**CS 4510 Automata and Complexity** March 13th 2023

# Lecture 14: Undecidability by Reduction

*Lecturer: Abrahim Ladha* *Scribe(s): Abrahim Ladha*

Today we are going to solidify our understanding of what we can know about the unknown. Recall last time we discussed the work of Russell, Gödel, and Turing. We showed there exist unanswerable questions in two ways

- There exist unprovable statements
- There exist unsolvable problems

We proved these using a specialization of the diagonalization technique. Note that nothing is preventing us from stating these unsolvable problems or unprovable statements, only proving or solving them. Today we expand on Turing's work. Our first known undecidable language is

$$HALT = \{ \ \langle M, w \rangle \ | \ M \text{ halts on } w\}$$

## 1 Some Closure

Recall the definition of decidable and recognizable languages. We say a language $L$ is decidable ($L \in \mathscr{L}_D(TM)$) if there exists a Turing machine $M$ such that

- $w \in L \iff M$ accepts $w$
- $w \notin L \iff M$ rejects $w$

We say a $L$ language is recognizable ($L \in \mathscr{L}_R(TM)$) if there exists a Turing machine $M$ such that

- $w \in L \iff M$ accepts $w$
- $w \notin L \iff M$ rejects or loops on $w$

Notice that by definition, $\mathscr{L}_D(TM) \subsetneq \mathscr{L}_R(TM)$. We showed that $HALT \notin \mathscr{L}_D(TM)$ but it turns out that $HALT$ is recognizable! Lets give a recognizer. Notice this correctly

**Algorithm 1** Recognizer for $HALT$

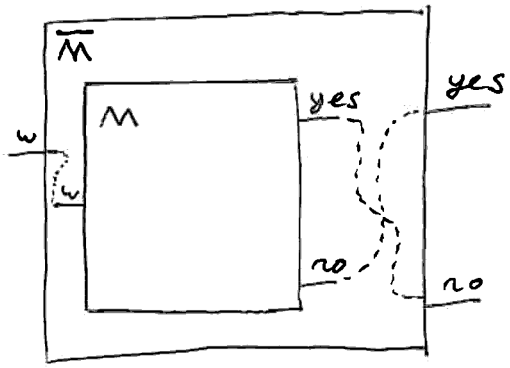
```
on input ⟨M, w⟩
simulate M on w
if M accepts or rejects w then
    accept
end if
```

recognizes $HALT$. If $\langle M, w \rangle \in HALT$ then we know $M$ halts on $w$, so if we simulate it we halt and correctly accept. But if $\langle M, w \rangle \notin HALT$, then $M$ loops on $w$ and so do we. The

step of simulating $M$ on $w$ does not terminate and we do not reach the conditional. This is then a correct recognizer for $HALT$. Since $HALT$ is recognizable but not decidable, our containment is strict. So Turing proved that not every language is decidable. But is every language recognizable? By a similar counting argument, we know that there are uncountably many languages and only countably many recognizable languages, so most languages are unrecognizable. Is there a notable unrecognizable language, in the same sense that $HALT$ is a notable undecidable language? Lets prove two theorems about closure to show the answer is yes.

### 1.1 A First Theorem

First we show that the decidable languages are closed under complement. That $L \in \mathscr{L}_D(TM) \iff \overline{L} \in \mathscr{L}_D(TM)$, that the decidable languages are closed under complement. This one is easy. If a language is decidable, then there exists a decider $M$ for it. Since its a decider, it halts on all inputs. Construct a new Turing machine $\overline{M}$ which is just $M$ but we swapped its accept and reject state. Since $M$ was a decider, so is $\overline{M}$, and we see that it decides $\overline{L}$, so it is decidable. We may represent this visually using the following diagram for $\overline{M}$.

