Computer Science

Dr. Y. Liow (May 15, 2024)

Contents

Chapter 16

Decidability

16.1 Turing Decidable $\neq$ Turing Recognizable: The Acceptance Problem debug: decidability.tex

I have just shown you that there is language that is not Turing recognizable, i.e., the collection of Turing recognizable language is not the set of all languages.

In this section, I want to show you that the collection of Turing decidable language is not the collection of Turing recognizable languages.

I have already shown you that Σ^* is countable. Therefore the words in Σ^* can be listed as $w_0, w_1, w_2, ...$

Remember that each Turing machine can already be encoded as a sequence of 0's and 1's. Likewise we can also encode w_i which I will also write as $\langle w_i \rangle$.

Define

$$\textsc{Accept}_{\text{TM}} = \{\langle M \rangle \# \langle w \rangle \mid M \text{ accepts } w\}$$ $\textsc{Accept}_{\text{TM}}$

Note that this Σ_1 set has another character # and the purpose is just to separate the TM encoding from the word encoding, i.e., $\Sigma_1 = \{0, 1, \#\}$.

This language

$$\textsc{Accept}_{\text{TM}}$$

has some interesting properties.

First of all, $\textsc{Accept}_{\text{TM}}$ is Turing–recognizable. Why? Because the universal TM M_U can be used. Build a TM $\mathcal{M}$ like this: If an input is not of the form $\langle M \rangle \# \langle w \rangle$, this TM $\mathcal{M}$ just rejects. If it is, $\mathcal{M}$ will use M_U to simulate M running with input w. If M_U accepts, the TM $\mathcal{M}$ will also accept. Likewise for the reject case. If the simulation of M_U of M running on w does not halt, then M_U does not halt and our TM $\mathcal{M}$ also does not halt, therefore $\langle M \rangle \# \langle w \rangle$ is not accepted.

Let's record this ...

Theorem 16.1.1. $\textsc{Accept}_{\text{TM}}$ *is Turing–recognizable.*

Now, I am going to show you that $\textsc{Accept}_{\text{TM}}$ is not decidable. The proof is pretty cunning.

I will prove this by contradiction. Suppose $\textsc{Accept}_{\text{TM}}$ is Turing–decidable, say it is accept by A where A is a TM that always halts, i.e., it will always

enter the q_{accept} or the q_{reject} state for any input. Furthermore since A accepts $\text{Accept}_{\text{TM}}$, we know that A accepts $\langle M\rangle\#\langle w\rangle$ when M accepts w, and rejects otherwise. For simplicity, if M is a Turing machine and w is a word, I will write

$$M(w) = 1 \qquad M(w) = 1$$

if M enters q_{accept} at some point when given an input of w and

$$M(w) = 0 \qquad M(w) = 0$$

if M enters q_{reject} at some point when the input is w. Don't forget that it's possible for M to run forever for input w. Recall that if M run forever with input w, we say that M diverges with input w. In this case I will write

$$M(w) = \infty \qquad M(w) = \infty$$

This third case cannot happen if M is a Turing decider.

By our definition of A, we have

$$A(\langle M\rangle\#\langle w\rangle) = 1 \iff M \text{ accepts } w$$

and

$$A(\langle M\rangle\#\langle w\rangle) = 0 \iff M \text{ does not accepts } w$$

Note that since A is a Turing decider, we will never have

$$A(\langle M\rangle\#\langle w\rangle) = \infty$$

I will now build a Turing machine A' as follows. It will take as input $\langle M\rangle$. (If the input is not the encoding of a TM, A' just rejects.) A' then simulates A running with input $\langle M\rangle\#\langle M\rangle$. While it simulates A, if A enters q_{accept}, A' will do the opposite: A' will enter its q_{reject} state. Likewise if if A enters q_{reject}, A' will do the opposite: A' will enter its q_{accept} state. Note that since A is a decider, A will always halt in the accept or reject state – it cannot run forever. Therefore A' will also never run forever – A' must enter the accept or reject state. Now what?

First of all as I have just said, A' cannot run forever (for any input). A' must halt (i.e., enter its accept or reject state). Therefore A' is a decider. Therefore

$$A'(\langle M\rangle) = 0 \text{ or } 1$$

i.e., cannot be ∞. (If the input is not the encoding of the TM, A' just reject.)

Let's look at these two cases carefully.

Suppose $A'(\langle M\rangle) = 1$. Then the simulation of A with input $\langle M\rangle\#\langle M\rangle$ must give 0 by definition. In the same way, if $A'(\langle M\rangle) = 0$ then the simulation of A on $\langle M\rangle\#\langle M\rangle$ must give 1. In other words,

$$A'(\langle M\rangle) = 1 \iff A(\langle M\rangle\#\langle M\rangle) = 0 \tag{1}$$

But don't forget that A is the decider for $\textsc{Accept}_{\text{TM}}$. So we have

$$\begin{aligned} A(\langle M\rangle\#\langle M\rangle) = 1 &\iff M(\langle M\rangle) = 1 \\ A(\langle M\rangle\#\langle M\rangle) = 0 &\iff M(\langle M\rangle) = 0 \text{ or } \infty \end{aligned}$$

In the case when M is a decider, M will never run forever. So when M is a decider, the above becomes

$$\begin{aligned} A(\langle M\rangle\#\langle M\rangle) = 1 &\iff M(\langle M\rangle) = 1 \\ A(\langle M\rangle\#\langle M\rangle) = 0 &\iff M(\langle M\rangle) = 0 \end{aligned}$$

i.e.,

$$A(\langle M\rangle\#\langle M\rangle) = M(\langle M\rangle) \tag{2}$$

Putting (1) and (2) together, when M is a decider, we get

$$A'(\langle M\rangle) = 1 \iff A(\langle M\rangle\#\langle M\rangle) = 0 = M(\langle M\rangle)$$

Now what? ... *drumroll* ...

