Instead, we will take a "reverse" approach to testing the emptiness of the grammar's language. We will check, for each nonterminal in the grammar, whether that nonterminal is able to generate some sequence of terminal symbols. If this is the case, then whenever that nonterminal appears in a derivation, we know that some sequence of terminal symbols will appear later in the derivation.

Our decision algorithm for E_{CFG} works a lot like our decision algorithm for E_{DFA}, except backwards: while the algorithm for E_{DFA} marked states starting from the initial state and leading to a final state, our algorithm for E_{CFG} will mark symbols starting from the terminal symbols and returning to the start nonterminal.

Theorem 11. *E_{CFG} is decidable.*

Proof. Construct a Turing machine $\mathcal{M}_{\text{ECFG}}$ that takes as input $\langle G \rangle$, where G is a context-free grammar, and performs the following steps:

1. Mark all terminal symbols in G.
2. Repeat until no new symbols are marked:
   (a) If G has a rule of the form $A \to \alpha_1 \dots \alpha_k$ and each symbol $\alpha_1, \dots, \alpha_k$ has already been marked, then mark the symbol A.
3. If the start nonterminal of G has not been marked, then accept. Otherwise, reject. □

1.3 Undecidable Problems for Context-Free Languages

Unlike the situation with the class of regular languages, the class of context-free languages has *just* enough expressive power to render certain decision problems undecidable. In order to prove that a decision problem is undecidable for a certain model, we typically need to use a special technique known as a *reduction*, which we will study in greater depth later. For now, we will simply present some undecidable problems for context-free languages and informally intuit why these problems can't be decided.

Universality Problem

When we established the decidability of the universality problem for regular languages, we relied on the fact that the class of regular languages was closed under complement, and this allowed us to use the decidability of E_{DFA} to decide U_{DFA}.

For context-free languages, we know that E_{CFG} is decidable, so we might be tempted to take a similar approach to show that U_{CFG} is decidable. However, we run into one big problem: the class of context-free languages is *not* closed under complement!

We saw earlier that, if a class of languages is closed under both union and intersection, then it must be closed under complement by De Morgan's laws. While it is true that context-free languages are closed under union, it is not true that they're also closed under intersection. We can reason about why this is the case with the following example:

- $L_1 = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^m \mid m, n \geq 0\}$ is context-free; and
- $L_2 = \{\mathtt{a}^m\mathtt{b}^n\mathtt{c}^n \mid m, n \geq 0\}$ is also context-free; but
- $L_1 \cap L_2 = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \geq 0\}$ is not context-free.

Thus, the class of context-free languages is not closed under intersection, and it is therefore not closed under complement. This explains why we can't use the same technique as we did for regular languages to show that U_{CFG} is decidable. However, it doesn't formally prove that U_{CFG} is undecidable; we will leave that proof for later.

Theorem 12. *U_{CFG} is undecidable.*

Equivalence Problem

Since we now know that the class of context-free languages is not closed under either intersection or complement, it seems hopeless to believe that the equivalence problem for context-free languages is decidable. Indeed, since we can test the equivalence of two languages by testing the emptiness of their symmetric difference, and since the symmetric difference is defined in terms of intersection and complement, we cannot use the same approach here as we used for the equivalence problem for regular languages.

Like before, this observation itself doesn't actually prove that EQ_{CFG} is undecidable; it only tells us that we can't use the same technique we used previously to prove the decidability of EQ_{DFA}. We will return to this result later so that we can prove it formally.

Theorem 13. *EQ_{CFG} is undecidable.*

1.4 Undecidable Problems for Turing Machines

We now move on to considering decision problems for Turing machines. But wait... for each of the previous two models, we started by considering decidable problems. Here, we jumped directly to undecidable problems. Did we accidentally skip over one section? No: as it turns out, Turing machines are simply so powerful that no common decision problem is decidable for this model.

