What We’re Covering Today

About a week ago, Darius and BJ and I were talking, and Darius said something along the lines of “Is it just me, or do you get the feeling that by the end of the semester, we’ll have this P vs NP thing solved? I mean, it really can’t be all *that* hard.”

So that’s what the next couple of lectures are about: Trying to prove something that doesn’t seem all that hard. Today we’re going to examine the technique we’ve been using to separate classes. On Friday, assuming I get my act together, we’ll just apply the technique to P vs. NP and be done with it all. OK, OK, so we’re going to *fail* on Friday (even if I do get my act together!), but it won’t be without a valient effort.

Consider this ubiquitous mathematical problem:

Goal: to show a containment is strict

For some set of things $A$ and a specific subset $S$, there are some things in $A$ that aren’t in $S$ (draw Venn Diagram here).

This problem is absurdly simple when the sets $A$ and $S$ are explicitly given to us. Unfortunately, when the sets are abstract and implicitly defined, this can be difficult. I can only think of one, well maybe two, times when we’ve achieved it this semester. I’m referring to when we showed

- The set of decidable languages is a strict subset of all languages (thus proving there are languages which are not decidable).
- Any enumerable set of numbers is a strict subset of the real numbers (thus proving the real numbers are not enumerable).

Both of these proofs had the same form: (write this on board)

Method:

1. Assume that $A = S$

2. Construct an element that’s in $A$, but differs from the $i$’th thing in $S$ in “the $i$th way.”
3. Therefore, since the element is in $A$ but not in $S$, conclude $A = S$ is a contradiction.

**Uncountability of the Reals**

Goal: show any enumerable set of numbers is a strict subset of the real numbers.

Method: Assume the reals are enumerable. Construct a real number that differs from the $i$th number in the enumeration in the $i$'th digit.

.9 4 2 5 7 7 . . .
.4 2 4 6 3 2 . . .
.4 5 2 6 6 7 . . .
.3 3 3 1 3 5 . . .
     . . . . . .
     . . . . . .

**Undecidability**

Goal: show the set of decidable languages is a strict subset of all languages.

Method: Assume that all languages are decidable. Construct a language that differs from the language decided by machine $I$ on input $⟨I⟩$.

$T =$ “On input $w$

1. Check that $w$ is a turing machine description $⟨I⟩$.
2. Simulate $I$ on $w$.
3. Output the opposite of what that machine outputs. If it accepts, reject. If it rejects, accept.

This Turing machine obviously differs from any decidable machine on at least one input. Thus, it can’t itself be decidable. (Note that this is a sort of condensed proof of the one we gave earlier to show that $A_{TM}$ was undecidable. This is because here, we’re just showing the existence of undecidable languages).

**The Space Hierarchy Theorem**

Goal: The set of languages solvable in space $o(f(n))$ is a strict subset of those solvable in $O(f(n))$

Method:

1. Assume that everything that can be solved in $O(f(n))$ space can be solved in $o(f(n))$ space.
2. Construct an algorithm \( T \) that runs in \( O(f(n)) \) space, but differs from any machine \( I \) that runs in \( o(f(n)) \) space by outputting the opposite of what \( I \) outputs on input \( \langle I \rangle \).

Unfortunately there are a few technical details (aren’t there always?), but the idea is simple. Let’s go with it (I’m going to present the solution in a backwards order from Sipser. I’ll first present the algorithm, with lots of holes, and we’ll patch them up together. I don’t know if this is a wise order of presentation, but you can always read Sipser to see it the other way).

\( T = \text{“On input } w: \)

1. Compute \( n \), the length of \( w \) (in other words, \(|w|\)).

2. Compute \( f(n) \), and mark off this much tape. In future steps, reject if we try to use more than is marked off.

3. If \( w \) is not of the form \( \langle I \rangle \) for some TM \( I \), reject. (later, we’ll change this to \( \langle I \rangle^{10^*} \))

4. Simulate \( I \) on \( w \) (keeping track of the number of steps used in the simulation. If the simulation ever takes more than \( 2^{f(n)} \) steps, reject).

5. Finally, do the opposite of what \( I \) does. If \( I \) accepts, reject. If \( I \) rejects, accept.

Now remember what we set out to do: “give an algorithm \( T \) that, while still running in \( O(f(n)) \) space, differs from any machine \( I \) that runs in \( o(f(n)) \) space by outputting the opposite of what \( I \) outputs on input \( \langle I \rangle \).” Have we done this?

First, we missed something very important. Can you spot it? It’s closely related to the email I sent a day or two ago, and the proof BJ posted. This machine isn’t even a decider yet! It’s easy to fix though (ammend step 4).

Now let’s make sure that this machine runs in \( O(f(n)) \) time. How long does step 1 take? Well, just computing the length of \( w \) takes \( \ln n \) space, so right off the bat we’re going to have to change the statement of the proof to stipulate that \( f(n) \geq \ln n \) (change proof statement on board).

