CMPUT 675: Computational Complexity Theory **Winter 2019**

Lecture 2 (Jan 10): Undecidability and Hierarchy Theorems

Lecturer: Mohammad R. Salavatipour *Scribe: Zachary Friggstad*

In the previous notes, we discussed Turing Machines and small variations in the first definition.

But it is convenient to fix some features. From now on, we allow TMs to have any fixed number of tapes and to use any fixed finite alphabet. By *fixed*, we mean the number of tapes and the alphabet size is regarded as a constant when we discuss the running time or space used by a the TM on various inputs x.

We designate one tape to be the **input tape**. This is a read-only tape, meaning the symbols can never be replaced by anything else apart from their initial contents. The remaining tapes are **work tapes**. We may designate some of these work tapes to have specific tasks, but when we refer to the work tapes we mean all other tapes apart from the read-only input tape.

The head of the input tape is not allowed to stray beyond the bounds of the input. This is a slightly technical requirement for when we discuss space complexity. For example, we can enforce this by tweaking the model so that an input is presented with two delimiting symbols written in the cells around the input: the TM is not allowed to move the input tape head past these two delimiting symbols.

2.1 Undecidability

We begin by demonstrating that not all functions can be computed. Consider the language UC $\subseteq \{0,1\}^*$ containing every $\alpha \in \{0,1\}^*$ such that $M_\alpha(\alpha) \neq$ ACCEPT (i.e. the computation either does not halt or it halts in the state q_{REJECT}). Notationally,

$$\text{UC} = \{\alpha \in \{0,1\}^* : M_\alpha(\alpha) \neq \text{ACCEPT}\}.$$

Theorem 1 *The language* UC *is not decidable by any TM.*

Proof. Suppose, for the sake of contradiction, UC is decidable by a TM M. Let $\alpha \in \{0,1\}^*$ be an encoding of M (i.e. $M_\alpha = M$).

Consider the computation of $M_\alpha(\alpha)$. Because M_α decides UC, it must be that $M_\alpha(\alpha) =$ ACCEPT if and only if $\alpha \in$ UC itself. But by definition of UC, we also have $M_\alpha(\alpha) =$ ACCEPT if and only if $\alpha \notin$ UC. This is a contradiction. ■

The proof technique used here is called *diagonalization*: if such a TM existed then how would it behave when given its own description as input? The argument used here is similar to other well-known results:

- Cantor's proof that the real numbers are uncountable.
- Russel's paradox, eg. the following is not a set: $\{T : T \notin T\}$.

In fact, one can use Cantor's argument more directly to show the existence of languages that cannot be decided by a Turing Machine. The number of Turing Machines is countable as each can be encoded with a finite-length string in $\{0,1\}^*$. But the number of languages $\mathcal{L} \subseteq \{0,1\}^*$ is uncountable as each language $\mathcal{L}$ can be thought of

as an infinite-length bitstring where the i'th bit is 1 if and only if the i'th string in $\{0,1\}^*$ (according to some fixed ordering of $\{0,1\}^*$) lies in $\mathcal{L}$. Cantor's proof that the real numbers are uncountable is very easily adapted to show that the number of infinite-length bitstrings is uncountable.

So, in some sense, "most" languages are not decidable. But "most" languages are not interesting.

2.1.1 The Halting Problem

A more relevant language is HALT $= \{(\alpha, x) \in \{0,1\}^* \times \{0,1\}^* : M_\alpha(x) \text{ halts}\}$. Note we are again using the fact that we can somehow encode pairs of inputs, we will not discuss this low-level detail again.

We would really like to be able to decide this language! Knowing if a program would halt on some input is very valuable information. Unfortunately we cannot.

Theorem 2 *The language* HALT *is not decidable by any TM.*

Proof. Suppose M^{HALT} decides HALT. We show how to use M^{HALT} to decide UC, which will be a contradiction.

Here is a TM M^{UC} deciding UC. On input α it first runs $M^{\text{HALT}}(\alpha, \alpha)$. If $M^{\text{HALT}}(\alpha, \alpha) = \text{REJECT}$ then we know $M_\alpha(\alpha)$ does not halt. In this case, we have M^{UC} halt and accept α.

Otherwise, we know $M_\alpha(\alpha)$ is a halting computation. We have M^{UC} simulate $M_\alpha(\alpha)$ until it halts. If $M_\alpha(\alpha) = \text{ACCEPT}$ then have M^{UC} reject α, otherwise have M^{UC} accept α.