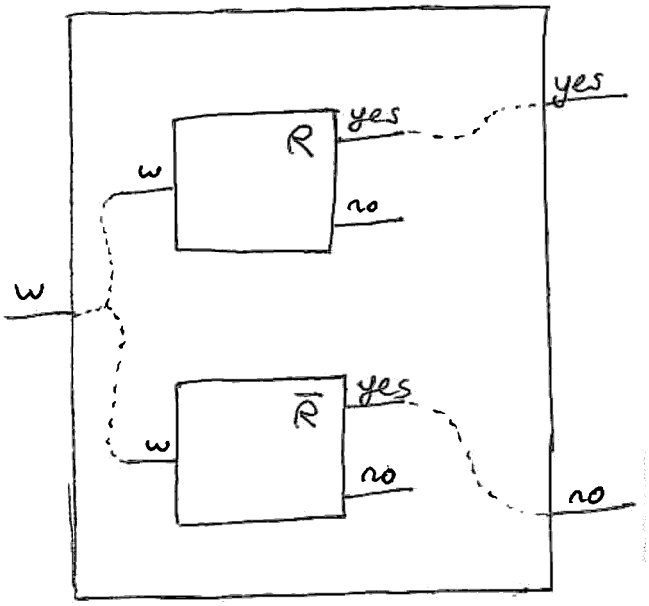



### 1.2 A Second Theorem

Now lets prove that if $L, \overline{L} \in \mathscr{L}_R(TM) \implies L \in \mathscr{L}_D(TM)$. If a language and its complement are both recognizable, then the language is decidable. Suppose that $L, \overline{L}$ are recognizable with recognizers $R, \overline{R}$. We give a decider for $L$ as follows. A recognizer is not guaranteed to halt on all inputs, but it is guaranteed to halt on the good ones. Let us argue correctness, and why our construction is a decider. If $w \in L$, then by definition $R$ halts and accepts on $w$, so our machine will halt and accept if $w \in L$. If $w \notin L$, then by definition $\overline{R}$ halts and accepts, so our machine will halt and reject if $w \notin L$. Then for all $w$, it correctly and exactly decides $L$ so we see that $L$ is decidable. We also now give the equivalent circuit diagram. Of independent interest, this proof shows you how to run two simulations "in parallel". They aren't really in parallel, but rather dovetailed together one at a time. This is necessary. For example, if $w \in L$, this may make $\overline{R}$ loop on $w$. But $R$ will eventually halt on $w$. We could not run these simulations sequentially.

**Algorithm 2** Decider for $L$

```
on input w
R = ...
R̄ = ...
while True do
    Simulate R on w for one step
    Simulate R̄ on w for one step
    if R accepts then
        accept
    end if
    if R̄ accepts then
        reject
    end if
end while
```



### 1.3 An Unrecognizable Language

We now immediately apply the theorems to find a useful unrecognizable language. Assume to the contrary that $\overline{HALT}$ is recognizable. Then since $HALT$ is recognizable, this would imply that $HALT$ is decidable. But we proved by diagonalization its not decidable, a contradiction. Therefore, $\overline{HALT}$ is not recognizable.

Here we have proved existence of an unrecognizable language. The takeaway is that we did not have to use diagonalization. Simply by the fact we can relate problems to one another, we were able to prove this language was also undecidable and unrecognizable. Let us generalize this idea to prove many more languages are undecidable.

## 2 Many-One Reductions

Recall that a function $f : \Sigma^* \to \Sigma^*$ is said to be computable (or Turing-computable) if there exists a Turing machine $M$ such that for every $w \in \Sigma^*$, $M$ begins with $w$ on the tape and halts with $f(w)$ on the tape. By the Church-Turing Thesis, in some sense, the computable functions are the largest class of functions.

