Computation Histories

1. Some very interesting proofs of undecidability rely on the technique of constructing a language that describes the possible computations of a TM on one or more inputs.

2. Recall that a TM configuration is a triple \((q,u,v)\) with \(q \in Q\) representing the current state of the control, \(u \in \Gamma^*\) representing the contents of the tape to the left of the current head position, and \(v \in \Gamma^*\) representing the tape contents from the head to the right end of the non-blank tape.

3. We can use a sequence of strings that describe configurations to describe the complete computation of a TM.

**Definition:** Given a TM \(M = (Q,\Sigma,\Gamma,\delta,q_0,q_{\text{accept}},q_{\text{reject}})\) and a string \(w \in \Sigma^*\), we define the language of accepting computation histories of \(M\) on \(w\) as

\[
L_{\text{Computation-history}}(M,w) = \{w_0w_1...w_n \mid \text{each } w_i \text{ is a configuration for } M \backslash w_0 \text{ is the initial configuration for } w \backslash w_n \text{ is a final/accept configuration each } w_i \text{ yields } w_{i+1} \text{ according to } \delta \}
\]

4. In general, \(L_{\text{Computation-history}}(M,w)\) is a pretty complicated language. It is decidable, but it is not regular or context-free.

5. Interestingly, if \(M\) is deterministic, then \(L_{\text{Computation-history}}(M,w)\) is actually a very simple language. It is either equal to \(\emptyset\) if \(w \notin L(M)\) or it contains a single string if \(w \in M\).

- This is interesting because the empty language and any language containing just 1 (or any finite number) of strings must be regular and therefore context-free.
- If given the description of \(M\) and \(w\) we could build a DFA that accepted \(L_{\text{Computation-history}}(M,w)\) then since emptiness of a regular language is decidable, we could decide whether \(w \in L(M)\). This question is better known as \(A_{TM}\) and we know that it is undecidable!
- This implies that even though \(L_{\text{Computation-history}}(M,w)\) must be regular if \(M\) is deterministic, there is no algorithm to find a DFA for the language given a description of \(M\). The DFA is uncomputable!

6. To get a bit closer to being able to find a way to compute \(L_{\text{Computation-history}}(M,w)\), we can define a slight variation of this language:

**Definition:** Given a TM \(M = (Q,\Sigma,\Gamma,\delta,q_0,q_{\text{accept}},q_{\text{reject}})\) and a string \(w \in \Sigma^*\), we define the language of reversed computation histories of \(M\) on \(w\) as

\[
L_{R_{\text{Computation-history}}}(M,w) = \{w_0w_1^R...w_n \mid \text{each } w_i \text{ is a configuration for } M \backslash w_0 \text{ is the initial configuration for } w \backslash w_n \text{ is a final/accept configuration each } w_i \text{ yields } w_{i+1} \text{ according to } \delta \text{ every other } w_i \text{ is written backwards } \}
\]

7. Again, if \(M\) is deterministic, this language is finite and therefore both regular and context-free.

8. You should imagine that it would be possible to construct a CFG or a PDA that ensures that all even-odd pairs of strings or all odd-even pairs of configurations in such strings satisfied the “yields” requirement, the PDA or CFG can only ensure that every other configuration yields its successor.
9. Somewhat amazingly the complement of \( L^R_{\text{computation-history}}(M, w) \) which is far from finite is context-free and we can effectively construct a PDF or CFG that describes it given a description of \( M \) and \( w \).

- The trick is that we can describe a PDA that uses its nondeterminism to guess where the feature that would make a string invalid as a computation history occurs and then use the stack of the PDA to validate the guesses.
- The PDA guesses one of the following issues:
  - The whole string is not formatted properly (this is regular),
  - For any pair of consecutive configurations, the earlier configuration does not yield the following configuration (a PDA could nondeterministically guess which pair didn’t match, push the first configuration on its stack and then verify the mismatch as it read the next configuration),
  - The first configuration is not a valid initial configuration for \( w \) (again regular),
  - The last configuration is not an accepting configuration (again regular).

10. Using this fact, we can show that \( \overline{ALL_{CFG}} \) is not recognizable by reducing \( \overline{A_{TM}} \) to \( ALL_{CFG} \). To do this, we will assume the existence of a TM \( M_{\overline{ALL_{CFG}}} \) that recognizes \( ALL_{CFG} \) and build a TM \( M_{\overline{A_{TM}}} \) that recognizes \( \overline{A_{TM}} \).