But wait, since Turing machines are so powerful, doesn't that mean they can solve all kinds of problems? Indeed, that's true, but they can't *decide* many problems. Remember, in order to decide a problem, the Turing machine must always halt and either accept or reject the input word corresponding to the problem instance. The main issue encountered by the Turing machine is in the "halt" step, since there's no guarantee that the machine will halt on every input word it receives.

Indeed, even the most basic decision problems are rendered undecidable on a Turing machine, simply because of the fundamental limitation that the machine may get caught in an infinite loop during its computation.

Membership Problem

We've seen that the membership problems for regular languages and context-free languages are both decidable, by virtue of the fact that the models recognizing such classes of languages always halt and either accept and reject their input words. For Turing machines, however, we lose this valuable decidability property.

Before we continue, let's review the notions of decidability and semidecidability from the previous lecture. Suppose that $\mathcal{M}$ is a Turing machine recognizing a language L. Recall that:

- L is *decidable* if (i) whenever $w \in L$, $\mathcal{M}$ accepts w; and (ii) whenever $w \notin L$, $\mathcal{M}$ rejects w; and
- L is *semidecidable* if (i) whenever $w \in L$, $\mathcal{M}$ accepts w; and (ii) whenever $w \notin L$, $\mathcal{M}$ either rejects w or enters a loop.

It's quite straightforward to show that the membership problem for Turing machines, A_{TM}, is semidecidable.

Theorem 14. *A_{TM} is semidecidable.*

Proof. Construct a Turing machine $\mathcal{M}_{\text{ATM}}$ that takes as input $\langle \mathcal{M}, w \rangle$, where $\mathcal{M}$ is a Turing machine and w is a word, and performs the following steps:

1. Simulate $\mathcal{M}$ on input w.
2. (a) If $\mathcal{M}$ ever enters its accepting state q_{accept}, then accept.
   (b) If $\mathcal{M}$ ever enters its rejecting state q_{reject}, then reject. □

The machine $\mathcal{M}_{\text{ATM}}$ that we constructed in the proof of Theorem 14 looks quite similar to the earlier machine $\mathcal{M}_{\text{ADFA}}$ we constructed to decide A_{DFA}. However, $\mathcal{M}_{\text{ATM}}$ only semidecides A_{TM}, since the machine has no way of rejecting its input word if it gets caught in an infinite loop.

Note also that $\mathcal{M}_{\text{ATM}}$ is the canonical example of a universal Turing machine, since we can give it an encoding of any Turing machine and any input word, and it will simulate the computation of that Turing machine on that input word.

Now, we will prove that A_{TM} is undecidable; that is, it is semidecidable but *not decidable*. The technique we will use is essentially a proof by contradiction: we will assume that we have some machine capable of deciding A_{TM}, and then show that the existence of such a machine leads to some logical absurdity when we give a particular input word to that machine.

Theorem 15. *A_{TM} is undecidable.*

Proof. Assume by way of contradiction that A_{TM} is decidable, and suppose that $\mathcal{H}$ is a Turing machine that decides A_{TM}. Then, given an encoding of a Turing machine $\mathcal{M}$ and an input word w,

- $\mathcal{H}$ accepts $\langle \mathcal{M}, w \rangle$ if $\mathcal{M}$ accepts w; and
- $\mathcal{H}$ rejects $\langle \mathcal{M}, w \rangle$ if $\mathcal{M}$ rejects w.