For $A, B \subseteq \Sigma^*$ languages, we say that $A$ is many-one[1] reducible to $B$ (written as $A \leq_m B$) if there exists a computable function $f$ such that $w \in A \iff f(w) \in B$. Notice that this also implies that $w \in \overline{A} \iff f(w) \in \overline{B}$. It maps $A$ to $B$ and $\overline{A}$ to $\overline{B}$. The $\leq_m$ relation satisfies the following few properties. If $A \leq_m B$ then:

- If $B$ is decidable, then $A$ is decidable ($B \in \mathscr{L}_D(TM) \implies A \in \mathscr{L}_D(TM)$)
- If $A$ is undecidable, then $B$ is undecidable ($B \notin \mathscr{L}_D(TM) \implies A \notin \mathscr{L}_D(TM)$)
- If $B$ is recognizable, then $A$ is recognizable ($B \in \mathscr{L}_R(TM) \implies A \in \mathscr{L}_R(TM)$)
- If $A$ is unrecognizable, then $B$ is unrecognizable ($B \notin \mathscr{L}_R(TM) \implies A \notin \mathscr{L}_R(TM)$)

This about how going left is "simpler" and going right is "more unsolvable". The relationship between $A, B$ if $A \leq_m B$ is that $A$ lower bounds $B$, and $B$ upper bounds $A$. The four statements we mentioned are intuitive but require proof. We only prove the first one. The other three are proved similarly.

Suppose that $A \leq_m B$ and $B$ is decidable. Then there exists a computable function $f$ as our many-one reduction. We give a decider for $A$ as follows.

**Algorithm 3** Decider for $A$ given $f$ as the reduction $A \leq_m B$ and decider for $B$

```
on input w
compute f(w)          ▷ This computation halts by definition of a computable function
if f(w) ∈ B then      ▷ Since B is decidable, we can run its decider on f(w)
    accept
else                  ▷ If the decider for B rejects, we reject
    reject
end if                ▷ Note this halts on all inputs, so it is a decider
```

[1] or mapping

# 3 Some Reductions

We now use the method of reduction to show more undecidable problems.

## 3.1 An Acceptance Problem

Let

$$A_{TM} = \{ \ \langle M, w \rangle \ | \ M \text{ accepts } w \}$$

This language looks close to $HALT$, so it shouldn't be surprising that it is also undecidable. We will prove it is undecidable not by diagonalization, but by reduction. We will show $HALT \leq_m A_{TM}$[2].

Assume to the contrary that $A_{TM}$ is decidable. We give a decider for $HALT$.

**Algorithm 4** Decider for $HALT$ given decider for $A_{TM}$

```
on input ⟨M, w⟩
if ⟨M, w⟩ ∈ A_TM then                ▷ If M accepts w it certainly halts on w
    accept
else                                 ▷ If M doesn't accept on w, it must reject or loop
    build M' from M swapping accept and reject states q_a, q_r
    if ⟨M', w⟩ ∈ A_TM then           ▷ M' accepts w ⟺ M rejects w
        accept                       ▷ If M rejects w, it certainly halts on w
    else                             ▷ If M doesn't accept or reject w, it must loop
        reject
    end if
end if                               ▷ Note this halts on all inputs, so it is a decider
```

Given that $A_{TM}$ was decidable, we were able to construct a decider for $HALT$. But we know by diagonalization that $HALT$ is undecidable, a contradiction. Therefore, we see that $A_{TM}$ is undecidable. Note that $A_{TM}$ is recognizable and $\overline{A_{TM}}$ is unrecognizable for similar reasons to $HALT$ and $\overline{HALT}$.

## 3.2 An Emptiness Problem

Let

$$E_{TM} = \{ \ \langle M \rangle \ | \ L(M) = \emptyset \ \}$$

This language consists of encodings of Turing machines which accept nothing. We show it is undecidable. The reduction for this one is slightly more advanced. There exists no reduction $A_{TM} \leq_m E_{TM}$[3] but we can do the reduction $A_{TM} \leq_m \overline{E_{TM}}$. By showing the complement $\overline{E_{TM}}$ is undecidable, so must be $E_{TM}$.

