Notes for CS251 Sat Sep 21 review session. I expanded on a few places here and there that I did not have time to during the lecture. Furthermore some presentations were streamlined.

Acronyms:

- TM: Turing Machine
- DFA: Discrete Finite Automata

1 TMs/DFAs as data

Recall from 2 HWs ago that we studied lots of languages with (encodings of) DFAs satisfying a certain property. We can also treat TMs as data.

This means that **we can use TMs to reason about TMs**.

1.1 An old CYU

Recall the following CYU:

"Why do we define TMs to have finite states?"

The official answer is of the form "because a computer in real life cannot have infinite states." This answer is wholly inadequate.

1. In mathematics we study many objects without a physical realization. The set of natural numbers is not some thing that exists in nature.
2. Our tape is infinite, so "physically realizing" a TM is not the point anyway.

Here is the real reason. **It is so that a TM and its current configuration are finitely describable.** This means that TMs can read in (encodings of) other TMs, because said encoding is finite. TMs can simulate, store, and manipulate other TMs because the configuration uqv is also finite (u and v are always finite strings).

TMs have finite states so that they can be used to reason about the behavior of other TMs.

2 The limits of computation

Now that we know TMs can study other TMs, two natural questions arise:

1. Can TMs solve every decision problem?
2. Can TMs analyze the behavior of other arbitrary TMs?

The answer to both of these is **no**. To see why, we need to first rehash some mathematical tools.

2.1 The mathematical machinery: countability

Let me remind you what countability means.

Definition 2.1 (Countable): We define what it means for a set S to be **countable** from two different perspectives.

1. There is the math definition: S is countable if there exists a bijection $f : S \to \mathbb{N}$. We may also establish an injection $S \to \mathbb{N}$ or a surjection $\mathbb{N} \to S$, both are also enough.
2. The CS definition: officially it is about the existence of an encoding. But I find the encoding definition to be a bit abstruse. Here is the intuition you should approach this with.

   A set is countable if each of its objects has a finite description. The finite descriptions must not overlap, of course (this is really the injectivity requirement). Mathematically they are the same idea, but pedagogically this is much more intuitive.

Let me give you some examples of the intuitive version of the CS definition: **finite descriptions**.

1. From recitation: is $\mathbb{N}^*$, i.e. the set of finite tuples of natural numbers, countable? Unless you are in my recitation, your TA probably spent a while on this question. Using our newfound intuition, we will solve it in about 2 seconds.

   Is a tuple like $(1, 2, 3)$ finitely describable? Yes! You are looking at the description right now: it is literally the string "$(1, 2, 3)$". Likewise I can finitely describe any other tuple in the same way.

   Now we should be careful to note that we implicitly needed each element of the tuple to be finitely describable. Fortunately, natural numbers are finitely describable for the obvious reasons.

2. Are the integer matrices countable? Yes, I can describe the matrix

   $$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

   by writing down exactly that matrix in exactly that way. Strictly speaking, is it a string? No. But use the LaTeX code used to typeset that matrix if you so wish. Details aside, what I have written is evidently a finite description of a matrix. And of course, every other integer matrix can be represented in the same way.

3. Are the finite subsets of $\mathbb{N}$ countable? Yes. You know the drill by now. A finite subset like $\{1, 2, 3\}$ can be explicitly listed out, so it has a finite description.

To be clear, this is not a replacement for finding an explicit bijection or encoding. But this perspective makes it very easy to see that some sets are countable, and it shows you exactly why.

2.2 The mathematical machinery: Cantor's Theorem

Theorem 2.2: Given a set S and its powerset $\mathcal{P}(S)$,

$$|S| < |\mathcal{P}(S)|.$$

First I give you the mathematical proof.[1] Then I will share an analogy from https://algorithmsoup.wordpress.com/2018/09/18/soviet-version-of-cantors-diagonalization-argument/.

Proof of Theorem 2.2: We will show that any $f : S \to \mathcal{P}(S)$ is not a surjection. Consider the subset

$$D = \{x \mid x \notin f(x)\}$$

of S. No matter which x we pick, we cannot have $f(x) = D$, for

$$x \in D \iff x \notin f(x).$$

But we need

$$x \in D \iff x \in f(x)$$

for $f(x) = D$, because for two sets to be equal they must contain the same elements. □

And now here is the Soviet Russia joke presented in my lecture.

"In Soviet Russia, there are infinitely many citizens. Every subset of the citizens forms their own secret underground organization. That includes not only the standard finite subsets, but also the infinite ones (which typically spend the vast majority of their time taking attendance), as well as the empty set (which happens to be one of the most productive).

