accepted by the Turing machine. But the inverse statement is not necessarily true: if the Turing machine's language is *nonempty*, then we can't conclude that the given word *is* accepted by the machine. The nonempty language could contain words that are not the given word.

Instead of the usual approach, then, we will do the following. We still receive as input $\langle \mathcal{M}, w \rangle$, where $\mathcal{M}$ is a Turing machine and w is an input word, but we will modify the description of $\mathcal{M}$ so that the only word it accepts is w. In doing so, we "bake in" w to the description of the Turing machine. We will then give the description of the modified machine $\mathcal{M}'$ to our machine that decides E_{TM}. In this way, testing the emptiness of $L(\mathcal{M}')$ is equivalent to testing whether $\mathcal{M}$ accepts w: $L(\mathcal{M}')$ is nonempty if and only if $w \in L(\mathcal{M})$.

Theorem 10. *E_{TM} is undecidable.*

Proof. Assume by way of contradiction that E_{TM} is decidable, and suppose that $\mathcal{M}_{\mathrm{ETM}}$ is a Turing machine that decides E_{TM}.

We construct a new Turing machine $\mathcal{M}_{\mathrm{A}}$ that decides the membership problem A_{TM}. The machine $\mathcal{M}_{\mathrm{A}}$ takes as input $\langle \mathcal{M}, w \rangle$, where $\mathcal{M}$ is a Turing machine and w is an input word, and performs the following steps:

1. Using the description of $\mathcal{M}$, construct the following Turing machine $\mathcal{M}'$ that takes as input x and performs the following steps:

   $\mathcal{M}'$1. If $x = w$, then simulate $\mathcal{M}$ on w.

   (a) If $\mathcal{M}$ accepts, then accept.

   $\mathcal{M}'$2. If $x \neq w$, then reject.

2. Run $\mathcal{M}_{\mathrm{ETM}}$ on input $\langle \mathcal{M}' \rangle$.
3. (a) If $\mathcal{M}_{\mathrm{ETM}}$ accepts, then reject.

   (b) If $\mathcal{M}_{\mathrm{ETM}}$ rejects, then accept.

Therefore, if such a machine $\mathcal{M}_{\mathrm{ETM}}$ existed to decide E_{TM}, then we could decide A_{TM} as well. However, we know that A_{TM} is undecidable. Thus, $\mathcal{M}_{\mathrm{ETM}}$ must not exist, and so E_{TM} must be undecidable. □

Now that we know E_{TM} is undecidable, is the problem at least semidecidable? Surprisingly, no! Remember that E_{TM} asks whether a given Turing machine $\mathcal{M}$ accepts *no* input words. In order to positively semidecide this property (i.e., get a "yes" answer), we would need to check that every possible input word over the alphabet Σ is *not* accepted by $\mathcal{M}$. Since there are infinitely many words over Σ, we quickly end up in an infinite loop.

On the other hand, the complementary problem $\overline{E_{\mathsf{TM}}}$ is semidecidable, since every Turing machine with a nonempty language must accept at least one input word. Indeed, we can reduce from A_{TM} to $\overline{E_{\mathsf{TM}}}$, so the procedure for semideciding $\overline{E_{\mathsf{TM}}}$ is similar to that for semideciding A_{TM}. As a consequence, we get the following result.

Theorem 11. *E_{TM} is co-semidecidable.*

Proof Sketch. The non-emptiness problem for Turing machines, $\overline{E_{\mathsf{TM}}}$, is semidecidable. As a result, E_{TM} is co-semidecidable. □

2.3 Universality Problem

Moving on to the universality problem for Turing machines, U_{TM}, we obtain the same outcome as we had for E_{TM}. Indeed, the proof of undecidability for the universality problem is almost identical to that for the emptiness problem; we just need to swap accepting and rejecting outcomes in the last step of the computation of $\mathcal{M}_{\mathrm{A}}$, since the universality problem is, in a sense, the "opposite" of the emptiness problem.

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

Proof. Assume by way of contradiction that U_{TM} is decidable, and suppose that $\mathcal{M}_{\mathrm{UTM}}$ is a Turing machine that decides U_{TM}.

We construct a new Turing machine $\mathcal{M}_{\mathrm{A}}$ that decides the membership problem A_{TM}. The machine $\mathcal{M}_{\mathrm{A}}$ takes as input $\langle \mathcal{M}, w \rangle$, where $\mathcal{M}$ is a Turing machine and w is an input word, and performs the following steps:

1. Using the description of $\mathcal{M}$, construct the following Turing machine $\mathcal{M}'$ that takes as input x and performs the following steps:

   $\mathcal{M}'$1. If $x = w$, then simulate $\mathcal{M}$ on w.