Assume to the contrary $E_{TM}$ is decidable. We give a decider for $A_{TM}$

Here, our decider constructs a new machine $M'$ with hardcoded $M, w$. Now notice that $M'$ automatically rejects all strings which aren't $w$. So $L(M') =$ either $\emptyset$ or $\{w\}$. Which one

[2]Its notable the Sipser book does the reverse, showing $A_{TM}$ is undecidable by diagonalization, and then showing $HALT$ is undecidable by reduction, namely $A_{TM} \leq_m HALT$

[3]See the exercise in the Sipser book

**Algorithm 5** Decider for $A_{TM}$ given decider for $E_{TM}$

```
on input ⟨M, w⟩
construct M' with M, w hardcoded
if ⟨M'⟩ ∈ E_TM then
    reject
else
    accept
end if
```

**Algorithm 6** $M'$ hardcoded from $M, w$

```
on input x
M = ...
w = ...
if x ≠ w then
    reject
else
    Simulate M on input w
    if M accepts w then
        M' accepts x
    end if
end if
```

depends on what happens to $M$ on input $w$. Our many-one reduction is $f(\langle M, w\rangle) = \langle M'\rangle$ such that

- if $\langle M'\rangle \in E_{TM}$, then $L(M') = \emptyset$. So $w \notin L(M')$ so $M$ must have rejected or looped on $w$. Either way $\langle M, w\rangle \notin A_{TM}$
- if $\langle M'\rangle \notin E_{TM}$, then $L(M') = \{w\}$. This was only true if $M$ accepted $w$ so we see that $\langle M, w\rangle \in A_{TM}$.

We witness that $\langle M, w\rangle \in A_{TM} \iff \langle M'\rangle \notin E_{TM} \iff \langle M'\rangle \in \overline{E_{TM}}$ so $A_{TM} \leq_m \overline{E_{TM}}$. We conclude that $E_{TM}$ is undecidable.

### 3.3 An Equivalence Problem

The more languages we prove are undecidable, the easier the next ones become. We have to make the educated decision on which undecidable language to reduce from, but atleast we have the choice. Let

$$EQ_{TM} = \{\ \langle M_1, M_2\rangle \mid L(M_1) = L(M_2)\}$$

This language is the set of encodings of pairs of Turing machines which recognize the same language. We now prove it is undecidable. This reduction is much simpler, and we choose to reduce from $E_{TM}$ to show $E_{TM} \leq_m EQ_{TM}$.

Assume to the contrary $EQ_{TM}$ is decidable. We give a decider for $E_{TM}$

**Algorithm 7** Decider for $E_{TM}$ given decider for $EQ_{TM}$

on input $\langle M \rangle$
Let $M_\emptyset$ be some hardcoded Turing machine to reject all strings
**if** $\langle M, M_\emptyset \rangle \in EQ_{TM}$ **then**
    accept
**else**
    reject
**end if**

This one is pretty simple. Its logically equivalent to `def iszero(x): return x==0`. Our reduction is $f(\langle M \rangle) = \langle M, M_\emptyset \rangle$. We observe that $\langle M \rangle \in E_{TM} \iff \langle M, M_\emptyset \rangle \in EQ_{TM}$ so $E_{TM} \leq_m EQ_{TM}$ and $EQ_{TM}$ is undecidable.

Of the languages we have shown, this one is the most applicable. Suppose you were given some code and asked to rewrite it in a modern language. How do you know if your rewrite is equivalent? I mean like semantically equivalent. On all inputs, the programs behave identically and equivalently. Since this language is undecidable, there is no algorithm which could take in both pieces of code and definitively say yes or no if they are semantically equivalent. Thats why the best you can do is a thousand unit tests and hope there is no missing case. This can never guarantee they are equivalent, but it can guarantee they may be close enough you couldn't notice a difference if there was one. Specifically for $EQ_{TM}$, it is a "more unsolvable" problem than the ones we have shown. Lets prove it.