The abundance of secret organizations causes a headache for the secret police. In order to handle this, they decide to assign every citizen to be a spy on one secret organization.

There are two types of spies: happy spies who are assigned to spy on an organization of which they are a member; and unhappy spies who are assigned to spy on an organization of which they are not a member.

The secret police want to assign spies to every organization. But, being highly trained mathematicians, they realize there's a problem. No matter how they assign spies to organizations, at least one secret organization is guaranteed not to be spied on: the organization consisting of all the unhappy spies.

In other words, no matter how large the set of citizens may be, the set of secret underground organizations is provably larger."

P.S. There is an obvious bijection between $\mathcal{P}(S)$ and the set of functions $f : S \to \{0, 1\}$. We associate $X \subseteq S$ with the function

$$f(s) := 1 \text{ if } s \in X$$
$$0 \text{ if } s \notin X.$$

I leave it to you to check this is a bijection.

2.3 Back to our two questions

Note. I will abuse terminology a little and refer to decidable problems and solvable problems interchangably. This is because the string "undecidable decision problem" looks a little silly.

TMs cannot solve every decision problem. We know every decision problem $f : \Sigma^* \to \{0, 1\}$ corresponds to a subset of Σ^*, and as $|\Sigma^*| = |\mathbb{N}|$ we have $|\mathcal{P}(\Sigma^*)| > |\mathbb{N}|$.

[1] This is different from the one presented in lecture.

Yet the set of all TMs is countable, because as we have established, TMs are finitely describable. There are more decision problems than TMs, and each TM may solve at most one decision problem. So we will be left with unsolvable decision problems.

TMs cannot solve the halting problem. This argument will be a little more sophisticated. First we will enumerate the TMs as $M_0, M_1, \ldots$ and enumerate the strings in Σ^* as $x_0, x_1, \ldots$

Now by diagonalization we know there is no TM M_U where for all $n \in \mathbb{N}$,

$$M_n(x_n) \text{ halts} \iff M_U(x_n) \text{ does not halt.}$$

This is because such a hypothetical M_U would behave differently from every M_n when taking the string x_n as input.

But now we reduce the behavior of M_U to the halting problem. More precisely, presume that some TM $M_H\langle M, x\rangle$ solves the halting problem. Then we may define M_U as such:

```
def M_U(x_n):
  if M_H(<M_n, x_n>) returns "halt":
    do not halt
  else # M_H(<M_n, x_n>) returns "does not halt"
    halt
```

(Implicitly this procedure presumes that given a string $x \in \sigma^*$, we can determine which x_n it is in finite time. This is easy: just compare x with $x_0, x_1, \ldots$ until we get to x_n.)

But hold on! We just showed such an M_U must not exist. So M_H could not possibly exist either.

3 Decidability and reductions

Definition 3.1: A language $L \subseteq \Sigma^*$ is **decidable** if there exists a TM M such that

$$x \in L \implies M(x) \text{ accepts } x \notin L \implies M(x) \text{ rejects}$$

Contrast this with a **semi-decidable** language L, which merely posits the existence of a TM M where

$$x \in L \implies M(x) \text{ accepts } x \notin L \implies M(x) \text{ rejects OR doesn't halt}$$

Note a **decider** for a decidable language always halts, while a **semi-decider** only halts on $x \in L$.

Reductions have two perspectives:

1. $L \leq K$, i.e. "L reduces to K", means that if some "magic genie" M_K solves K, we can use M_K as a subroutine for a decider M_L of L.
2. $L \leq K$ means "the difficulty of L is at most that of K".

It turns out the two perspectives are equivalent, something we will not show here. Being able to tie the two perspectives together is very helpful. At a base level it means you will be able to rederive that $L \leq K$ means "L reduces to K" rather than the other way around.

Let me elaborate on the second perspective. Remember the formal definition of $p \to q$ in Concepts/DMP? Really if we assign false $\mapsto 0$ and true $\mapsto 1$, the statement $p \to q$ really is saying $p \leq q$.

Now if we assign "difficulty values" to languages, where 1 is "difficult" (i.e. undecidable) and 0 is "easy" (decidable), then substituting in this numerical interpretation is perfectly consistent with the sentence "$L \leq K$".

What can a reduction tell us? If we know K is decidable, then $L \leq K$ means $L \leq 0$, i.e. L is decidable. And if L is undecidable, then $L \leq K$ means $1 \leq K$, i.e. K is undecidable.

Moving forwards, this is how we will usually show a problem is undecidable. More rarely we will use it to show a problem is decidable, though this is usually easier as we can just provide a direct construction of an algorithm that decides the question.

There are many examples of this in the textbook/recitation/HW (unlike the other topics and perspectives I have mentioned) so I will not rehash them here.