

CSCI 341–Fall 2024: Lecture Notes

Set 16: Reductions

Edward Talmage

December 18, 2024

We want to be able to prove that other languages are undecidable, but the undecidability proof for A_{TM} was not something that we can replicate easily! It involved at least two seemingly arbitrary choices that worked out to give what we needed, though it was not obvious beforehand that they would. We would like some more concrete, generally useful tools. Reduction is that tool. The idea is that we can transform one problem into another, relating the solvability of a new problem to the solvability of an understood problem.

Definition 1. A *mapping reduction* (a.k.a. many-one reduction) is a terminating algorithm that transforms a string in language A to a string in language B and a string not in language A to one not in language B . If such an algorithm exists, we say that $A \leq_m B$.

- Formally, a mapping reduction must leave its output (a string in B) on the tape and nothing else.

Example: Let $\Sigma_A = \{00\}$ and language A over Σ_A be the set $\{(00)^x \mid x = 2^n, n \geq 0\}$. Let $\Sigma_B = \{0\}$ and language B over Σ_B be the set $\{0^x \mid x = p^k, p \text{ prime}, k \geq 0\}$.

We can show that $A \leq_m B$, by giving a mapping reduction. Specifically, $M((00)^x) = (0 \circ 0)^x$ is a reduction from A to B , as any string in A will map to a string in B (length a power of a prime), any string not in A will map to a string not in B (length divisible by 2, not a power of 2).

Example: Let $\Sigma = \{0, 1\}$ and $A = \{w \mid |w| \text{ is odd}\}$. We can reduce this to the set $B = \{w \mid |w| \text{ is even}\}$ by appending a single 0 to each string. That is, $f(x) = x0$. Now, if $x \in A$ ($|x|$ was odd), $|x0|$ is even, so $x0 \in B$. If $x \notin A$, then its length was even, so $|x0|$ is odd and $x0 \notin B$.

Note that it is easy to come up with “half” of a reduction, where strings in the first set map to strings in the second set, but it is tricky to keep strings not in the first set from mapping into the second set.

Usage: More generally, a reduction transforms one problem into another, because languages represent problems we want to solve. We can then conclude the following, where A, B are languages s.t. there is a reduction from A to B ($A \leq_m B$):

Lemma 1. *If B is decidable, then A is decidable.*

- *If A is undecidable, then B is undecidable.*
- If B is recognizable, then A is recognizable.*
- *If A is unrecognizable, then B is unrecognizable.*

Phrased differently: If A reduces to B and B is easy, then A is also easy, as we just built a solution. If A is hard, then B must also be hard, or we would have just constructed an easy solution contradicting A 's hardness.

1 Examples

1.1 Halting Problem

Given a TM M and a string w , does M halt on input w ? As a language,

$$HALT_{TM} = \{\langle M, w \rangle \mid M \text{ a TM, } w \text{ a string, } M(w) \text{ halts}\}$$

Claim 1. $HALT_{TM}$ is undecidable.

Proof. Assume in contradiction that $HALT_{TM}$ is decidable, by decider H . We construct a TM to decide A_{TM} :

On input $\langle M, w \rangle$, where M is a TM, w a string:

1. Run $H(\langle M, w \rangle)$.
2. If H rejects, reject
3. If H accepts, run $M(w)$ until it halts.
4. If M accepted, accept. Else, reject.

This accepts iff $w \in L(M)$, so it decides the acceptance problem A_{TM} . We know that A_{TM} is undecidable, so this machine cannot exist, contradicting the existence of H . \square

1.2 Emptiness Problem

Given a TM M , is its language empty?

$$E_{TM} = \{\langle M \rangle \mid M \text{ is a TM, } L(M) = \emptyset\}$$

Claim 2. E_{TM} is undecidable.

Red Herring: Assume E is a decider for E_{TM} . Given $\langle M, w \rangle$, run E on M to decide A_{TM} . But that just tells us that M accepts some string, nothing about the specific w , so we have not solved A_{TM} .

Proof. Assume in contradiction that TM E decides E_{TM} . Show that we can decide A_{TM} :

On input $\langle M, w \rangle$, where M is a TM, w a string:

1. Write code for a new machine M' based on M :

On input x :

- Check to see if $x = w$. If so, run $M(w)$ and follow its decision. (that is, $M'(w) = M(w)$)
- If $x \neq w$, reject ($M'(x \neq w) = reject$)

2. Run $E(\langle M' \rangle)$.

3. If E accepted, reject. If E rejected, accept.

First, note that constructing M' is finite, since we just add a chain of states of length $|w|$ to check the input and reject if it is not w , then run M . Then, consider the behavior of $E(\langle M' \rangle)$: If $w \in L(M)$, then M' accepts w , and $L(M') = \{w\}$. If $w \notin L(M)$, then M' will not accept w , and already rejects all other strings, so $L(M') = \emptyset$. Thus, we can use the emptiness detector to see which case applies, determining whether $w \in L(M)$.