Step 2 is the evaluation of the function, \( f(|w|) \). It’s not entirely clear how much space it takes to compute this function, but we’ll skirt the issue by further ammending the statement of the proof to say that \( f(n) \) is computable in \( f(n) \) space. BTW, these last two things I brought up \( f(n) \geq \ln n \) and \( f(n) \) computable in \( O(f(n)) \) space, are what make up the definition of \textit{space constructibility}. (change proof statement on board). Most normal functions greater than or equal to log are space constructible. I’ll come back to this in a second though.

Step 3 runs in negligible space (definitely no more than \( O(\ln n) \), though the particulars depend on the representation of the TM).

The counter used in Step 4 uses no more than \( \ln 2^{f(n)} = f(n) \) space. We have to be a little careful with the simulation itself, since we’re simulating a machine that may have an arbitrary alphabet with one that has a fixed alphabet. But this only introduces a constant space factor.
OK, so we see that as long as \( f(n) \) is space constructible, this algorithm does indeed run in \( O(f(n)) \) space. Now we need to verify that this algorithm is different from any algorithm running in less than \( O(n) \) space. Just as we did with the proof that \( A_{TM} \) is undecidable, we'll assume that there is such a an equivalent algorithm, decided by a machine \( S \), which runs in \( g(n) \) which is \( o(f(n)) \).

Clearly this algorithm differs from \( T \) on input \( \langle S \rangle \), right? Well, it gets past the first step since it runs in less than \( f(n) \) space, and then the next two steps, and then the fourth step since it’s a decider. Then it gets to the fifth step and we get the contradiction: If \( S \) accepts \( \langle S \rangle \), then \( T \) rejects \( \langle S \rangle \). If \( S \) rejects \( \langle S \rangle \), then \( T \) accepts.

However, we made one goof. Can you spot it? It’s fairly subtle. The goof is this: remember when I said “Well, it gets past the first step since it runs in less than \( f(n) \) space...” That’s not true. It runs in \( o(f(n)) \) space, but that behavior may not kick in until \( n \) reaches some specified \( n_0 \). What if \( \langle S \rangle \) is less than \( n_0 \)? Then step 2 might prematurely reject, and we won’t necessarily have a contradiction.

So we have to fix up the proof. It’s not too difficult though. Instead of making the contradictory input \( \langle S \rangle \), we make it \( \langle S \rangle^{10^{n_0}} \) (I’m guessing sipser put the “1” in front just to distinguish this string from the description string). Now if we change step three above to check if \( w \) is of the form \( \langle I \rangle^{10^{n_0}} \), we’ll definitely output the opposite of \( S \) on the same input.

QED

**Consequences**

Now talk about space constructability. It’s a very strange definition, seemingly made just for purposes of this particular proof. It does the job though, because we get the following cool corrolaries.

The first consequence is Corrolary 9.4: this just says that if we have a space constructable function, and a smaller function that’s “little-o” of the first, there are strictly more languages solvable with the larger amount of resources than with the smaller amount.

The next consequence is \( SPACE(n^{c_1}) \) is strictly contained in \( SPACE(n^{c_2}) \) for natural numbers \( c_1 < c_2 \).

This should be pretty obvious for space constructable functions. It extends to rationals, but this is *not* obvious, I just didn’t have time to try to prove it myself. If you accept it though, you get corrolary 9.5.

Finally, correloary 9.6, that NL is contained in PSPACE also becomes a cinch.

**The Time Hierarchy Theorem**

Write it out just like the space heirarchy theorem. Why can’t we do it this way? Well, look at step 4. while we can simulate a machine that runs in \( f(n) \) space
with only $cf(n)$ space (where $c$ is dependent upon the ratio of tape symbols on the machine we’re simulating to the tape symbols on the machine that’s doing the simulation), we don’t know how to simulate an $O(t(n))$ time machine, and simultaneously keep a count of the number of steps we’ve used without introducing a logarithmic factor overhead (note that this complication arises because we’re using a single tape TM. With a multitape machine, we can do better.) In fact, it takes a bit of ingenuity just to get the thing running this well. Let’s see how:

Now, we include in our premise that $t(n) \geq n \log n$ and $t(n)$ is computable in time $O(t(n))$

**Consequences**
Corrolary 9.11
Corrolary 9.12
Corrolary 9.13: P is strictly contained in EXPTIME

**P vs NP**

So we’ve seen 4 different incantations of this diagonalization method (enumerables different from reals, decidables different from recognizables, Space($O(f(n))$) different from Space($o(f(n))$), Time($O(t(n))$) different from Time($o(f(n) / \log t(n))$).

Goal: show that P is a strict subset of NP
Method:
1. Assume P=NP.
2. Construct a machine that runs in NP, but differs from any P machine $I$ by outputing the opposite of what $I$ outputs on some input.

The homework for next time: Finish this proof. This homework isn’t to hand in. Don’t worry if you don’t get it :-}