- \( M_{\overline{A_{TM}}} \) should accept its input if it is not a valid encoding of a TM description and input.
- Next, \( M_{\overline{A_{TM}}} \) should create a CFG \( G_{\overline{M}} \) for \( L^R_{\text{computation-history}}(M, w) \).
- Finally, \( M_{\overline{A_{TM}}} \) should run \( M_{\overline{ALL_{CFG}}} \) on \( \langle G_{\overline{M}} \rangle \) and accept if \( M_{\overline{ALL_{CFG}}} \) accepts.
- This machine recognizes \( \overline{A_{TM}} \) because \( M \) has an accepting computation history on \( w \) iff \( w \in L(M) \). As a result, \( L^R_{\text{computation-history}}(M, w) = L(G_{\overline{M}}) = \Sigma^* \) exactly when \( w \notin L(M) \).
- Since it would be a contradiction if we could recognize \( \overline{A_{TM}} \), it must be impossible to recognize \( ALL_{CFG} \).

11. Since we can recognize \( \overline{ALL_{CFG}} \) by simply checking all strings looking for one that cannot be derived from the grammar, this implies that \( \overline{ALL_{CFG}} \) is recognizable but not decidable.

Just about Everything about TMs is Undecidable

1. It turns out that just about every interesting question about Turing machines is undecidable. One consequence of this is that just about half of all interesting questions are not recognizable (because if both a language and its complement are recognizable then both languages are decidable).

2. With this in mind, it should be easy to find some more languages to practice our reduction proof skills.

3. Here are just a few possible examples:

- \( REVERSIBLE_{TM} = \{ \langle M \rangle \mid w \in L(M) \text{ iff } w^R \in L(M) \} \)
- \( REGULAR_{TM} = \{ \langle M \rangle \mid L(M) \text{ is regular} \} \)
- \( DISJOINT_{TM} = \{ \langle M, N \rangle \mid L(M) \cup L(N) = \emptyset \} \)
- \( PRIME_{TM} = \{ \langle M \rangle \mid w \in L(M) \Rightarrow |w| \text{ is prime} \} \)

4. To give us one more bit of practice with this, let’s show that \( REVERSIBLE_{TM} \) is not recognizable.

- First, observe that \( \{ab\} \) and \( \{a^n b^n \mid n \geq 0, \text{ are nice examples of sets that are not reversible while } \emptyset \text{ and } \Sigma^* \text{ are definitely reversible.} \)
- We have seen several proofs in which the machine \( M' \) we built given an input \( \langle M, w \rangle \) accepted all inputs if \( M \) accepted \( w \) and rejected all inputs (possible by looping) otherwise.
- It is easy to refine this machine by running another Turing machine as a sub-module after \( M' \) determines whether \( M \) accepts \( w \). For example, we could only accept strings of the form \( a^n b^n \) if \( M \) accepts \( w \).
• This results in $M' \in \text{REVERSIBLE}_{TM}$ exactly when $w \notin L(M)$.

• This gives us what we need to give a proof that $\text{REVERSIBLE}_{TM}$ is not recognizable:

**Proof:** Assume that $\text{REVERSIBLE}_{TM}$ is recognized by some Turing machine $M_{RTM}$. Construct a machine $M_{TM}$ that operates as follows:

(a) If the machine’s input is not a valid encoding of a Turing machine and its input, accept.

(b) Otherwise,
   - construct a description of a new machine $M'$ which behaves as follows:
     * On input $w'$, simulate $M$ on $w$. If $M$ accepts $w$, then accept $w'$ if it is a string of the form $a^n b^n$.
     * Otherwise, reject.
   - Run $M_{RTM}$ on $\langle M' \rangle$ and accept if it does.

The machine $M_{TM}$ will accept $\langle M, w \rangle$ exactly when $w \notin L(M)$ since in this case, $L(M') = \emptyset$ is reversible while otherwise $L(M') = a^n b^n$ which is not reversible.

5. It is just as easy to show that $\text{REVERSIBLE}_{TM}$ is not recognizable.

6. We will stick with $a^n b^n$ as our non-regular language. We will switch from $\emptyset$ to $\Sigma^*$ for our reversible language.

### Mapping Reductions

1. We have observed that many of the proofs of undecidability and non-recognizability we have explored have a very similar structure.

2. We can formalize these similarities in the notion of mapping reducibility and then use this idea to “simplify” the proofs for many results involving decidability and recognizability.

3. First we must define the idea of a computable function.

**Definition:** A function $f$ from $\Sigma^* \rightarrow \Sigma^*$ is **computable** if and only if some Turing machine $M$ on every input $w$, halts with $f(w)$ on its tape.

4. This definition is mainly an admission that Turing machines can do interesting things other than just accept and reject.

   • This is not new. One of the first TMs we considered implemented a computable function. It took input strings and did its best to insert a # in the middle of them.

   • In each of the non-recognizability proofs we have given, we have embedded such a computable function. Namely, the computation that generated $\langle M' \rangle$ given some $\langle M, w \rangle$.

5. Given the notion of computable functions, we can capture the essence of what our $M$’s are really about.

**Definition:** Language $A$ is many-to-one reducible to language $B$ (written $A \leq_m B$) if there exists a computable function $f : \Sigma^* \rightarrow \Sigma^*$ such that for every $w \in \Sigma^*$

$$w \in A \iff f(w) \in B.$$ 

In this case we call $f$ a reduction.