Since we have shown how to decide A_{TM} using an assumed decider for E_{TM} , and we know that A_{TM} is undecidable, we have contradicted our assumption and there is no decider for E_{TM} . \square

Observation: We **never actually run** M' . It is fairly common in these proofs to construct a new TM in such a way that its language has a certain property based on the condition we want to detect. We then use an assumed decider to check whether the language has that property and use that return value to make our decision.

Aside: These “reduction proofs” do not explicitly construct a reduction function. What we are doing is building an algorithm that takes the given input and builds a new thing that will be in the language of our assumed decider iff the input was in the undecidable language we are trying to solve. This algorithm is itself a description of a reduction function, since it takes strings in the undecidable problem and maps them to strings in the assumed decidable function and maps strings not in the undecidable language to be outside the assumed decidable function. It’s just a less obvious way of representing such a function.

1.3 Equality Problem

Given two TMs, do they recognize the same language?

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

Claim 3. EQ_{TM} is undecidable.

Proof. We reduce E_{TM} to EQ_{TM} . Assume that there is a decider Q for EQ_{TM} , and construct a TM deciding E_{TM} .

Exercise: Figure out how to determine whether a TM's language is empty using a program which compares two TMs.

Such a machine is:

On input $\langle M \rangle$:

1. Construct a TM M' which rejects all strings. (Set $q_r = q_0$.)
2. Decide $R(\langle M, M' \rangle)$.

If we could decide EQ_{TM} , we could build this machine, which we know is impossible. □

1.4 Regularity Problem

Is the language of a given TM regular?

$$REG_{TM} = \{\langle M \rangle \mid M \text{ is a TM, } L(M) \text{ is regular}\}$$

Claim 4. REG_{TM} is undecidable.

Idea: Similarly to the previous proof, we will modify our input so that we can use a decider R for regularity to build a decider, though this time for A_{TM} . Given input $\langle M, w \rangle$, if M accepts w we will construct a new machine for a simple regular language. If M does not accept w , we will construct a machine that recognizes a non-regular language. We can then use R to determine whether the input machine accepts the input string based on whether our intermediate machine's language is regular.

Proof. Assume in contradiction that R is a decider for REG_{TM} and construct the following decider for A_{TM} .

On input $\langle M, w \rangle$, where M is a TM and w is a string:

1. Construct M' where on input x :
 - (a) if $x = 0^n 1^n$ for some n , accept
 - (b) else, decide $M(w)$
2. Decide $R(\langle M' \rangle)$

Here, M' starts by accepting the language $\{0^n 1^n \mid n \geq 0\}$. Then, if M accepts w , M' also accepts all other strings. If M does not accept w (whether it halts or not), M' accepts no more strings. This gives us

$$L(M') = \begin{cases} \Sigma^* & w \in L(M) \\ \{0^n 1^n\}_n & w \notin L(M) \end{cases}$$

Now, we just check whether $L(M')$ is regular, and we know whether M accepted w . Again, we never actually run M' , because if $M(w)$ loops, then M' loops, and we cannot decide. Instead, we use our assumed regularity detector to examine the code of M' and decide based on its output. □

Claim 5. REG_{TM} is unrecognizable.

Idea: Similar to undecidability, but we must reduce from an unrecognizable problem.

Exercise: What unrecognizable language(s) do we know?

Of course, this reduction should be a little easier, since we just need to build recognizers, not deciders.

Proof. Assume T is a TM which recognizes REG_{TM} . We construct the following TM recognizing $\overline{A_{TM}}$:

On input $\langle M, w \rangle$, M a TM, w a string:

1. Let N be the following TM:

On input x :

- (a) $b = M(w)$
- (b) if b is reject, reject x .
- (c) else if $x \in \{0^n 1^n \mid n \geq 0\}$, accept
- (d) else reject

2. Decide $T(\langle N \rangle)$

There are three possible cases:

1. If M accepts w , we want to reject, so we set $L(N) = 0^n 1^n$, which we know is not regular.
2. If M rejects w , we want to accept, so we set $L(N) = \emptyset$, which is regular.
3. If M loops, then $L(N) = \emptyset$, which is regular.

Now, by detecting whether $L(N)$ is regular, we can detect whether M does not accept w . We should double-check the case when our assumed detector T does not halt, which is possible when $L(N)$ is not regular, but in that case we want to reject anyway, so it is not a problem. \square

Claim 6. $\overline{REG_{TM}} = NON-REG_{TM}$ is unrecognizable.

Proof. We prove this by reducing $\overline{A_{TM}}$ to $NON-REG$, assuming R recognizes $NON-REG_{TM}$:

On input $\langle M, w \rangle$:

1. Let N be a new TM which, on input x :

- (a) if $x \in \{0^n 1^n \mid n \geq 0\}$ accept
- (b) $b = M(w)$
- (c) if b , accept x , else reject x .