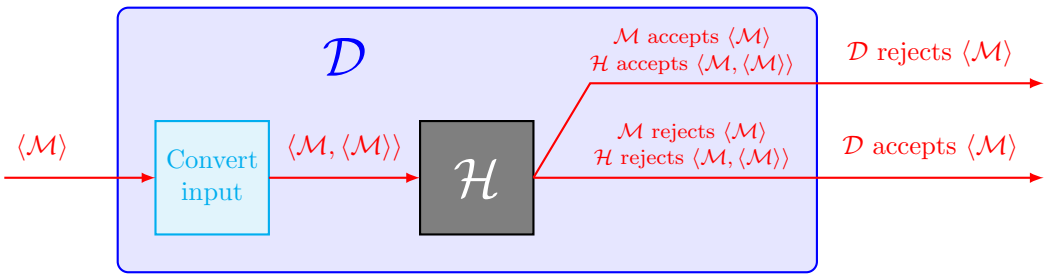
We now construct a new Turing machine $\mathcal{D}$ that relies on using $\mathcal{H}$ as an integral part of its computation. The machine $\mathcal{D}$ takes as input $\langle \mathcal{M} \rangle$, where $\mathcal{M}$ is a Turing machine, and performs the following steps:

1. Run $\mathcal{H}$ on input $\langle \mathcal{M}, \langle \mathcal{M} \rangle \rangle$.
2. (a) If $\mathcal{H}$ accepts, then reject.
   (b) If $\mathcal{H}$ rejects, then accept.

In other words, $\mathcal{D}$ gives as input to $\mathcal{H}$ an encoding of the Turing machine $\mathcal{M}$ and an encoded *description* of the Turing machine $\mathcal{M}$ as the input word to $\mathcal{M}$. Then, $\mathcal{D}$ returns the opposite of whatever $\mathcal{H}$ returns. That is,

- $\mathcal{D}$ accepts $\langle \mathcal{M} \rangle$ if $\mathcal{M}$ rejects $\langle \mathcal{M} \rangle$; and
- $\mathcal{D}$ rejects $\langle \mathcal{M} \rangle$ if $\mathcal{M}$ accepts $\langle \mathcal{M} \rangle$.

Diagrammatically, $\mathcal{D}$ functions in the following way. Observe that $\mathcal{H}$ is a "black box"; we don't know exactly how it performs its computation, we only assume that it can do so.



Now, consider what happens when we give $\mathcal{D}$ an encoding of *itself* as input. Suppose we give $\mathcal{D}$ the input $\langle \mathcal{D} \rangle$, so that $\mathcal{D}$ runs $\mathcal{H}$ on input $\langle \mathcal{D}, \langle \mathcal{D} \rangle \rangle$. Then, we get that

- $\mathcal{D}$ accepts $\langle \mathcal{D} \rangle$ if $\mathcal{D}$ rejects $\langle \mathcal{D} \rangle$; and
- $\mathcal{D}$ rejects $\langle \mathcal{D} \rangle$ if $\mathcal{D}$ accepts $\langle \mathcal{D} \rangle$.

In either case, $\mathcal{D}$ on input $\langle \mathcal{D} \rangle$ must return the opposite answer as the simulation of $\mathcal{D}$ on input $\langle \mathcal{D} \rangle$, which is of course impossible! As a result, we obtain a contradiction, and so no such machine $\mathcal{H}$ can exist to decide A_{TM}. □

The underlying concept that makes the proof of Theorem 15 work is *Cantor's diagonal argument*. In 1891, the German mathematician Georg Cantor showed that there exist uncountably infinite sets, or sets whose elements cannot be placed into a bijection with the (countably infinite) set of natural numbers.

The crux of the diagonal argument is that, if we take a set T of all binary words and assume that it is countably infinite, then we can enumerate all the elements of T. We can then construct a new element s that doesn't belong to T by setting the ith symbol of s to be the inverse of the ith symbol in the ith word; that is, if the ith symbol of the ith word is `0`, then we take the ith symbol of s to be `1`, and vice versa. In this way, s differs from every other word in T in at least one position. Since s is not an element of T, s could not have been accounted for in the enumeration of T, and so T must not be countably infinite.

The same diagonal argument can be applied to Turing machines in order to prove Theorem 15. If we could enumerate all Turing machines, then we could construct a table where each row corresponds to a Turing machine $\mathcal{M}_i$ and each column corresponds to an encoding of a Turing machine $\langle \mathcal{M}_j \rangle$. Then, the entries of the table could be taken to be the result of running $\mathcal{H}$ on input $\langle \mathcal{M}_i, \langle \mathcal{M}_j \rangle \rangle$.