      (a) If $\mathcal{M}$ accepts, then accept.

      (b) If $\mathcal{M}$ rejects, then reject.

   $\mathcal{M}'$2. If $x \neq w$, then reject.

2. Run $\mathcal{M}_{\mathrm{UTM}}$ on input $\langle \mathcal{M}' \rangle$.
3. (a) If $\mathcal{M}_{\mathrm{UTM}}$ accepts, then accept.

   (b) If $\mathcal{M}_{\mathrm{UTM}}$ rejects, then reject.

Therefore, if such a machine $\mathcal{M}_{\mathrm{UTM}}$ existed to decide U_{TM}, then we could decide A_{TM} as well. However, we know that A_{TM} is undecidable. Thus, $\mathcal{M}_{\mathrm{UTM}}$ must not exist, and so U_{TM} must be undecidable. □

Unlike E_{TM}, however, we cannot prove that U_{TM} is co-semidecidable. In fact, U_{TM} is neither semidecidable nor co-semidecidable; it lies entirely outside of our language hierarchy! The idea behind the proof involves a reduction from another decision problem about *total machines*, or Turing machines that halt on all inputs:

$$T_{\mathsf{TM}} = \{\langle \mathcal{M} \rangle \mid \mathcal{M} \text{ is a Turing machine that halts on all input words}\}.$$

By analogy, if $HALT_{\mathsf{TM}}$ is the "halting" version of A_{TM}, then T_{TM} is the "halting" version of U_{TM}. Observe that, since total machines halt on all inputs, such machines decide their associated languages.

While we won't go through the complete proofs here, we can construct two reductions: $HALT_{\mathsf{TM}} \leq_m T_{\mathsf{TM}}$ and $HALT_{\mathsf{TM}} \leq_m \overline{T_{\mathsf{TM}}}$. As a consequence of the first reduction, and by the fact that mapping reductions are closed under complement, we know that $\overline{HALT_{\mathsf{TM}}} \leq_m \overline{T_{\mathsf{TM}}}$, which implies that $\overline{T_{\mathsf{TM}}}$ is not semidecidable. At the same time, by the second reduction and the same fact, we know that $\overline{HALT_{\mathsf{TM}}} \leq_m T_{\mathsf{TM}}$, which similarly implies that T_{TM} is not semidecidable.

Using a reduction from the decision problem T_{TM}, we obtain our negative semidecidability results for U_{TM}.

Theorem 13. *U_{TM} is neither semidecidable nor co-semidecidable.*

Proof Sketch. We can construct a reduction $T_{\mathsf{TM}} \leq_m U_{\mathsf{TM}}$. By the fact that mapping reductions are closed under complement, we know that $\overline{T_{\mathsf{TM}}} \leq_m \overline{U_{\mathsf{TM}}}$.

Since neither T_{TM} nor $\overline{T_{\mathsf{TM}}}$ are semidecidable, we have that neither U_{TM} nor $\overline{U_{\mathsf{TM}}}$ are semidecidable. Saying that $\overline{U_{\mathsf{TM}}}$ is not semidecidable is equivalent to saying that U_{TM} is not co-semidecidable. □

2.4 Equivalence Problem

Recall that the equivalence problem for Turing machines asks whether the languages of two Turing machines are equivalent; that is, no word belongs to one language but not the other.

In each of our previous undecidability proofs, we reduced from A_{TM} to the given problem. We did this mostly because of the fact that A_{TM} was our "first" undecidable problem, and because it was easy for us to connect the membership problem to other familiar decision problems. For the equivalence problem, on the other hand, we don't need to restrict ourselves to A_{TM}; we can reduce from a different decision problem.

Think back to the definition of the emptiness problem for Turing machines: if $\langle \mathcal{M} \rangle \in E_{\mathsf{TM}}$, then $L(\mathcal{M}) = \emptyset$. The emptiness problem is just the equivalence problem in disguise, where one of the languages is the empty language! Therefore, if we fix one of the Turing machines to accept no words, we can reduce the problem of testing emptiness to the problem of testing equivalence, and we can in turn obtain our undecidability result.

Theorem 14. *EQ_{TM} is undecidable.*

Proof. Assume by way of contradiction that EQ_{TM} is decidable, and suppose that $\mathcal{M}_{\text{EQTM}}$ is a Turing machine that decides EQ_{TM}.

