Operations on Turing-Recognizable and Turing-Decidable Languages

Jeffrey Shallit
School of Computer Science
University of Waterloo
shallit@uwaterloo.ca
https://cs.uwaterloo.ca/~shallit
Let’s figure out what operations we can do on Turing-recognizable and have them still be Turing-recognizable: the closure properties.

And the same thing for Turing-decidable languages.

Let’s start with union of Turing-decidable languages (easy!).

We have a TM $M_1$ deciding $L_1$ and a TM $M_2$ deciding $L_2$. How can we combine them to get a TM $M$ for $L_1 \cup L_2$?
Closure under union

Create a TM $M$ that works as follows: it first runs $M_1$ on input $x$, then runs $M_2$ on $x$, and accepts if either $M_1$ and $M_2$ accepted $x$. Otherwise, $M$ rejects $x$.

Now let’s do union of Turing-recognizable languages.

If we use the same argument, something goes wrong. What is it?

The problem is that when we run $M_1$ on $x$, it might not halt. So if $M_1$ runs forever on $x$, while $M_2$ accepts $x$, our new TM $M$ would not accept $x$ (which it should).

So what can we do?
The idea is for $M$ to simulate $M_1$ and $M_2$ in parallel.

First $M$ does one step of $M_1$, then one step of $M_2$, then one step of $M_1$, then one step of $M_2$, etc.

So $M$ accepts iff either $M_1$ or $M_2$ eventually halts and accepts $x$.

So $M$ recognizes $L_1 \cup L_2$. 

Closure under union
Closure under intersection

How about intersection?

In this case, we can use the same construction for either Turing-recognizable or Turing-decidable languages.

Here it doesn’t matter if $M_1$ doesn’t halt.
How about complement?

For Turing-decidable languages, it’s easy: we just interchange the states $q_{\text{acc}}$ and $q_{\text{rej}}$.

This shows that if $L$ is Turing-decidable, so is $\overline{L}$.

However, this proof doesn’t work for Turing-recognizable languages. Why?
Closure under complement

In fact, we already know an example of a Turing-recognizable language whose complement is not Turing-recognizable. Namely,

\[ L = \{ w : w \text{ is not the encoding of a TM } T \} \]
\[ \cup \{ e(T) : T \text{ accepts } e(T) \}. \]

This language is clearly Turing-recognizable: on input \( w \) we first check if \( w \) is the valid encoding of a TM, and accept if it is not.

Otherwise, \( w = e(T) \). We simulate \( T \) on input \( e(T) \) and accept if this simulation accepts.

However \( \overline{L} = \text{DIAG} = \{ e(T) : T \text{ doesn’t accept } e(T) \} \), which is not Turing-recognizable.

So the Turing-recognizable languages are not, in general, closed under complement.
I just want to remind you of what it means when we say a class $C$ is \textit{not} closed under some operation, such as complement.

It does \textit{not} mean that for every $L \in C$, the complement of $L$ is not in $C$.

It just means that for \textit{some} $L \in C$, the complement of $L$ is not in $C$. 
A useful theorem

So the Turing-recognizable languages are not closed under complement. However, we do have the following very useful theorem:

**Theorem.** If both $L$ and $\overline{L}$ are Turing-recognizable, then $L$ is Turing-decidable.

Take a moment and try to create the proof yourself.
Theorem. If both $L$ and $\overline{L}$ are Turing-recognizable, then $L$ is Turing-decidable.

OK, here's the proof: we know there is a TM $M_1$ recognizing $L$ and a TM $M_2$ recognizing $\overline{L}$.

On input $x$, we run $M_1$ on $x$ and $M_2$ on $x$ in parallel.

Eventually either $M_1$ accepts $x$ or $M_2$ accepts $x$.

At that point we halt and

- accept if $M_1$ accepts $x$,
- reject if $M_2$ accepts $x$.

So this gives a TM $M$ deciding $L$. 

Application

Application: let’s prove that $L_{\text{empty}} := \{e(T) : L(T) = \emptyset\}$ is not Turing-recognizable.

Assume that it is. Then

$$L := L_{\text{empty}} \cup \{w : w \text{ is not a valid TM encoding}\}$$

is also Turing-recognizable.

But

$$\overline{L} = \{e(T) : L(T) \neq \emptyset\}$$

is the language Accepts-Something, which we already saw is Turing-recognizable (by dovetailing).

But then both $L$ and its complement $\overline{L}$ are Turing-recognizable. So by our theorem $\{e(T) : L(T) \neq \emptyset\} = \text{Accepts-Something}$ is Turing-decidable, contradicting what we proved last week.

So $L_{\text{empty}} := \{e(T) : L(T) = \emptyset\}$ is not Turing-recognizable.
A more general result

Exactly the same proof shows

**Theorem.** If $L$ is Turing-recognizable but not Turing-decidable, then $\overline{L}$ is not Turing-recognizable.

This is very useful for creating lots of languages that aren't Turing-recognizable.
How about concatenation?

**Theorem.** If $L_1$ and $L_2$ are Turing-recognizable, then so is $L_1L_2$.

**Proof.** Here is another example where nondeterminism makes the proof very easy.

Given TM $M_1$ recognizing $L_1$, and $M_2$ recognizing $L_2$, let us make a 3-tape nondeterministic TM $M$ behaving as follows:

$M$ uses tape 1 for the input $x$.

On tape 2 we nondeterministically write out some finite string $y$.

On tape 3 we nondeterministically write out some finite string $z$.

We run $M_1$ on tape 2, and $M_2$ on tape 3. If both accept, and if $x = yz$, we accept. ■

Exercise: prove that if both $L_1$ and $L_2$ are Turing-decidable, then so is $L_1L_2$.  