We run A' with $\langle A'\rangle$!!! Why? ... Because the above becomes

$$A'(\langle A'\rangle) = 1 \iff A(\langle A'\rangle\#\langle A'\rangle) = 0 = A'(\langle A'\rangle)$$

Checkmate!!! ... because the above gives this bizarre contradiction:

$$A'(\langle A'\rangle) = 1 \iff A'(\langle A'\rangle) = 0$$

Let me write down what we have at this point ...

Theorem 16.1.2. $\textsc{Accept}_{\text{TM}}$ *is Turing–recognizable but not Turing–decidable.* □

Recall that I have already shown you that Diagonal is not Turing–recognizable.

Now I give you another:

Theorem 16.1.3. $\overline{\text{ACCEPT}_{\text{TM}}}$ *is not Turing–recognizable.*

Why? Recall this result from the section on closure rules:

$$L, \overline{L} \text{ Turing recognizable} \iff L \text{ decidable}$$

This implies that

Theorem 16.1.4. *If L is Turing–recognizable and not Turing–decidable, then $\overline{L}$ not Turing–recognizable.*

Why? Otherwise, if L and $\overline{L}$ are both Turing–recognizable, then L is decidable but L is not decidable.

This and the theorem just proven immediately tells us that $\overline{\text{ACCEPT}_{\text{TM}}}$ is not Turing–recognizable. So now you have two languages which are not Turing–recognizable:

1. DIAGONAL
2. $\overline{\text{ACCEPT}_{\text{TM}}}$

16.2 The Halting Problem debug: halting-problem.tex

Consider this problem:

Is it possible to write a C++ program M that when presented with a C++ program, say P, with input w, M can tell us if P (when compiled and executable) will run forever?

This is definitely helpful since we can use M to prevent us from executing programs (with inputs) that contain infinite loops.

It turns out that this is impossible! You may write a C++ program that analyzes many common bad programming patterns that give rise to infinite loops. However your program can never catch all possible cases.

This can be proven using what we know about Turing machines at this point. First let me formulate the problem but in terms of Turing machines:

HaltingProblem: Decide if Turing machine M with input w will halt. HaltingProblem

Here's the corresponding language

$$\text{HaltingLanguage} = \{\langle M\rangle\#\langle w\rangle \mid \text{TM } M \text{ halts on input } w\} \quad \text{HaltingLanguage}$$

For convenience, I'll rewrite it as:

$$\text{Halt}_{\text{TM}} = \{\langle M\rangle\#\langle w\rangle \mid w \text{ is an input for } M \in \text{TM and } M(w) = 0, 1\} \quad \text{Halt}_{\text{TM}}$$

Remember our convention: $M(w) = 1$ means that when M executes with w as input, M will enter q_{accept}; $M(w) = 0$ means that M will enter q_{reject}. Finally $M(w) = \infty$ means that M will not enter q_{accept} or q_{reject} but will run forever.

Now when I say I want to solve the halting problem, I mean that I want an algorithm (or program that does stop at some point) that will solve this problem. This is the same as saying that I want a Turing decider to solve this problem. Now let me assume that Halt_{TM} is decidable. Remember that this means that there is some Turing machine, say I call it H, such that $L(H) = \text{Halt}_{\text{TM}}$ and H will enter q_{accept} or q_{reject} for any input.

I will use H to build a Turing machine A that is a decider ... and more importantly ... A accepts $\text{Accept}_{\text{TM}}$ which gives us a contradiction since in the previous section, I have shown you that $\text{Accept}_{\text{TM}}$ is not decidable.

This is how A works with input $\langle M\rangle\#\langle w\rangle$. First M will simulate H on

$\langle M \rangle \# \langle w \rangle$. Note that I have assumed that H is a decider. So H must halt:

$$H(\langle M \rangle \# \langle w \rangle) = 0 \text{ or } 1$$

(i.e., not ∞.)

1. Suppose $H(\langle M \rangle \# \langle w \rangle) = 1$. In this case, A proceeds to simulate M with input w. Note that H already tells us this simulation will halt since

   $$H(\langle M \rangle \# \langle w \rangle) = 0, 1$$

   If the simulation of M with input w ends with an accept state, A will also accept. If the simulation of M with input w ends with a reject state, A will also reject. In both cases, you see that

   $$A(\langle M \rangle \# \langle w \rangle) = M(w)$$

   Note that A does not run on forever.
2. Suppose $H(\langle M \rangle \# \langle w \rangle) = 0$. In this case, H tells us that if M executes with input w, M will not halt, i.e.,

   $$M(w) = \infty$$

   In this case, A will not bother with simulating M with input w: A simply enters its reject state and halt. In other words, in this case

   $$A(\langle M \rangle \# \langle w \rangle) = 0$$

   and

   $$M(w) = \infty$$

   Note that in this case, A does not run on forever.