	$\langle \mathcal{M}_1 \rangle$	$\langle \mathcal{M}_2 \rangle$	$\langle \mathcal{M}_3 \rangle$	$\langle \mathcal{M}_4 \rangle$	$\cdots$
$\mathcal{M}_1$	accept	accept	reject	reject	
$\mathcal{M}_2$	accept	reject	reject	accept	
$\mathcal{M}_3$	reject	reject	accept	accept	
$\mathcal{M}_4$	accept	accept	accept	reject	
$\vdots$					$\ddots$

By our assumption that $\mathcal{H}$ existed, and by the fact that we constructed $\mathcal{D}$ using $\mathcal{H}$, we know that $\mathcal{D}$ must appear in this table. We also know each entry in the row corresponding to $\mathcal{D}$, since the entry in column i is the opposite of the entry at index (i, i) of the table. However, this presents a problem: what is the entry corresponding to $\langle \mathcal{D}, \langle \mathcal{D} \rangle \rangle$? It somehow needs to be the opposite of itself, and we therefore arrive at the same contradiction as before.

	$\langle \mathcal{M}_1 \rangle$	$\langle \mathcal{M}_2 \rangle$	$\langle \mathcal{M}_3 \rangle$	$\langle \mathcal{M}_4 \rangle$	$\cdots$	$\langle \mathcal{D} \rangle$	$\cdots$
$\mathcal{M}_1$	accept	accept	reject	reject		accept	
$\mathcal{M}_2$	accept	reject	reject	accept		reject	
$\mathcal{M}_3$	reject	reject	accept	accept		reject	
$\mathcal{M}_4$	accept	accept	accept	reject		accept	
$\vdots$					$\ddots$		
$\mathcal{D}$	reject	accept	reject	accept		?	
$\vdots$							$\ddots$

Non-membership Problem

Now that we know the membership problem A_{TM} is semidecidable but not decidable, what can we say about the complement of the membership problem? Let us refer to this decision problem as the "non-membership problem", defined as follows:

$$\overline{A_{\mathsf{TM}}} = \{\langle \mathcal{M}, w \rangle \mid \mathcal{M} \text{ is a Turing machine that does not accept input word } w\}.$$

Instead of proving the decidability status of $\overline{A_{\mathsf{TM}}}$ directly, as we did in proving A_{TM} was undecidable, we will use an intermediate result to draw our conclusion about $\overline{A_{\mathsf{TM}}}$. The intermediate result we will use establishes a bicondition between the decidability of a language and the semidecidability of the same language.

Consider $\overline{L}$, the complement of a language L. If L is semidecidable, then we say that $\overline{L}$ is *co-semidecidable*. Essentially, this means that we can *semidecide* the *co*mplement of $\overline{L}$; this is just another way of saying that we can semidecide $\overline{\overline{L}} = L$. Since L and the language of all words *not* in $\overline{L}$ are the same language, we can use the Turing machine $\mathcal{M}$ semideciding L also to co-semidecide $\overline{L}$.

Note that $\overline{L}$ is co-semidecidable whenever L is semidecidable, but this does not necessarily imply that $\overline{L}$ is itself semidecidable. If $\overline{L}$ is semidecidable, then we naturally have that L is co-semidecidable by our definition. In the situation where L is both semidecidable and co-semidecidable, we obtain our characterization.