In this way, M^{UC} decides UC which contradicts the fact UC is not decidable. So HALT is not decidable. ■

This proof used a **reduction**. Essentially it showed that HALT is at least as "hard" as UC because a a TM deciding the former could also be used to decide the latter.

You may have already seen reductions when discussing **NP**-hardness. Polynomial-time reductions are used to show that if you have a polynomial-time algorithm for any single **NP**-hard problem then you can solve any problem in **NP** in polynomial time. In this way, **NP**-hard problems represent the "hardest" problems in **NP**. We will revisit this soon in this course.

There is more that can be discussed about decidability of problems. For now, our discussion continues with more examples of the diagonalization technique.

2.2 Time Complexity

Definition 1 *For any function $T : \mathbb{N} \to \mathbb{N}$, let* $\mathbf{DTIME}(T(n))$ *be all languages that can be decided by a Turing machine with running time $O(T(n))$.*

The **D** stands for **deterministic**, as we will shortly be covering variants of the standard Turing machine model and will need separate notation to discuss time complexity with those variants.

We can now properly define perhaps the most well-known class in complexity theory.

Definition 2 $\mathbf{P} = \bigcup_{c=0}^{\infty} \mathbf{DTIME}(n^c)$.

That is, **P** is all languages that can be decided in polynomial time. You know many of these languages:

- The language of connected graphs.
- The language of various arithmetic operations, such as $\{(a, b, c) : a \cdot b = c\}$.
- The language of prime numbers (the algorithm is not obvious).

The famous **P** vs. **NP** problem asks if there are some problems in **NP** that cannot be solved in polynomial time. But let us start with more basic questions. Do we know if there are even decidable problems that cannot be solved in polynomial time?

Thankfully, we can at least answer this question. But we need a technical definition.

Definition 3 *A function $T : \mathcal{N} \to \mathcal{N}$ is a* **time-constructible function** *if there is a TM which, given some input x, runs for exactly $T(|x|)$ steps.*

Note that functions like $\lfloor \log(x + 1) \rfloor$ are not time constructible. It takes $|x|$ time just for the TM to scan x to determine its length. Essentially all interesting functions T with $T(n) \geq n + 1$ are time constructible including polynomials, factorials, and exponentials. This is a programming exercise that we skip.

Theorem 3 (Time Hierarchy Theorem) *For any time-constructible function $f(n) \in \Omega(n)$,* $\mathbf{DTIME}(f(n)) \subsetneq \mathbf{DTIME}(f(n) \cdot \log f(n))$.

The proof uses the fact that a universal Turing machine exists that can simulate any Turing machine with logarithmic overhead in the running time. But we did not discuss this in the previous lecture. Using this TM we discussed last lecture in the following proof would show $\mathbf{DTIME}(f(n)) \subsetneq \mathbf{DTIME}(O(g(n))^2)$ if we have $f(n) \in \Omega(n^2)$.

Proof. We prove a marginally weaker result. That there is some $g(n) \in O(f(n) \log f(n))$ and some language $\mathcal{L}_f$ such that $\mathcal{L}_f$ can be decided in time $g(n)$ but not in time $f(n)$. The full extension to the asymptotic definition of **DTIME** follows essentially the same main ideas, but contains a number of distracting, low-level details.

Consider the language

$$\mathcal{L}_f = \{\alpha \in \{0, 1\}^* : M_\alpha \text{ does not accept input } \alpha \text{ in within } f(|\alpha|) \text{ steps}\}.$$

Note this is akin to the language UC from the start of these notes, except we are concerned with accepting within a given time bound rather than just accepting in finite time.

Now consider the following TM M^f that decides $\mathcal{L}_f$. On input α, M^f does the following:

- Write a string of $f(|\alpha|)$ 0s to a "clock" tape.
- Simulates the computation of $M_\alpha(\alpha)$ for up to $f(|\alpha|)$ steps. This can be done using the same algorithm as the universal TM, except after each simulated step of M_α it increments the head on the clock tape. If $M_\alpha(\alpha)$ does not halt within $f(|\alpha|)$ steps (i.e. the clock tape head goes past the 0s), the simulation stops.
- If $M_\alpha(\alpha)$ halted within $f(|\alpha|)$ steps, then have M^f output the **opposite** of what $M_\alpha(\alpha)$ output. Otherwise, have M^f accept.