Note that A halts in both cases: A is a decider. Now I want to show

$$L(A) = \text{Accept}_{\text{TM}}$$

If

$$\langle M \rangle \# \langle w \rangle \in L(A)$$

then by definition

$$A(\langle M \rangle \# \langle w \rangle) = 1$$

This means that I'm not in case 2 above. So I must be in case 1. This means that

$$M(w) = A(\langle M \rangle \# \langle w \rangle) = 1$$

Therefore M accepts w, i.e.,

$$\langle M\rangle\#\langle w\rangle \in \text{Accept}_{\text{TM}}$$

by definition of $\text{Accept}_{\text{TM}}$. This proves that

$$L(A) \subseteq \text{Accept}_{\text{TM}}$$

Now I will prove $\text{Accept}_{\text{TM}} \subseteq L(A)$. Let $\langle M\rangle\#\langle w\rangle \in \text{Accept}_{\text{TM}}$. This means that M accepts w. Again, this means that I'm not in case 2. This implies that I'm in case 1. Therefore

$$A(\langle M\rangle\#\langle w\rangle) = M(w) = 1$$

This means that

$$\langle M\rangle\#\langle w\rangle \in L(A)$$

Hence $\text{Accept}_{\text{TM}} \subseteq L(A)$.

Where are we? At this point, I have shown that

$$\text{Accept}_{\text{TM}} = L(A)$$

But don't forget: A is a Turing decider!!! That contradicts the fact that $\text{Accept}_{\text{TM}}$ is not Turing decidable which was proven in the previous section. Therefore my assumption that Halt_{TM} is decidable is incorrect.

We conclude that the language Halt_{TM} is *not* decidable.

Theorem 16.2.1. Halt_{TM} *is not decidable, i.e., the halting problem is not decidable.*

Notice that the technique above reduces the decidability of the halting problem (or the decidability of the Halt_{TM} language) to the Turing–recognizability of the acceptance problem (or the $\text{Accept}_{\text{TM}}$ language). This is a very standard technique in proving the non–Turing–decidability of a language L. You assume your language is decidable, say it's decided by a Turing decider M and you follow up by designing another Turing machine (or decider) that contradicts some known result. In the above case, a decider for Halt_{TM} would result in a decider for $\text{Accept}_{\text{TM}}$.

This technique is called "**reduction**". The idea of reducibility therefore relates reduction

languages. We say that Halt_{TM} is **reducible** to $\text{Accept}_{\text{TM}}$ and we write reducible

$$\text{Accept}_{\text{TM}} \leq \text{Halt}_{\text{TM}}$$

Our proof technique above is this: The assumption that Halt_{TM} is decidable forces $\text{Accept}_{\text{TM}}$ to be decidable as well.

In general, you have the following facts:

Theorem 16.2.2. *Let L and L' be two languages. Then*

1. *$L \leq L'$, L' decidable $\implies$ L decidable.*
2. *$L \leq L'$, L not decidable $\implies$ L not decidable.*

You can define $\leq$ more formally. You can think of $\leq$ as a way to relate languages in the space of Lang_{TM} through their complexity. This is somewhat analogous to $\leq$ on real numbers $\mathbb{R}$.

16.3 The Empty Problem debug: empty-problem.tex

Here's another problem:

EmptyProblem$_{\text{TM}}$: Given Turing machine M, decide if $L(M) = \{\}$ EmptyProblem$_{\text{TM}}$

Let me give you the corresponding language:

$$\text{Empty}_{\text{TM}} = \{\langle M \rangle \mid L(M) = \emptyset\}$$

I will show you that Empty$_{\text{TM}}$ is not decidable.

Suppose Empty$_{\text{TM}}$ is decided by Turing decider E. I'm going to design a Turing machine E' that will decide Accept$_{\text{TM}}$.

For an input of $\langle M \rangle \# \langle w \rangle$ to E', E' will simulate $\langle M_w \rangle$ on E. Now what is this M_w?!? This is a slight modification to M. Basically, M_w first checks if an input is w. If the input is not w, M_w rejects right away. If the input is w, M_w behaves just like M on the input of w. It's clear that given $\langle M \rangle$, you can easily modify it to become $\langle M_w \rangle$. The important thing is either

$$M_w = \{\}$$

or

$$M_w = \{w\}$$

There are no other cases. Note that the case of

$$M_w = \{w\}$$

implies that M_w will of course halt (in its accept state) on input w. On the other hand, the case of

$$M_w = \{\}$$

might be the case of either

$$M_w(w) = 0$$

i.e., M_w halts in its reject state or

$$M_w(w) = \infty$$

i.e., M_w runs forever. Ahh .... but that's not a problem!!! Why? ... Because we can use E to determine if

$$M_w = \{\}$$

or not and therefore we can avoid running M_w on w which can potentially run

forever!!! See it!!!

So this is how I'm going to construct E'. On input $\langle M \rangle \# \langle w \rangle$, E' will simulate E with input $\langle M_w \rangle$. (Remember that it's easy to construct $\langle M_w \rangle$ from $\langle M \rangle$.)