We construct a new Turing machine $\mathcal{M}_{\text{ETM}}$ that decides the emptiness problem E_{TM}. The machine $\mathcal{M}_{\text{ETM}}$ takes as input $\langle \mathcal{M} \rangle$, where $\mathcal{M}$ is a Turing machine, and performs the following steps:

1. Run $\mathcal{M}_{\text{EQTM}}$ on input $\langle \mathcal{M}, \mathcal{M}_{\emptyset} \rangle$, where $\mathcal{M}_{\emptyset}$ is a Turing machine that accepts no input words.
2. (a) If $\mathcal{M}_{\text{EQTM}}$ accepts, then accept.
   (b) If $\mathcal{M}_{\text{EQTM}}$ rejects, then reject.

Therefore, if such a machine $\mathcal{M}_{\text{EQTM}}$ existed to decide EQ_{TM}, then we could decide E_{TM} as well. However, we know that E_{TM} is undecidable. Thus, $\mathcal{M}_{\text{EQTM}}$ must not exist, and so EQ_{TM} must be undecidable. □

Similar to the universality problem, the equivalence problem for Turing machines is neither semidecidable nor co-semidecidable, meaning yet another decision problem lies entirely outside of our language hierarchy. To prove this result, we reduce from U_{TM} instead of E_{TM} as we did in our previous proof. We are able to construct this reduction since the universality problem asks whether $L(\mathcal{M}) = \Sigma^*$ for some Turing machine $\mathcal{M}$, meaning that it too is just the equivalence problem in disguise. Since we know that U_{TM} is neither semidecidable nor co-semidecidable, the same must be true for EQ_{TM}.

Theorem 15. *EQ_{TM} is neither semidecidable nor co-semidecidable.*

Proof Sketch. We can construct a reduction $U_{\mathsf{TM}} \leq_m EQ_{\mathsf{TM}}$. Since U_{TM} is neither semidecidable nor co-semidecidable by Theorem 13, we get the same result for EQ_{TM}. □

3 Undecidable Problems for Context-Free Languages (Redux)

Recall that, previously, we noted both U_{CFG} and EQ_{CFG} were undecidable. However, we didn't prove either of those claims, mainly because we didn't have the tools to do so back then. Here, let's wrap up our work by presenting both of these proofs. Before we do so, however, we require one notion relating to the computation of a Turing machine.

Recall that the *configuration* of a Turing machine is a representation of the current state, tape contents, and input head position of the Turing machine at some point in the computation. In essence, a configuration is a "snapshot" of the Turing machine mid-computation. Depending on the current state of the machine, a configuration may be a *start configuration* (if the current state is q_0), an *accepting configuration* (if the current state is q_{accept}), or a *rejecting configuration* (if the current state is q_{reject}).

If we consider the entire sequence of configurations of a Turing machine from start to finish—that is, from the start configuration to either an accepting or rejecting configuration—we get a complete picture of the Turing machine's computation. This sequence of configurations is known as a *computation history.*

Definition 16 (Computation history)**.** Given a Turing machine $\mathcal{M}$ and an input word w, a computation history for $\mathcal{M}$ on w is a sequence of configurations $C_1, C_2, \ldots, C_m$, where C_1 is the start configuration of $\mathcal{M}$ on w, C_m is either an accepting configuration or a rejecting configuration, and each configuration C_i yields the following configuration C_{i+1}.

- If C_m is an accepting configuration, then the sequence forms an accepting computation history.
- If C_m is a rejecting configuration, then the sequence forms a rejecting computation history.

Note that a computation history for a Turing machine $\mathcal{M}$ on an input word w only exists when $\mathcal{M}$ halts on w. As a result, the sequence of configurations $C_1, C_2, \ldots, C_m$ always has a finite number of elements. Deterministic computations always have exactly one computation history per input word, while nondeterministic computations may have multiple computation histories per input word.

Why do we require this notion of computation histories in order to prove that our remaining context-free problems are undecidable? As it turns out, we can apply the idea of reductions to the computation histories of Turing machines in order to establish undecidability results. We will use these *reductions via computation histories* to prove the undecidability of one of our context-free decision problems.

3.1 Universality Problem

To prove the undecidability of U_{CFG}, we will construct a reduction from A_{TM} to U_{CFG} that makes use of computation histories. Our reduction will require us to make two slight changes: first, as part of the reduction, we will be constructing a context-free model of computation rather than a Turing machine; and second, we must format the computation history in a way that allows us to process it correctly. We will see how both of these changes are handled as we work through the proof.