### 3.4 A "More Unsolvable" Problem

In general for a language $L \subseteq \Sigma^*$ and a class $C \subseteq \mathcal{P}(\Sigma^*)$, we say that $L \in$ co-$C$ if $\overline{L} \in C$. It is not true in general that $\overline{C} =$co-$C$. We specifically say that a language is co-Turing recognizable (or co-recognizable) if $\overline{L} \in \mathscr{L}_R(TM)$. Note that since $HALT$ was recognizable and not decidable, $\overline{HALT}$ is co-recognizable and not decidable. The languages which are both recognizable and co-recognizable are exactly the decidable languages. This will elucidate our map later on. To show a language isn't recognizable, we may combine the following facts

- If $A \leq_m B$, and if $A$ isn't recognizable, niether is $B$
- $A_{TM}$ is recognizable and not decidable, $\overline{A_{TM}}$ is co-recognizable, unrecognizable, and not decidable.
- $A \leq_m B \iff \overline{A} \leq_m \overline{B}$

We combine these facts to prove some $B$ is unrecognizable by showing either of the following

$$\overline{A_{TM}} \leq_m B \iff A_{TM} \leq_m \overline{B}$$

We can give a reduction from $A_{TM}$ to the complement of a language, to show a language is unrecognizable. We have shown many undecidable languages, but could they perhaps be recognizable or co-recognizable. Are there any which are neither recognizable nor co-recognizable? Yes, lets prove it. We show $EQ_{TM}$ is neither recognizable nor co-recognizable.

In some sense, this makes it "more unsolvable" than any of the languages we have shown so far. A recognizer for an undecidable language feels atleast half right[4].

First we show $EQ_{TM}$ is not recognizable. Sipser has a more informative reduction to prove this. I found this shorter, cuter proof using only the calculus of reductions, but it is less informative. Recall we proved $A_{TM} \leq_m \overline{E_{TM}}$. This implies $\overline{A_{TM}} \leq_m E_{TM}$. Also recall we proved $E_{TM} \leq_m EQ_{TM}$. Many-one reducibility is a transitive relation (something you should have to prove) so we see that

$$\overline{A_{TM}} \leq_m E_{TM} \leq_m EQ_{TM} \implies \overline{A_{TM}} \leq_m EQ_{TM}$$

Since $\overline{A_{TM}}$ is unrecognizable, so is $EQ_{TM}$.

Now lets show $EQ_{TM}$ isn't co-recognizable. We equivalently show $\overline{EQ_{TM}}$ isn't recognizable. We would show $\overline{A_{TM}} \leq_m \overline{EQ_{TM}}$ but this is equivalent to $A_{TM} \leq_m EQ_{TM}$. So it suffices to give a reduction from $A_{TM}$ to $E_{TM}$. We want a computable function $f(\langle M, w\rangle) = \langle M_1, M_2\rangle$ such that

$$\langle M, w\rangle \in A_{TM} \iff \langle M_1, M_2\rangle \in EQ_{TM}$$

Our reduction is as follows.

**Algorithm 8** Reduction from $A_{TM}$ to $EQ_{TM}$

on input $\langle M, w\rangle$
build $M_{\Sigma^*}$ to accept all strings
build $M_2$ such that on input $x$, it runs $M$ on $w$ and accepts $x$ if $M$ accepts $w$
return $\langle M^*_\Sigma, M_2\rangle$

Notice that $L(M_{\Sigma^*}) = \Sigma^*$ obviously. So

$$\langle M_{\Sigma^*}, M_2\rangle \in EQ_{TM} \iff L(M_2) = \Sigma^* \iff M \text{ accepts } w \iff \langle M, w\rangle \in A_{TM}$$

We conclude that $EQ_{TM}$ is not recognizable or even co-recognizable.

