seems to have helped at least one person to understand alternation a bit better.

2.5.1 Sketch of the alg.

As I said above, we're basically going to string together a bunch of proofs. It will be fortunate that all of these proofs specify an actual algorithm (the problem was to *design* an algorithm, not merely to prove that one exists).

The first theorem is one we proved in class: $\mathrm{NE}_{\mathrm{NFA}} \in \mathrm{PSPACE}$. In fact we saw an algorithm for this, which we proved to be in PSPACE (technically, we saw an NPSPACE algorithm; however, we can use the algorithm from Savitch's theorem to convert the NPSPACE algorithm to PSPACE. We do this by applying the Savitch's Thm algorithm, but leaving the input up for grabs. The amount of detail I gave here is probably insufficient for full credit, but I'm going to get a 0.0 in this class anyway, because I plagiarized your answers for several homeworks. At any rate, you can see already why Richard didn't like this...). So, now we have a PSPACE alg for $\mathrm{NE}_{\mathrm{NFA}}$.

The next theorem is in the book, and shows that any problem in PSPACE is poly-time mapping reducible to TQBF (pp. 283-). Once again, it shows the actual algorithm that maps it in poly time. So, now we have a TQBF problem that's equivalent to the original $\mathrm{NE}_{\mathrm{NFA}}$ problem.

However, we really want to solve $\mathrm{E}_{\mathrm{NFA}}$, where $\mathrm{E}_{\mathrm{NFA}} = \overline{\mathrm{NE}_{\mathrm{NFA}}}$. Well, in section 2.3 (page 3) we saw a way to negate QBF formulas (although in that context it was a way to reverse the answer of an alternating alg).

Now we have a QBF with a boolean predicate. At this point, it will be easy to use alternation to solve this in APTIME.

2.5.2 The alg

The algorithm is:

1: Have a hardcoded copy of the PSPACE TM that decides $\mathrm{NE}_{\mathrm{NFA}}$, as described in lecture. Call this machine M_{NE}.
2: Given $\langle M_{\mathrm{NE}}, N\rangle$, where N is the NFA in question, run the poly-time algorithm from the book to create an instance of QBF. Note that the resulting QBF will be poly-length, since otherwise it couldn't have been written by a poly-time alg.
3: In the QBF, swap $\forall$ and $\exists$, and put a not sign before the predicate (this negates the QBF, so it's now a QBF for $\mathrm{E}_{\mathrm{NFA}}$, as desired). At this point, we'll have a problem like

$\forall x_1 \exists x_2 \ldots \exists x_n \neg(p(x_1, x_2, \ldots, x_n))$ where $p(x_1, x_2, \ldots, x_n)$ was a 3-cnf formula. Note that, as per the definition of QBF, the x_i are all boolean variables.

4: Run through the QBF. Whenever you see something like $\forall x_i$, universally select a value for x_i. Similarly, whenever you see something like $\exists x_i$, existentially select a value for x_i.

5: Eventually you'll get to the actual predicate, $\neg p(x_1, x_2, \ldots, x_n)$. At this point, simply substitute the values of the variables you universally/existentially selected into the formula for $\neg p$, and test it.

2.5.3 Running time

Step 1 is constant time, since we're just hardcoding the machine we learned about in lecture. Step 2 is (deterministic) poly time, which the book proved. Step 3 is clearly poly-time (in fact it's just a single sweep through the QBF problem). Step 4 is alternating poly time, since you're making one alternating decision for each variable in a QBF of poly size. Step 5 is poly-time, since you're just checking a poly-size boolean formula.