1. If the simulation of E with input $\langle M_w \rangle$ results in an accept state, i.e.,

   $$E(\langle M_w \rangle) = 1$$

   then

   $$M_w = \{\}$$

   and we know that M does not accept w. In this case, I make E' go to its reject state, i.e.,

   $$E'(\langle M \rangle \# \langle w \rangle) = 0$$

   (You'll see why I have to invert 0 and 1 later.)
2. On the other hand if the simulation goes into the reject state,

   $$E(\langle M_w \rangle) = 0$$

   then

   $$M_w \neq \{\}$$

   But if $M_w \neq \{\}$, then M_w must be $\{w\}$. which means that M must accept w. At this point, I make E' goes into its accept state, i.e.,

   $$E'(\langle M \rangle \# \langle w \rangle) = 1$$

Note that since E is a decider, the above are the only cases, i.e., we cannot possibly have $E(\langle M_w \rangle) = \infty$. By design,

$$\begin{aligned} E(\langle M_w \rangle) = 0 &\implies E'(\langle M \rangle \# \langle w \rangle) = 1 \\ E(\langle M_w \rangle) = 1 &\implies E'(\langle M \rangle \# \langle w \rangle) = 0 \end{aligned}$$

E and E' cannot have ∞ as results. Therefore

$$E(\langle M_w \rangle) = 0 \iff E'(\langle M \rangle \# \langle w \rangle) = 1$$

We also have the following based on the definition of E, i.e., that it's a decider

for Empty_{TM} and the definition of M_w:

$$E(\langle M_w \rangle) = 1 \implies M_w = \{\} \implies M(w) = 0, \infty \implies M \text{ does not accept } w$$
$$E(\langle M_w \rangle) = 0 \implies M_w \neq \{\} \implies M(w) = 1 \implies M \text{ accepts } w$$

(Note that we actually did not run or simulate M on w: this is key because the execution of M with input w might go on forever!) In other words

$$E(\langle M_w \rangle) = 0 \iff M \text{ accepts } w$$

Putting everything together we have

$$E'(\langle M \rangle \# \langle w \rangle) = 1 \iff E(\langle M_w \rangle) = 0 \iff M \text{ accepts } w$$

Altogether, we have constructed a decider E' which can decide the acceptance problem, i.e.,

$$L(E') = \text{Accept}_{\text{TM}}$$

But this contradicts the fact that $\text{Accept}_{\text{TM}}$ is not a Turing–decidable language from an earlier section.

Done!

Theorem 16.3.1. Empty_{TM} *is not a Turing–decidable language.*

In the above proof, I reduce the Empty_{TM} to $\text{Accept}_{\text{TM}}$:

$$\text{Accept}_{\text{TM}} \leq \text{Empty}_{\text{TM}}$$

Exercise 16.3.1. Prove that Empty_{TM} is not decidable by reducing it to Halt_{TM} instead.

(Hint: Given M and w. You want to know if there's an algorithmic (i.e., by a decider) way to tell if M halts on input w. If M accepts w, then of course it does halt. However if M does not accept w, it can either go to its reject state and halt or it can run forever. Therefore you want to prevent running M on w if it runs on forever. Not a problem! We simply run M_w through a purported decider for Empty. If M_w is in Empty, then M does not accept w. If M_w is not in Empty, then M accepts w. Therefore M must halt on w.)

16.4 More decision problems for TM

debug:

decision-problems-for-tm.tex

The acceptance problem and halting problem earlier are examples of what we call **decision problems for Turing machines**. You can of course rephrase them for NFAs, context-free grammars (or PDAs) and DFAs.

decision problems for Turing machines

Here are more decision problems for TMs.

Exercise 16.4.1. Is the following language decidable:

$$\textsc{Size-One}_{\text{TM}} = \{\langle M \rangle \mid |L(M)| = 1\}$$

Do not peek ... the solution is on the next page.

SOLUTION.

We will reduce to this the acceptance problem, i.e.,

$$\text{Accept}_{\text{TM}} \leq \text{Size-One}_{\text{TM}}$$

Suppose $\text{Size-One}_{\text{TM}}$ is decidable, say it's decided by S. Then I claim that I can construct a decider for $\text{Accept}_{\text{TM}}$.

Construct A as follows. Suppose $\langle M \rangle \# \langle w \rangle$ is given as an input to A where M is a TM and w is an input for M. (If the input is not of the form $\langle M \rangle \# \langle w \rangle$, A rejects immediately.) A will first construct M_w, a Turing machine that works exactly like M if the input is w but if the input is not w, M will reject after checking that the input is not w. In other words $L(M_w) = \{\}$ or $\{w\}$. A then does the following:

- A will first simulate S with M_w. Note that S is a decider and therefore must halt.
- If while A is simulating S, A sees that S halts in the accept state, then A will enter its accept state. [Note that S accepts iff M_w accept w since by definition M_w can only accept w and nothing else.]
- If while A is simulating S, A sees that S halts in the reject state, then A will enter its reject state. [Note that S rejects iff M_w rejects which means that M does not accept w.]
- Again, note that S is a decider and therefore must enter the accept or reject state at some point – i.e., S cannot run forever.