### 3.5 Language Problems for our Other Computational Models

What is the decidability these language problems relative to our other, weaker automata? Consider the following table. Let $D$ mean decidable and $U$ mean undecidable.

| | $A\_$ | $E\_$ | $EQ\_$ | $ALL\_$ |
|---|---|---|---|---|
| DFA, NFA, REGEX | D | D | D | D |
| CFG, PDA | D | D | U | U |
| TM, and more | U | U | U | U |

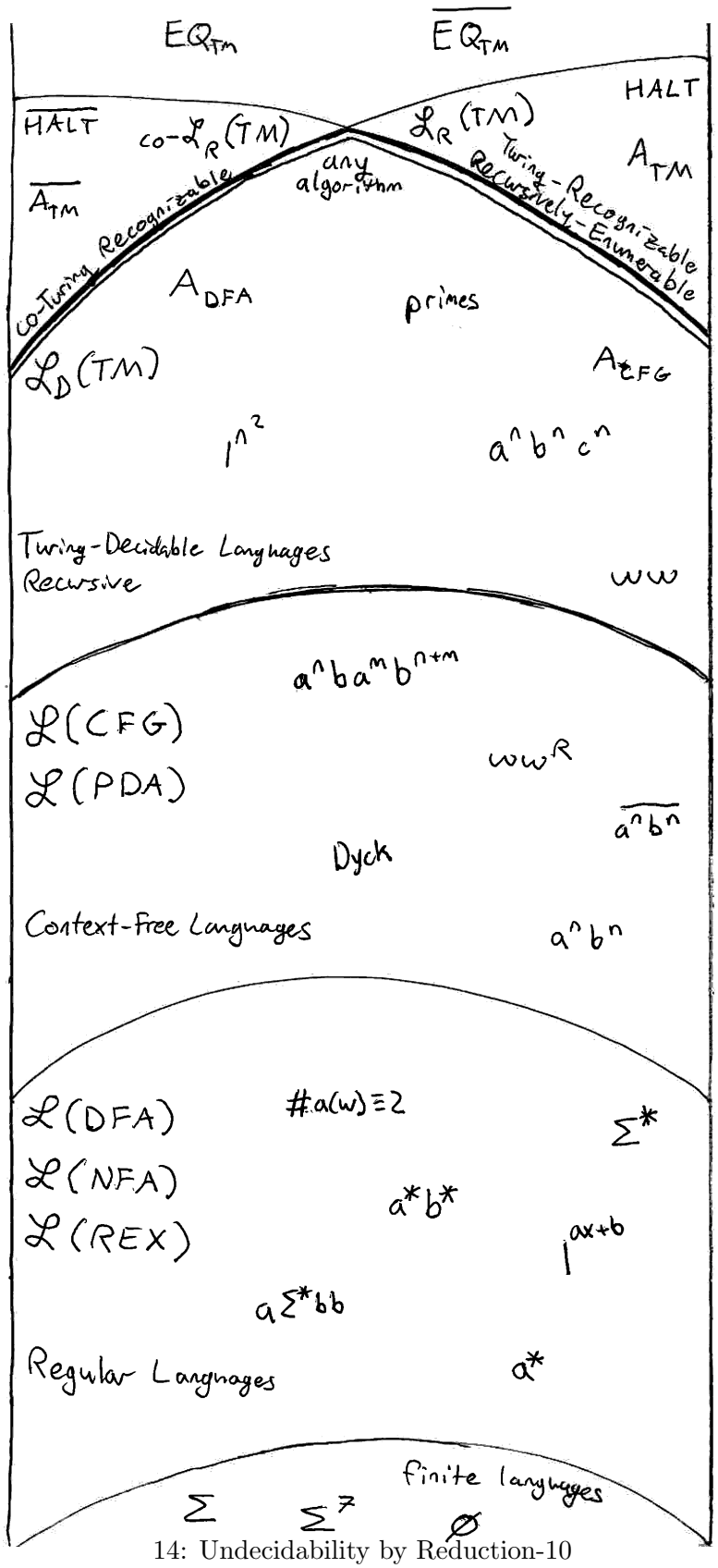
Lets try to give a brief summary of what is an entire chapter of Sipser. Note that if we have proved two kinds of computational models or automata to be equivalent, they should both be decidable or both be undecidable. Otherwise, you could transform from one to the other, and decide it.