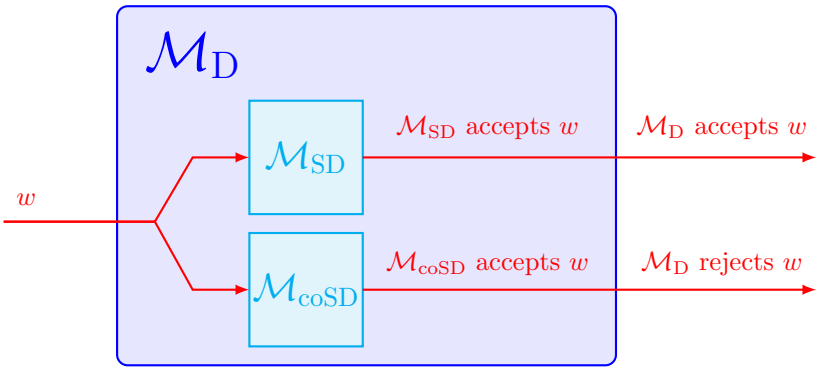
Theorem 16. *A language L is decidable if and only if L is both semidecidable and co-semidecidable.*

Proof. ($\Rightarrow$): Suppose L is decidable. Since all decidable languages are also semidecidable, we have that L is semidecidable. If we take the Turing machine deciding L and exchange the accepting and rejecting outputs, then we get a machine that decides $\overline{L}$. Thus, $\overline{L}$ is semidecidable, and so L is co-semidecidable.

($\Leftarrow$): Suppose L is both semidecidable and co-semidecidable. Then there exists a Turing machine $\mathcal{M}_{\text{SD}}$ semideciding L and a Turing machine $\mathcal{M}_{\text{coSD}}$ co-semideciding L. Using these two Turing machines, we can construct a Turing machine $\mathcal{M}_{\text{D}}$ that takes as input a word w and performs the following steps:

1. Run both $\mathcal{M}_{\text{SD}}$ and $\mathcal{M}_{\text{coSD}}$ on the input word w in parallel.
2. (a) If $\mathcal{M}_{\text{SD}}$ accepts, then accept.
   (b) If $\mathcal{M}_{\text{coSD}}$ accepts, then reject.

Diagrammatically, $\mathcal{M}_{\text{D}}$ functions in the following way.



Since every word w must belong to either L or $\overline{L}$, either $\mathcal{M}_{\text{SD}}$ or $\mathcal{M}_{\text{coSD}}$ must accept w. Moreover, since $\mathcal{M}_{\text{D}}$ halts whenever either $\mathcal{M}_{\text{SD}}$ or $\mathcal{M}_{\text{coSD}}$ accepts, we have that $\mathcal{M}_{\text{D}}$ decides L, and so L is decidable. □

Since Theorem 14 tells us that A_{TM} is semidecidable, we know that $\overline{A_{\mathsf{TM}}}$ is co-semidecidable. However, if $\overline{A_{\mathsf{TM}}}$ were not just co-semidecidable but also semidecidable, then we would have that A_{TM} is co-semidecidable. This ultimately leads us to a contradiction.

Corollary 17. *$\overline{A_{TM}}$ is not semidecidable.*

Proof. Assume by way of contradiction that $\overline{A_{\mathsf{TM}}}$ is semidecidable. Then A_{TM} would be co-semidecidable, and by Theorem 16, A_{TM} would be decidable, which contradicts Theorem 15. □

Therefore, we obtain a remarkable result: $\overline{A_{\mathsf{TM}}}$ is not just undecidable, it isn't even semidecidable! In the next lecture, we'll see a number of other examples of languages that are neither decidable nor semidecidable; in some cases, these languages are co-semidecidable, while in other cases, the languages lie outside of our language hierarchy entirely.

Speaking of our language hierarchy, we now have enough information to complete the "big picture" of the relationships between languages. We know by Corollary 10 that all context-free languages are also decidable. We also know by Theorem 16 that a language is decidable if and only if it is both semidecidable and co-semidecidable, so the class of decidable languages forms a subset positioned at the intersection between the classes of semidecidable and co-semidecidable languages.

Adding these new language classes to our hierarchy, we get the following diagram. Each of the decision problems we discussed in this lecture have been added to the appropriate class in this diagram, along with a number of other decision problems we will investigate further in the next lecture.