CSE547T Class 17

Jeremy Buhler

March 27, 2017

1 Intro to Complexity

- Up until now, we cared only whether a computation could be done at all.
- We're now going to switch gears and consider how *fast* a computation can be done.
- The answer will be a lot more sensitive to the details of our computational model.
- To get started, let's lay the ground with some definitions.

How do we measure the complexity (in space and time) of a computation?

- Let M be a (deterministic) TM that decides language L. The *running time* of M on input w is the number of moves taken by M before it accepts or rejects M.
- The running time $t(n)$ of M on inputs of size n is its maximum running time on any input of size n.
- The *space* used M on input w is the number of tape cells ever read by M before it accepts or rejects M.
- The space $s(n)$ used by M on inputs of size n is its maximum space usage on any input of size n.

Note that, as you have probably seen before, we measure worst-case time and space usage as a function of input size n.

- As usual, we care not about performance for any particular n but the asymptotic behavior as n becomes large.
- In particular, we'd like to show that $t(n) = O(f(n))$ for some well-behaved function $f(n)$; that is, for large enough n, $t(n) \leq cf(n)$.
- Or, we might want to show that $t(n)$ grows strictly slower than $f(n)$ in an asymptotic sense, i.e. $t(n) = o(f(n))$, meaning

$$\lim_{n \to \infty} \frac{t(n)}{f(n)} = 0$$

- (Pause to make sure everyone also knows Ω, ω, Θ.)

Every TM has an associated worst-case complexity $t(n)$. We can ask "which languages have a fast TM to decide them?"

- The time complexity of a language L is the time complexity of the fastest TM (in the worst-case asymptotic sense) that decides L.
- For example, every regular language has time compexity $O(n)$ (because a TM can simulate a DFA in time linear in its input size).
- Let DTIME($t(n)$) be the set of *all* languages with time complexity $O(t(n))$.
- DTIME($t(n)$) is a (time) complexity class of problems.

2 A Simple Example

- Consider $L_{01} = \{0^n 1^n \mid n \geq 0\}$.
- We know that L is decidable but not regular.
- How might we build a *single-tape* TM to decide L quickly?
- *Idea*: locate the middle and compare corresponding chars.
- On input w,
  1. Scan w and make sure it has form $0^i 1^j$. While doing this, mark locations of first 0 and of first 1.
  2. Reset to left end of tape.
  3. Advance the two marks, each one step at a time, until the first mark ends up over a 1. At that point, accept iff the second mark is over a blank preceded by a 1.
- How fast is this TM on input of size n?
- Initial scan, marking, and reset take $O(n)$ steps.
- Each advance of the two marks requires us to travel to the second mark, then back to the first, which surely takes $\Theta(n)$ steps.
- We have to go back and forth until we've checked all the 0's, so total time is $\Theta(n) \times \Theta(n) = \Theta(n^2)$.

So, $L_{01} \in$ DTIME(n^2). Is this the best we can do?

- Let's see if we can decide this language faster.
- Can we avoid making $\Theta(n)$ passes over the input?
- *Obs*: Suppose w has form $0^n 1^n$. If we look at only every other 0 and every other 1 of w (starting with left-most in each case), the resulting *decimated string* also has form $0^m 1^m$.
- If w does *not* have form $0^n 1^n$, but is in 0^*1^*, then either

  - the numbers of 0s and 1s have opposite parity, or
  - the numbers of 0s and 1s have the same parity.

- In the first case, we can trivially reject w. In the second case, the decimated string remaings unbalanced.
- This suggests the following algorithm.
  1. Check that $w \in \{0^*1^*\}$. If not, reject.
  2. If $w = \epsilon$, accept.
  3. If $w = 0$ or $w = 1$, reject.
  4. "Cross off" every other 0 and 1, starting with left-most of each.
  5. If parity of #'s of crossed off 0s and 1s differ, reject.
  6. Otherwise, recur from second step, considering only chars not crossed off.
- For correctness, proceed by induction on $|w|$ for all strings w of form 0^i1^j.
- Algo does the right thing for any string of length ≤ 1.
- For a longer string w, we've argued above that decimation rejects some strings of the wrong form, and recursively maintains right or wrong form for all others.
- by IH, decimated string is accepted iff it has form 0^m1^m.

Is the second algorithm faster than the first?

- Each pass takes constant time to check for base cases, plus $O(n)$ time for an input of size n to cross off even-numbered 0's and 1's.
- Each pass reduces the number of remaining characters by at least half.
- Hence, in at most $\log n$ passes, we reach a base case.
- Conclude that total time for this algo is $O(n \log n)$.
- Hence, $L_{01} \in \text{DTIME}(n \log n)$.

3 How Does Complexity Interact with TM Extensions?

- Previously, we introduced some extensions to TMs to help us describe algorithms with them.
- These extensions provably did not change the set of languages a TM can decide.
- But what do they do to complexity of decision algorithm?
- The simplest extensions (marking, remembering a cell) take at most constant time each time we use them.
- Seeking takes time proportional to the seek length.