It is easy to check that $\mathcal{L}_f$ is indeed the language decided by M^f.

Let $g(n)$ be the best possible bound on the running time of M^f. Observe $g(n) \in O(f(n) \cdot \log f(n))$ because the universal simulation has only logarithmic overhead and, by time constructibility, we can produce the 0s on the clock tape in $O(f(|x|))$ time.

We claim $\mathcal{L}_f$ cannot be decided by a TM with running time bound $f(n)$ to finish the proof. Suppose otherwise, that there is a TM M^* deciding $\mathcal{L}_f$ in time $f(n)$. In particular, $M^f(\alpha) = M^*(\alpha)$ for each possible input α.

Now for the diagonalization part. Pick $\alpha \in \{0,1\}^*$ to be an encoding of M^*. Then in the calculation of $M^f(\alpha)$, the simulation of $M^*(\alpha)$ will stop after $f(|\alpha|)$ simulated steps and $M^f(\alpha)$ will output the **opposite** of $M^*(\alpha)$. This is our contradiction. ■

We can now show that some decidable problems cannot be solved in polynomial time. First, consider some shorthand notation for exponential time algorithms.

Definition 4 $\mathbf{E} = \mathbf{DTIME}(2^n)$ *and* $\mathbf{EXP} = \bigcup_{c=0}^{\infty} \mathbf{DTIME}(2^{n^c})$.

Theorem 4 $\mathbf{P} \subsetneq \mathbf{E}$.

In fact, the following proof shows that if $g(n)$ is any function that is not $O(n^c)$ for any constant c, such as $n^{\log n}$, then $\mathbf{P} \subsetneq \mathbf{DTIME}(O(g(n)))$.

Proof. Simply put

$$\mathbf{P} \subseteq \mathbf{DTIME}(2^n/n) \subsetneq \mathbf{DTIME}(2^n).$$

The first inclusion is because any polynomial is asymptotically bounded by $2^n/n$. The second is by the Time Hierarchy theorem. ■

2.3 Space Complexity

Another resource we often care about is the amount of space or memory required to solve a problem.

Definition 5 *For any function $S : \mathbb{N} \to \mathbb{N}$, let* $\mathbf{DSPACE}(S(n))$ *be all languages that can be decided by a Turing machine such that the number of cells of all* **work tapes** *that are visited by the tape head at some point in the computation is $O(S(n))$.*

Of course, an input x will use $|x|$ non-blank cells. We are concerned with how much space is used in addition to the input but we don't want to cheat by allowing the input cells to, later, be used to store other contents of memory. This is why we require the input to be read-only.

We have a Space Hierarchy theorem as well. But first we again need to discuss what it means to be space constructible.

Definition 6 *A function $S : \mathcal{N} \to \mathcal{N}$ is a* **space-constructible function** *if there is a TM which, given some input x, stops after using precisely $S(|x|)$ space.*

Theorem 5 (Space Hierarchy Theorem) *Let $f(n)$ be a space-constructible function. For any $g(n) \in \omega(f(n))$,* $\mathbf{DSPACE}(f(n)) \subsetneq \mathbf{DSPACE}(g(n))$.

Note that $\lfloor \log_2(n+1) \rfloor$ is space constructible (a slightly pesky exercise), even though it is not time constructible. So we can even conclude that $\mathbf{DSPACE}(\log n) \subsetneq \mathbf{DSPACE}(\log n \cdot \log\log n)$.

Unlike the Time Hierarchy theorem, this says that any asymptotic increase in the allowed space yields a more powerful complexity class.

Proof.[sketch] The proof is essentially the same as the Time Hierarchy theorem. There are two main differences. First, observe that universal simulation of a TM can be done with only constant overhead in the amount of space used (which is why we get the Space Hierarchy separation for any $g \in \omega(f(n))$). Second, we consider

$$\mathcal{L}_f = \{\alpha \in \{0,1\}^* : M_\alpha \text{ does not accept input } \alpha \text{ using only } f(|\alpha|) \text{ space}\}.$$

The TM we construct in this proof will simulate $M_\alpha(\alpha)$ and terminate it prematurely if it ever uses more than $f(|x|)$ space. The rest of the proof proceeds like before using the diagonalization argument ■

We can define two common space complexity classes.