The above means that A accepts $\langle M \rangle \# \langle w \rangle$ iff M accepts w. Furthermore A is a decider. We have shown that But this means that $\text{Accept}_{\text{TM}}$ is decidable (by the decider A) which is a contradiction.

Exercise 16.4.2. Let $k \geq 0$ be an integer. Is the following language decidable:

$$\{\langle M \rangle \mid |L(M)| = k\}$$

Exercise 16.4.3. Is the following language decidable:

$$\{\langle M\rangle \mid |L(M)| \text{ is finite}\}$$

Exercise 16.4.4. Is the following language decidable:

$$\textsc{Infinite}_{\text{TM}} = \{\langle M \rangle \mid L(M) \text{ is infinite } \}$$

Exercise 16.4.5. Is the following language decidable:

$$\text{Null}_{\text{TM}} = \{\langle M \rangle \mid \epsilon \in L(M)\}$$

SOLUTION.

Assume that Null_{TM} is decidable, say it is decided by D.

Construct a TM A that does this: Let $\langle M \rangle \# \langle w \rangle$ be an input for A.

1. A construct a TM M' (which depends on M and w) which does the following: When M' executes, it replaced the input with w, and then proceeds to work exactly like M. Note that if we run M' on ϵ, then it would be the same as running M on w. Of course when I say A "constructs" M', I mean A write the encoding, $\langle M' \rangle$ somewhere on its tape.
2. A then simulates D on $\langle M' \rangle$. Note that D is decidable therefore D must either accept or reject (D cannot run forever). This means that as A simulates D one step at a time A will at some point in time see D entering the accept or the reject state.
3. If while A is simulating D running on $\langle M' \rangle$, A sees that D accepts $\langle M' \rangle$ (i.e., M' accepts ϵ, which means that M accepts w), then A enters its accept state.
4. If while A is simulating D running on $\langle M' \rangle$, A sees that D rejects $\langle M' \rangle$ (i.e., M' does not accepts ϵ, which means that M does not accepts w), then A enters its reject state.

We have shown that

$$\text{Accept}_{\text{TM}} \leq \text{Null}_{\text{TM}}$$

This means that A is a decider for $\text{Accept}_{\text{TM}}$ which is a contradiction. Therefore Null_{TM} cannot be a decidable language. □

As exercise, try to write a proof that shows

$$\text{HALT}_{\text{TM}} \leq \text{Null}_{\text{TM}}$$

Exercise 16.4.6. Is the following language decidable:

$$\{\langle M \rangle \mid M \text{ has 42 states } \}$$

SOLUTION.

Yes. Depending on the way TMs are encoded, design a TM, D, such that when D is given $\langle M \rangle$, D moves the read-write head to the section of $\langle M \rangle$ that encodes the set of states and begin counting the number of states. If the counting stops before 42, D rejects. If the counting goes up to 43, D rejects. Otherwise D accepts.

Exercise 16.4.7. Consider the following problem:

AllProblem$_{\text{TM}}$: Decide if a TM M accepts all strings. AllProblem$_{\text{TM}}$

Is the following language decidable:

$$\text{All}_{\text{TM}} = \{\langle M\rangle \mid L(M) = \Sigma^* \text{ where } \Sigma \text{ is the input alphabet of } M\} \quad \text{All}_{\text{TM}}$$

Exercise 16.4.8. Is the following language decidable:

$$\textsc{Equal}_{\text{TM}} = \{\langle M_1 \rangle \# \langle M_2 \rangle \mid L(M_1) = L(M_2)\} \qquad \textsc{Equal}_{\text{TM}}$$

This is related to the problem of determining if two given TMs accept the same language.

Exercise 16.4.9. Is the following language decidable:

$$\textsc{Regular}_{\text{TM}} = \{\langle M\rangle \mid \exists DFA M' \text{ such that } L(M') = L(M)\} \qquad \textsc{Regular}_{\text{TM}}$$

This is the problem of determining if a Turing machine accepts the same language as a DFA, i.e., the problem of determining if the language of a Turing machine is regular.

I'll show

$$\textsc{Accept}_{\text{TM}} \leq \textsc{Regular}_{\text{TM}}$$

Suppose I'm given $\langle M\rangle\#\langle w\rangle$. Suppose I assume $\textsc{Regular}_{\text{TM}}$ is decided by a TM R. Basically I then need to construct (using R), a decider, say A for $\textsc{Accept}_{\text{TM}}$. I would expect something like this:

$$\begin{aligned} M(w) = 1 &\iff R(M) = 1 \iff A(Mw) = 1 \\ M(w) = 0/\infty &\iff R(M) = 0 \iff A(Mw) = 0 \end{aligned}$$

Now it turns out that for R, I cannot use M directly. I have to modify M to say another TM say I call it M_w so that the above desired implications become

$$\begin{aligned} M(w) = 1 &\iff R(M_w) = 1 \iff A(Mw) = 1 \\ M(w) = 0/\infty &\iff R(M_w) = 0 \iff A(Mw) = 0 \end{aligned}$$

M_w works as follows: on input x, if $x = 0^n1^n$, M_w accepts. If $x \neq 0^n1^n$, M_w runs on w like M on w. In other words

$$M_w(x) = \begin{cases} 1 & \text{if } x = 0^n1^n \text{ or } M(w) = 1 \\ ? & \text{otherwise} \end{cases}$$