2. Decide $R(\langle N \rangle)$

N 's language is either $0^n 1^n$, which is not regular, or Σ^* , which is, depending on whether M does not or does accept w , so detecting whether N 's language is regular is sufficient to solve $\overline{A_{TM}}$. \square

We have now shown that REG_{TM} and $NON-REG_{TM}$ are neither recognizable nor co-recognizable (which is when the complement of a language is recognizable). This is the most extreme possible example of a language being unapproachable with Turing Machines. The conclusion is that we cannot algorithmically detect whether a program could be replaced with a DFA/NFA/regular expression.

Hopefully you are starting to get comfortable with the structure of these proofs using contradiction and nested construction. But what happened to reductions as functions from one language to another? We have not been explicitly defining such reduction functions. Implicitly, though, we are building them as we construct our intermediate machines to pass to the assumed detectors.

2 Rice's Theorem

All of these results are about whether a Turing Machine or its language has a particular property. There is a general result encompassing most of these examples which tells us that not only in these instances, but in general, we cannot solve this type of problem.

Theorem 1 (Rice's Theorem). *Every non-trivial property of languages is undecidable.*

A property of languages is a language of TM descriptions which is neither empty nor contains all TMs (non-trivial) and inclusion in the set depends on the language of the machine, not the specific structure (property of languages)—that is, any two TMs for the same language will either both be in the set or both out of the set.

Proof. Assume in contradiction that an arbitrary non-trivial property P is decidable, with decider R . We will use this to construct a decider for A_{TM} , showing a contradiction.

First, let T_\emptyset be a TM that always rejects. Either $T_\emptyset \in P$ or $T_\emptyset \in \bar{P}$. WLOG, assume that $T_\emptyset \in \bar{P}$ (if not, then the following construction uses a decider for \bar{P} , and shows that that language is undecidable, which implies that P is also undecidable). Because P is non-trivial, there must be some TM T with $\langle T \rangle \in P$.

Exercise: Construct a decider for A_{TM} which builds an intermediate machine and uses R to differentiate whether that machine has property P , then uses that distinction to accept or reject for A_{TM} .

Construct decider S for A_{TM} as follows:

On input $\langle M, w \rangle$:

1. Use M and w to construct the machine N :

On input x :

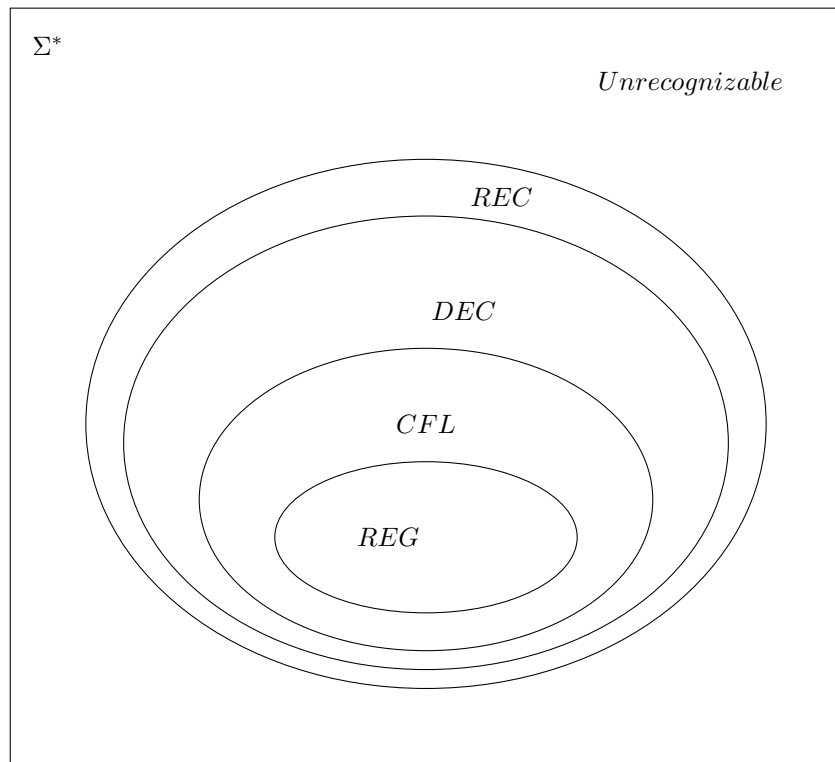
- (a) Run $M(w)$. If it rejects, reject. If it accepts, continue.
- (b) Return $T(x)$.

2. Return $R(\langle N \rangle)$.

Here, if $w \in L(M)$, then $L(N) = L(T)$. If $w \notin L(M)$, then $L(N) = \emptyset = L(T_\emptyset)$. Any machine which recognizes $L(T)$ is in P , and any machine recognizing \emptyset is not in P , so R will accept $\langle N \rangle$ iff $w \in L(M)$. \square

3 Chomsky Hierarchy

We have so far explored several increasingly-powerful types of computation and the sets of languages they recognize—corresponding to the sets of problems they can solve. This approximates the Chomsky hierarchy, which relates several classes of languages based on the grammars which can generate them. We have looked at a slightly different hierarchy, skipping some intermediate levels (such as context-sensitive grammars) and looking more at the upper limits of computation.



Co-recognizable languages are also a superset of decidable languages, such that $COREC \cap REC = DEC$.