6. Let me share two handy memory aids for dealing with the notation.

   (a) The point of the $\leq$ goes in the direction opposite the function arrow.

   (b) It helps to read the $\leq$ as “is easier than” rather than “is less than”.

7. The following diagram illustrates what this definition requires and allows.

   • It must map members of $A$ to members of $B$.

   • It must map strings that are not in $A$ to strings that are not in $B$. 

3
• It can map multiple input strings to the same output.

8. As a simple example, the computable function
\[ f(<M,w>) = <M'> \quad \text{where } M' \text{ is a TM that ignores its input, runs } M \text{ on } w \text{ and accepts its input if } M \text{ accepts } w. \]

shows that \( A_{TM} \leq_m A_{TM} \text{ALL}_{TM} \) since it maps any \(<M,w>\) that belongs in \( A_{TM} \) to an \(<M'>\) that belongs in \( A_{TM} \text{ALL}_{TM} \).

9. Note that there is nothing that inherently ties this computable function to \( A_{TM} \) and \( A_{TM} \text{ALL}_{TM} \). As long as we can identify two languages \( A \) and \( B \) such that \( w \in A \iff f(w) \in B \), \( A \leq_m B \). As an easy example of this, note that this function also shows that \( A_{TM} \leq_m \overline{A_{TM}} \text{ALL}_{TM} \). In general \( A \leq_m B \iff \overline{A} \leq_m \overline{B} \).

10. As a more diverse example, this function also shows that \( \overline{A_{TM}} \leq_m E_{TM} \) since it maps any \(<M,w>\) that belongs in \( \overline{A_{TM}} \) to an \(<M'>\) that belongs in \( E_{TM} \).

11. With these definitions, we can succinctly formalize the technique we have been using in all our proofs for the last few classes:

**Theorem:** If \( A \leq_m B \) and ...

(a) \( B \) is decidable, then \( A \) is decidable,

(b) \( A \) is undecidable, then \( B \) is undecidable,

(c) \( B \) is recognizable, then \( A \) is recognizable, and

(d) \( A \) is not recognizable, then \( B \) is not recognizable

• We won’t give a detailed proof of these claims, but they are all just obvious applications of the proof techniques we have been employing.

12. As an example of how we might use such a mapping reduction, consider the language
\[ \text{DISJOINT}_{TM} = \{ \langle M,M \rangle \mid M \& N \text{ are TMs and } L(M) \cap L(N) = \emptyset \} \]

• We will show that \( \text{DISJOINT}_{TM} \) is not recognizable by showing that \( E_{TM} \leq_m \text{DISJOINT}_{TM} \).

To do this, we need a mapping that will take any \( \langle M \rangle \) to a pair of TM descriptions \( \langle N,N' \rangle \) in such a way that \( L(N) \) and \( L(N') \) are disjoint if and only if \( L(M) \) is empty.

• Let \( \text{ACCEPT} \) be a TM that accepts all strings.

• Consider the function \( f(<M>) = <M, \text{ACCEPT}> \).

• This is clearly computable.

• It is also clear that \( <M, \text{ACCEPT}> \in C \) if and only if \( <E> \in E_{TM} \).

• Given that we know that \( E_{TM} \) is not recognizable, we can conclude that \( \text{DISJOINT}_{TM} \) is not recognizable.

**Rice’s Theorem**

1. While we have categorized a large number of languages as decidable, recognizable, or not recognizable, there are still plenty of additional examples we could consider:

• \( \text{REGULAR}_{TM} = \{ \langle M \rangle \mid L(M) \text{ is regular } \} \)

• \( \text{CONTEXT-FREE}_{TM} = \{ \langle M \rangle \mid L(M) \text{ is context-free } \} \)

• \( \text{PALINDROME}_{TM} = \{ \langle M \rangle \mid w \in L(M) \iff w^R \in L(M) \} \)
2. There is a single theorem that will quickly allow us to show that all of the languages listed above are undecidable.

   • This result is known as Rice’s Theorem.

3. Informally, Rice’s Theorem says that any nontrivial property of a Turing machine’s language is undecidable.

   • Nontrivial means that the languages of some but not all TMs have this property.
   • The fact that it is a property of the language rather than the TMs means that it must be based strictly on the set of strings a given TM accepts rather than on how the TM is designed or operates.

4. We can formalize this notion as:

   **Rice’s Theorem.** Suppose that $L$ is a language with

   \[
   \emptyset \subset L \subset \{ \langle M \rangle \mid \langle M \rangle \text{ is a valid Turing machine} \}
   \]

   such that if $L(M) = L(N)$ then $\langle M \rangle \in L \iff \langle N \rangle \in L$

   then $L$ is undecidable.