Now the problem is that in the otherwise case, and this is important, you can't say it's 0. It might be ∞. In the "if" case, the fact that $x = 0^n1^n$ can be checked without running forever. But the "$M(x) = 1$" if not true, might run forever, i.e., it can be "$M(x) = 0$" or ∞. In other words, the complete description of $M_w(x)$ is

$$M_w(x) = \begin{cases} 1 & \text{if } x = 0^n1^n \text{ or } M(w) = 1 \\ ? & \text{if } x \neq 0^n1^n \text{ and } M(w) \neq 0 \end{cases}$$

i.e.,

$$M_w(x) = \begin{cases} 1 & \text{if } x = 0^n1^n \text{ or } M(w) = 1 \\ 0 & \text{if } x \neq 0^n1^n \text{ and } M(w) = 0 \\ \infty & \text{if } x \neq 0^n1^n \text{ and } M(w) = \infty \end{cases}$$

Get it? This is very important. Read the above again very carefully.

If $M(w) = 1$, let M_w be a TM that accepts Σ^*. If $M(w) = 0$ or ∞, let M_w be a TM that accepts $\{0^n1^n \mid n \geq 0\}$. (The pair language Σ^* and $\{0^n1^n \mid n \geq 0\}$ can be replaced a pair L, L' where L is regular and L' is not regular.)

Basically we want

$$R(M_w) = 1 \iff M(w) = 1$$

Let M, w be a turing machine and input. M_w is defined to the TM that works this way:

1. If input x is 0^n1^n, M_w accepts
2. Otherwise, M_w runs M on w and accepts x iff M accepts w. (and M_w rejects w iff M rejects w and M_w runs forever on w iff M runs forever on w.)

Note that

$$\begin{aligned} M(w) = 1 &\implies L(M_w) = \Sigma^* \\ M(w) = 0 \text{ or } \infty &\implies L(M_w) = \{0^n1^n \mid n \geq 0\} \end{aligned}$$

In other words

$$\begin{aligned} M(w) = 1 &\iff L(M_w) = \Sigma^* \\ M(w) = 0 \text{ or } \infty &\iff L(M_w) = \{0^n1^n \mid n \geq 0\} \end{aligned}$$

Why is this important? Suppose we have M_w. Now I run R on $\langle M_w \rangle$. What will happen? Note that R is a decider. So we have only two (not three) possibilities: $R(\langle M_w \rangle) = 0$ or 1. But hang on ... $R(\langle M_w \rangle) = 0$ means that M_w is regular and $R(\langle M_w \rangle) = 1$ means that M_w is not regular. The way I have it set up, it's not just that M_w is not regular, it's not regular in the specific sense that $L(M_w) = \{0^n1^n \mid n \geq 0\}$!!! In other words, I can add the following to the above:

$$\begin{aligned} M(w) = 1 &\iff L(M_w) = \Sigma^* \iff R(\langle M_w \rangle) = 1 \\ M(w) = 0 \text{ or } \infty &\iff L(M_w) = \{0^n1^n \mid n \geq 0\} \iff R(\langle M_w \rangle) = 0 \end{aligned}$$

What's the point of "$x = 0^1 1^n$"? Nothing other than the fact that $\{0^n 1^n \mid n \geq 0\}$ is not regular and Σ^* is regular.

Now I'm going to build a TM A that does the following: Let x be an input.

1. If x is not of the form M, w (NOTE: this check does not run forever)
   a) A rejects.
2. Otherwise (i.e., $x = M, w$) A continues as follows:
   a) A constructs M_w on its input tape
   b) A runs R on M_w
   c) If $R(M_w) = 1$, A accepts, i.e., $A(x) = A(\langle M, w \rangle) = 1$
   d) Otherwise (i.e., $R(M_w) = 0$), A rejects, i.e., $A(x) = A(\langle M, w \rangle) = 0$. (NOTE: It's 0 and not ∞ since every step does not run forever.)

SOLUTION. . Suppose the above problem is decidable, say D is a Turing decider such that $L(D) = \text{REGULAR}_{\text{TM}}$. Construct a TM D' as follows. D' takes $\langle M \rangle$ as inputs. D' will run D on M. Since D is a decider, $D(\langle M \rangle)$ can only be 0 or 1. If $D(\langle M \rangle) = 1$, then D' runs M on $\langle M \rangle$ and accept if M accepts $\langle M \rangle$ and rejects if M rejects $\langle M \rangle$. If $D(\langle M \rangle) = 0$

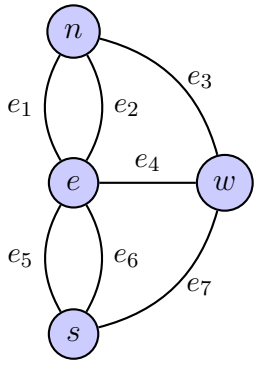
16.5 Some decision problems debug: some-decision-problems.tex

Decision problems need not be about problems on automatas to be solved (decided) by Turing deciders. They can be problems about pretty much any discrete mathematical structures that we hope to solve using Turing deciders. Just so you see that the idea of decision can be very general, here are some examples.