Theorem 17. *U_{CFG} is undecidable.*

Proof. Assume by way of contradiction that U_{CFG} is decidable. We will show how to use the decision algorithm for U_{CFG} to decide A_{TM}.

Given a Turing machine $\mathcal{M}$ and an input word w, we construct a context-free grammar G that generates all words if and only if $\mathcal{M}$ does not accept its input word w. Specifically, we construct G in such a way that the words it generates correspond to non-accepting computation histories of $\mathcal{M}$ on w. In this way, $\mathcal{M}$ accepts w if and only if G does not generate the accepting computation history of $\mathcal{M}$ on w, meaning that the language of G is not universal.

We will assume that the computation history of $\mathcal{M}$ is written in the form $\texttt{\#}C_1\texttt{\#}C_2^{\mathrm{R}}\texttt{\#}C_3\texttt{\#}C_4^{\mathrm{R}}\texttt{\#}\ldots\texttt{\#}C_m\texttt{\#}$, where C_i is the ith configuration of $\mathcal{M}$ and # is a special boundary marker. Note that every even-numbered configuration is written in reverse; this is the aforementioned "format" change we require in order to process the computation history correctly.

Working from Definition 16, we can deduce that a computation history is non-accepting if one of the following conditions is met:

1. The computation history does not start with the start configuration as C_1;
2. The computation history does not end with an accepting configuration as C_m; or
3. Some configuration C_i does not yield the following configuration C_{i+1} according to the transition function of $\mathcal{M}$.

To construct the context-free grammar G, we will construct a pushdown automaton $\mathcal{A}$ and use our conversion process to turn it into a grammar. The pushdown automaton $\mathcal{A}$ checks each of the three non-accepting conditions, and it does so by nondeterministically guessing which condition it checks.

- In one nondeterministic branch, $\mathcal{A}$ checks the first condition by reading the beginning of its input word and accepting if the segment between the first two boundary markers, C_1, is not the start configuration.
- In another nondeterministic branch, $\mathcal{A}$ checks the second condition by reading the end of its input word and accepting if the segment between the last two boundary markers, C_m, is not an accepting configuration.
- In the last nondeterministic branch, $\mathcal{A}$ checks the third condition by scanning the input word until it nondeterministically selects a configuration C_i. $\mathcal{A}$ then pushes the symbols of C_i to its stack until it

reads a boundary symbol, and pops each symbol of C_i as it reads the corresponding symbol of C_{i+1} from the input word. (We can match corresponding symbols in the correct order as a result of the "format" change from earlier.) If there is any difference between symbols that was not produced by the transition function of $\mathcal{M}$, $\mathcal{A}$ accepts.

Clearly, every word accepted by the pushdown automaton $\mathcal{A}$ corresponds to a non-accepting computation history of $\mathcal{M}$, and so the language of the context-free grammar G consists of the same non-accepting computation histories.

Therefore, if it were possible to construct such a grammar G to decide U_{CFG}, then we could decide A_{TM} as well. However, we know that A_{TM} is undecidable. Thus, G must not exist, and so U_{CFG} must be undecidable. □

3.2 Equivalence Problem

Finally, we come to the equivalence problem for context-free grammars, EQ_{CFG}. As we did with EQ_{TM}, we will use the observation that another undecidable problem for context-free grammars—namely, U_{CFG}—is really the equivalence problem in disguise, and we will reduce from that problem to EQ_{CFG}.

Theorem 18. *EQ_{CFG} is undecidable.*

Proof. Assume by way of contradiction that EQ_{CFG} is decidable, and suppose that $\mathcal{M}_{\text{EQCFG}}$ is a Turing machine that decides EQ_{CFG}.

We construct a new Turing machine $\mathcal{M}_{\text{UCFG}}$ that decides the universality problem U_{CFG}. The machine $\mathcal{M}_{\text{UCFG}}$ takes as input $\langle G \rangle$, where G is a context-free grammar, and performs the following steps:

1. Run $\mathcal{M}_{\text{EQCFG}}$ on input $\langle G, H \rangle$, where H is a context-free grammar such that $L(H) = \Sigma^*$.
2. (a) If $\mathcal{M}_{\text{EQCFG}}$ accepts, then accept.
   (b) If $\mathcal{M}_{\text{EQCFG}}$ rejects, then reject.

Therefore, if such a machine $\mathcal{M}_{\text{EQCFG}}$ existed to decide EQ_{CFG}, then we could decide U_{CFG} as well. However, we know that U_{CFG} is undecidable. Thus, $\mathcal{M}_{\text{EQCFG}}$ must not exist, and so EQ_{CFG} must be undecidable. □