Let's think about multi-tape TMs.

- We know that any k-tape TM can be simulated by a single-tape TM.
- But adding tapes can impact complexity!
- *Example*: we can easily decide L_{01} in time $O(n)$ using a 2-tape TM.
- After checking that $w \in 0^*1^*$, copy the 0s to tape 2, then use the heads on the two tapes to "match" the 0s on tape 2 to the 1s on tape 1.
- *Interesting fact*: we cannot decide L_{01}, or any non-regular language, in time $o(n \log n)$ on a single-tape TM (Prove it!) Hence, this two-tape TM can decide L_{01} strictly faster than any one-tape TM.
- Hmmm... so how much extra performance might multiple tapes buy us?
- **Thm**: if a k-tape TM can decide a language in time $t(n) \geq n$, then a single-tape TM can do it in time $O(t(n)^2)$.
- **Pf**: think about how we simulated k tapes on a single-tape TM.
- Each tape cell stored k symbols, one per tape.
- The head position for each tape was stored as a unique mark.
- In order to make one move of the k-tape TM, we must
  - locate and read the symbol under each tape's head
  - Update the symbol under each tape's head and move head by up to one cell.
- Heads could be at any position on the tape.
- Hence, after TM has taken m steps, we might have to scan up to m tape cells to find all the heads.
- Similarly, we might have to scan up to m cells to update all the heads after reading them all.
- Hence, simulating mth step of the multi-tape TM takes time $O(m)$ on the single-tape TM.
- Conclude that, if the multi-tape TM makes at most $t(n)$ moves to decide its input, then our single-tape simulation makes at most $O(t(n) \times t(n)) = O(t(n)^2)$ moves.
- **Bad News**: simulating multiple tapes (this way) has a nontrivial cost.
- **Good News**: if your multi-tape TM can decide a language in time $O(n^c)$, then a single-tape TM can decide it in time $O(n^{2c})$, which is still polynomial in n.
- **Caveat**: we did *not* prove that there is no faster way to simulate a multi-tape TM by a single-tape TM, or that any particular problem cannot be solved faster on a single-tape TM than via this multi-tape simulation.

OK, what about our other major extension – nondeterminism?

- Recall that a non-deterministic TM, like an NFA, can "try all possibilities" and accept if any set of nondeterministic choices leads to acceptance.
- **Example**: Let $X = \{x_1 \ldots x_n\}$ be a set of numbers.
- $L_{SSS} = \{X, t \mid \text{some subset of the } x\text{'s adds up to } t\}$.
- a non-deterministic TM could simply choose each x_i or not nondeterministically, then accept if the ones it chooses add up to t.
- For n distinct numbers, each of $\log n$ bits, this computation takes time $O(n \log n)$ (we can add two $\log n$-bit numbers in $O(\log n)$ time).
- Do you know of a deterministic, polynomial-time algorithm for this problem? (*Hint*: nobody does.)
- Nondeterminism seems awfully powerful!

What about the simulation we developed for nondeterminism?

- Let M be a non-deterministic TM. Say that at any step, M can choose one of at most $b \geq 2$ next moves non-deterministically.
- Our simulation uses a 3-tape TM M'.
- One tape caches a copy of the input w, while another stores non-deterministic choices for each move (as a base-b integer).
- The main tape simulated execution of M on w using the current set of non-deterministic choices. If M accepts, so does M'.
- We use iterative deepening search to try computations of length 1, 2, 3, and so forth, enumerating all b^k possible non-deterministic for computation of length k.
- How long does the simulation take?
- **Thm**: if a non-deterministic TM can decide a language in time $t(n) \geq n$, then a deterministic, single-tape TM can do it in time $2^{O(t(n))}$.
- **Pf**: M runs for at most $t(n)$ steps for *some* set of non-deterministic choices before accpting or rejecting.
- The simulation tries b^k computations of length k for each k up to $t(n)$.
- Hence, total number of steps in M is at most $\sum_{k=1}^{t(n)} ckb^k$ for some constant c.
- For $b \geq 2$, this sum is at most $ct(n)b^{t(n)}$.
- Applying a bit of math, we have

$$\begin{aligned} ct(n)b^{t(n)} &= b^{t(n)+\log_b t(n)+\log_b c} \\ &= 2^{\log_b(2)\times(t(n)+\log_b t(n)+\log_b c)} \\ &= 2^{O(t(n))}. \end{aligned}$$

- Now M' has three tapes, so a single-tape TM implementing this simulation might need time up to $d[2^{O(t(n))}]^2$.
- But this is still $2^{O(t(n))}$.
- Hence, we can simulate a non-deterministic TM by a deterministic, single-tape TM, but this simulation might take *exponentially* more time!
- Can we do better? Tune in next time!