[4]some other books even call recognizable languages semi-decidable

- $A_{DFA}$ is decidable, simply run the word on the DFA. Similarly by the equivalence of DFAs, NFAs, and regular expressions, they also all have decidable acceptance problem.
- $E_{DFA}$ is decidable. Treat the DFA like a graph and see if an accept state is reachable from the start state using DFS or BFS or any other graph traversal algorithm. Similarly, $ALL_{DFA}$ is decidable by checking to see if a reject state is reachable from the start state.
- $EQ_{DFA}$ is decidable. We didn't prove it, but there exists an algorithm to convert DFAs into a "normal form" where they are isomorphic (in a vertex and edge colored graph sense) if they decide the same language. A regular language cannot have two different looking DFAs for it and both of them be in this minimal normal form.
- $A_{CFG}$ is decidable. This was the point of Chomsky normal form. There also exists the CYK dynamic programming algorithm to decide this.
- $E_{CFG}$ is decidable. For each non-terminal, you mark it if it is capable of producing strings. You repeat this until you can test if the start non-terminal is capable of producing strings.
- $EQ_{CFG}, ALL_{CFG}$ are both undecidable. This should surprise you. Its certainly seems like a hard problem. While you can tell if a CFG produces any word, or a specific word, how can you decide if a CFG doesn't non-deterministically skip over some word? You can't. Since CFLs aren't closed under complement, $E_{CFG}, ALL_{CFG}$ do not have the same duality like they do for the regular languages. The proof of this undecidability is not beyond you, but it would simply take a lecture. It uses the method of computation histories, which we will go into next lecture. This also implies that for two semantically equivalent grammars, there doesn't exist a normal or minimal form like there is for DFAs. Chomsky normal form isn't then really a "normal form".
- For Turing machines, we proved that $A_{TM}, E_{TM}, EQ_{TM}$ are undecidable. $ALL_{TM}$ is undecidable for similar reasons. This also holds true for any Turing-complete computational model.

The takeaway here is that the more powerful a computer is, the less we can know about the languages they decide, just from looking at their descriptions. Notice that these are all language membership problems. All code problems are decidable. That would include things like "This Turing machine has seventeen states", easily checkable. We don't care about the computers at all. Our true love is the languages. We only use these computational models as a tool to study the classes of languages that they characterize.

It took fourteen lectures, but we finally have enough information to give a full and complete world map. Note that we are quite limited. By the Church-Turing Thesis, anything beyond the decidable languages is incomprehensible. Unfathomable. The recognizable and co-recognizable languages are atleast half fathomable. This may look like a complete map, but we know the world is much bigger than what we can understand. There is an edge to it. The part we can see is only countably many of the languages. A tiny pathetic window into the vast scale of the uncountably large universe of languages.


$EQ_{TM}$
$\overline{EQ_{TM}}$
HALT
$\overline{HALT}$
$co\text{-}\mathcal{L}_R(TM)$
$\mathcal{L}_R(TM)$
$A_{TM}$
$\overline{A_{TM}}$
any algorithm
co-Turing Recognizable
Turing-Recognizable
Recursively-Enumerable
$A_{DFA}$
primes
$\mathcal{L}_D(TM)$
$A_{CFG}$
$1^{n^2}$
$a^n b^n c^n$
Turing-Decidable Languages
Recursive
$ww$
$a^n b a^m b^{n+m}$
$\mathcal{L}(CFG)$
$\mathcal{L}(PDA)$
$ww^R$
$\overline{a^n b^n}$
Dyck
Context-Free Languages
$a^n b^n$
$\mathcal{L}(DFA)$
$\#a(w) \equiv 2$
$\Sigma^*$
$\mathcal{L}(NFA)$
$\mathcal{L}(REX)$
$a^* b^*$
$1^{ax+b}$
$a\Sigma^* bb$
Regular Languages
$a^*$
finite languages
$\Sigma$
$\Sigma^7$
$\emptyset$