16.5.1 Some decision problems for undirected graphs

graph
vertices
edges

An (undirected) **graph** is a finite bunch of dots and a bunch of lines where each line joins two dots. The dots are called **vertices** and the lines are called **edges**. (Refer to your discrete math textbook for more details.) Here's the picture of a graph (probably the most famous of all graphs):

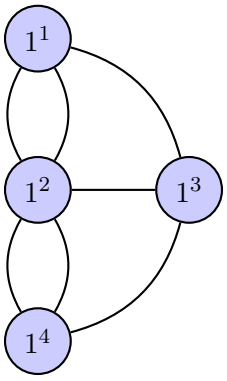


If you like, you can think of the vertices as places and the edges as roads. If the edges of a graph have arrows, then you think of the roads as enforcing directions and you can only travel on the road in the direction specified by the arrow.

multigraph
loop

Technically speaking when one say a "graph", one is refering to graphs where there is at most one edge joining two vertices and furthermore a edge cannot join the same vertex. The above is an example of **multigraph** where there can be multiple edges joining two vertices and where there can be loops (a **loop** is an edge joining the same vertex). The important thing about a graph is not the way the vertices and edges are named, but rather the structure of the graph, i.e., how the vertices are connected.

Suppose I rename the vertices like this and I ignore names for the edges:



Mathematically, the definition of a multigraph looks like this:

$$G = (V, E)$$

where

$$V = \{1^1, 1^2, 1^3, 1^4\}$$

and

$$E = \{2(1^1, 1^2),\ 2(1^2, 1^1),\ 2(1^2, 1^4),\ 2(1^4, 1^2),\ 1(1^1, 1^3),\\ 1(1^3, 1^1),\ 1(1^2, 1^3),\ 1(1^3, 1^2),\ 1(1^3, 1^4),\ 1(1^4, 1^3)\}$$

The value

$$2(1^1, 1^2)$$

in E tells us that you can go from 1^1 to 1^2 in 2 ways. (Aside: Something like $\{a, a, b, c, c\}$ is a set and repetitions are not significant, i.e., $\{a, a, b, c, c\} = \{a, b, c\}$. Something like $\{2 \cdot a, 1 \cdot b, 2\cdot\}$ is called a **multiset** – it's like a set except that repetitions *are* significant.)

multiset

To encode G, we encode V and E. For V we can do this:

$$\langle V \rangle = 1^1 01^2 01^3 01^4$$

and for each value in E, there are three values and we encode them in the obvious way and separate the three values by 0. For instance

$$\langle 2(1^1, 1^2) \rangle = 1^2 01^1 01^2$$

To encode E, say $E = \{t_1, t_2, t_3, \cdots\}$, we encode each value t_i in E and separate them by 00:

$$\langle E \rangle = 1^2 01^1 01^2 001^2 01^2 01^1 00 \cdots 001^1 01^4 01^3$$

We then encode G by concatenating the encoding of V, 000, and the encoding of E:

$$\begin{aligned}\langle G \rangle &= \langle V \rangle 000 \langle E \rangle \\ &= 1^1 01^2 01^3 01^4 \cdot 000 \cdot 1^2 01^1 01^2 001^2 01^2 01^1 00 \cdots 001^1 01^4 01^3\end{aligned}$$

With the above encoding, Turing machines can now analyze and compute with graphs.

Here are some very famous decision problems for graphs:

1. EulerianPath: Given a multigraph G, is there an Euler path in G? (i.e., a path that visits every edge exactly once).
2. EulerianCycle: Given a multigraph G, is there a closed Euler path in G? (i.e., a path that visited every edge exactly once and the first vertex in the path is also the last).
3. HamiltonianPath: Given a graph G, is there a Hamilton path in G? (i.e., a path that visits every node exactly once).
4. HamiltonianCycle: Given a multigraph G, is there a closed Euler path in G? (i.e., a path that visited every node exactly once and the first vertex in the path is also the last).

In fact they are all decidable problems.

Exercise 16.5.1. Prove that the above problems are all decidable. □

But that's not the end of the story – there's more.

You can divide Turing machines into deciders and non-deciders. Among Turing deciders, you can further divide them in different ways. One way to classify deciders is by their "runtime performance". Meaning to say the following: Suppose M is a decider and w is an input for M. You can measure the "runtime performance" of M when it runs w by counting the number of transitions to a halting state (acccept or reject state). So the "number of transitions" to reach a halting state can be used as a measurement of time used. Based on this concept of time, one can divide problems into different classes. This gives a measure of complexity of a problem. For details see notes on time complexity classes.

The interesting thing about the HamiltonianCycle and EulerianCycle is that despite the fact that they are so similar, the Eulerian cycle problem can be decided in a way that is very different from the way a Hamiltonian cycle problem is decided:

There is a decider of the Eulerian cycle that has a polynomial runtime while nobody knows if there Hamiltonian cycle can be decided in polynomial runtime. When I say "polynomial runtime", I mean that there is a decider M and a polynomial function $p(n)$ such that for any Eulerian problem, i.e., $\langle G \rangle$ where G is a multigraph, M will run with $O(p(n))$ number of transitions where n is the length $\langle G \rangle$. The class of problems that can be solved by deciders with polynomial runtime is denoted by P. Therefore we would write

$$\text{EulerianCycle} \in \text{P}$$

You can and should think of problems (or rather their languages) in P as problems that can be solved quickly.