Definition 7 $\mathbf{L} = \mathbf{DSPACE}(\log n)$ *and* $\mathbf{PSPACE} = \cup_{c=0}^{\infty}\mathbf{DSPACE}(n^c)$.

In a way similar to how we proved $\mathbf{P} \subsetneq \mathbf{E}$, we can show:

Theorem 6 $\mathbf{L} \subsetneq \mathbf{PSPACE}$.

2.4 Relationships Between Time and Space

Theorem 7 *For any function* $f(n) \in \Omega(\log n)$, $\mathbf{DTIME}(f(n)) \subseteq \mathbf{DSPACE}(f(n)) \subseteq \mathbf{DTIME}\left(2^{O(f(n))}\right)$.

Note the slightly informal use of $\mathbf{DTIME}\left(2^{O(f(n))}\right)$. By this, we mean $\bigcup_{g \in O(f(n))} \mathbf{DTIME}\left(2^{g(n)}\right)$. We will occasionally use notation like this.

Proof. If a Turing Machine runs in time $O(f(n))$, it necessarily uses $O(f(n))$ space because each step can have each of its heads change by at most 1 position. So the first inclusion is simple.

For the second inclusion, we show that if a TM M always halts using space $O(f(n))$ then it runs in time $2^{O(f(n))}$. Let k be the number of tapes of M and Γ the alphabet and say that M uses at most $g(n)$ space for some $g(n) \in \Theta(f(n))$.

Consider some fixed input x to M. Let $\mathcal{C}$ be all cells of all work tapes that are visited by the head at some point. Note $|\mathcal{C}| \leq g(|x|)$. Let $\Gamma^{\mathcal{C}}$ be all possible assignments of symbols to cells in $\mathcal{C}$.

Since every cell outside of $\mathcal{C}$ remains blank throughout the entire computation, then we can specify a snapshot of the computation by just specifying the contents of $\mathcal{C}$, the state, and the head positions.

That is, a snapshot can be specified simply by a tuple in $\mathcal{S} := Q \times [0, |x|+1] \times [-g(|x|), g(|x|)]^{k-1} \times \Gamma^{\mathcal{C}}$. The term $[0, |x|+1]$ is the position of the input head and the term $[-g(|x|), g(|x|)]^{k-1}$ are for the positions of the remaining heads.

No two steps of the computation of $M(x)$ have the same snapshot, or else the TM would be stuck in a loop including this snapshot and would not halt (i.e. a snapshot entirely determines the next snapshot). So the number of steps taken by $M(x)$ is at most

$$|\mathcal{S}| = |Q| \cdot (x+2) \cdot (2g(|x|)+1)^k \cdot 2^{g(|x|)\cdot \log_2 |\Gamma|}.$$

As $|Q|, |\Gamma|$ are constants independent of $|x|$ and since $g(n) \in \Omega(\log n)$, this quantity is bounded by $2^{O(g(|x|))}$. ■

Note this proof does not even require us to modify TMs to show the inclusions. A TM using time $O(f(n))$ necessarily uses space $O(f(n))$ and a TM using space $O(f(n))$ necessarily uses time $2^{O(f(n))}$.

2.4.1 On Time and Space Constructibility

The requirement of having $f(n)$ be time constructible in the Time Hierarchy theorem may seem like a technical requirement to make the proof easier. But it is an essential assumption! The results here are presented without proof. Let me know if you want references.

Theorem 8 *There is a computable function $f(n)$ that tends to ∞ such that* $\mathbf{DTIME}(f(n)) = \mathbf{DTIME}(2^{f(n)})$.

While such an $f(n)$ is computable, it is a very strange function. One should not take this result too seriously. Really it says that computation is not very "well behaved" at this level.

In the same vein, we have another strange result that shows space constructibility is required in the Space Hierarchy theorem.

Theorem 9 *For any $g = o(\log\log n)$,* $\mathbf{DTIME}(g(n)) = \mathbf{DTIME}(1)$.

In fact, the proof shows that any TM that uses $o(g(n))$ work space **necessarily** uses $O(1)$ space. It is impossible to construct a TM that uses, say, $\Theta(\log\log\log n)$ work space. This result is tight in the following sense: it is known that $\mathbf{DTIME}(1) \subsetneq \mathbf{DTIME}(\log\log n)$.