However we do not know if there's a decider for the Hamiltonian cycle that runtimes in polynomial runtime. HamiltonianCycle belongs to a class of problems denoted by NP. You can and should think of the problems in NP as being problems which are probably difficult and needs more time to solve. The class NP contains P. One of the biggest unsolved problems in computer science and mathematics is whether P is actually NP.

For details see notes on time complexity classes.

16.5.2 More decision problems

Exercise 16.5.2. (Tough) Let n be a positive integer. Is the following problem decidable: Let S be a set of matrices with integer entries. Is there a sequence of matrices in S which multiplies to a zero matrix? □

Exercise 16.5.3. (Tough) Let m, n, a, b be positive integers. Given an m-by-n tray and a set S of a-by-b tiles, is there a collection of tiles from S that will completely tile the tray? □

Exercise 16.5.4. (Tough) Are the following problems decidable? Given a collection P of $n > 0$ points in a plane,

1. What is the largest subcollection of points P that lie on a single straight?
2. What is the least number of lines contains all points of P?

□

Exercise 16.5.5. (Tough) Is the following problem decidable: Given a polynomial $p(x)$ with integer coefficients, does $p(x)$ have an integer root? □

16.6 Decision problems for DFAs debug: decision-problems-for-dfas.tex

Here are some decision problems for DFAs.

Exercise 16.6.1. Decision problems for DFAs. It's not too surprising that DFAs can also be encoded. Therefore one can ask if it's possible to construct TM or deciders to accept DFA encodings (possibly also with encoded inputs) to solve certain problems. Are the following problems decidable? This means "can you find a decider for the given problems.?"

1. Given a DFA M, is $L(M) = \{\}$? (This is $\text{EMPTY}_{\text{DFA}}$).
2. Given a DFA M, is $L(M) = \Sigma^*$? (Σ is the input alphabet of M).
3. Given a DFA M, is $\epsilon \in L(M)$?
4. Given a DFA M, is $L(M)$ infinite?
5. Given a DFA M, is $L(M)$ finite? (What I you are given M and n and the question is to decide is $|L(M)| = n$?)
6. Given a DFA M and input w (for M), is w in $L(M)$? (This is the acceptance problem for DFAs)
7. Given a DFA M and input w (for M), does M halt when it runs on input w? (This is the halting problem for DFAs.)
8. Given two DFAs M, M', is $L(M)$ the same as $L(M')$?

□

Don't peek ... some solutions on next page.

SOLUTION.

1. If M is a DFA, then $L(M) = \{\}$ exactly when there is no path (in the DFA diagram) from the initial state to any of its accept state. Therefore we do the following:

Construct a TM D that accepts $\langle M \rangle$ (where M is a DFA) and does the following (if the input is not the encoding of a DFA, D rejects right away). The tape for A is of the form

$$\$\langle M \rangle \# x \# y$$

First A writes the encoding of the initial state q_0:

$$\$\langle M \rangle \# \# \langle q_0 \rangle$$

y collects the states which are "not processed" while x collects the states which are already "processed". A will process each state q by computing what states q can go to via transitions. The states that q can go to and which are not stored in x and y and stored in y. Once a state is processed, the state is removed from y and moved to x. If an accept state is stored in x, then this means that there's a path from the initial state of M to an accept state of M. Therefore if A sees an accept state in x, A enters its reject state. If y is empty (and no accept state of M appears in x), then A enters its accept state.

3. Construct a TM A such that when given $\langle M \rangle$, A checks if the initial state is an accept state. If it is so, A enters its accept state. Otherwise A enters its reject state. Clearly $L(A) = \{\langle M \rangle \mid M \text{ is a DFA}, \epsilon \in L(M)\}$.

4. HINT: If M is a DFA, $L(M)$ is infinite iff in the DFA diagram there is a path from the start state to an accept that contains a loop.

6. HINT: w is in $L(M)$ is w is a path in the DFA diagram of M from M's initial state to one of its accept states.

7. Design a TM A that checks that the input is valid and then accept. Otherwise, reject. (Right?)

8. HINT: Let $L_i = L(M_i)$. Let $L = ((L_1 \cap \overline{L_2}) \cup (L_2 \cap \overline{L_1}))$ (the symmetric difference of L_1 and L_2). Then $L_1 = L_2$ iff $L = \{\}$.

16.7 Decision problems for context-free grammars

debug: decision-problems-for-grammars.tex

Exercise 16.7.1. Decision problems for CFLs. Are the following problems decidable?

1. Given a CFG G, is $L(G) = \{\}$?
2. Given a CFG G, is $L(G) = \Sigma^*$? (Σ is the set of terminals of G).
3. Given a CFG G, is $\epsilon \in L(G)$?
4. Given a CFG G, is $L(G)$ infinite?
5. Given a CFG G, is $L(G)$ finite?
6. Given a CFG G and string w (of terminals of G), is w in $L(G)$?
7. Given two CFGs G, G', is $L(G)$ the same as $L(G')$?

□

16.8 The Equal Problem debug: equal-problem.tex

Index