Decidability and Semidecidability

Universal Turing Machines and the Halting Problem

Decidability and Semidecidability

A language $L$ is said to be decidable if there exists a DTM $M$ such that $M$ has 2 halting states: $H_a$ (accepting state) and $H_r$ (rejecting state), and on any input $x$, $M$ always halts, such that if $x \in L$, $M$ accepts $x$ (i.e., $M$ on input $x$ reaches the halting state $H_a$) and if $x$ is not in $L$, then $M$ rejects $x$ (i.e., $M$ on input $x$ reaches the halting state $H_r$).

- This means we have an algorithm to solve the membership problem for $L$ iff $L$ is decidable.
- The membership problem for $L$ is to determine whether a given $x$ is in $L$ or not.

**Proposition:**
$L$ is decidable iff $L$ and $L'$ are both Turing-Acceptable.

**Proof:**

$\Rightarrow$
Assume $L$ is decidable. Then, by definition there exists a DTM $M$ with 2 halting states and on any input $x$, $M$ always halts such that $x \in L$ iff $M$ accepts $x$.

We construct 2 TMs, $M_1$ and $M_2$ such that both $M_1$ and $M_2$ on input $x$ will simulate $M$ on $x$. If $M$ on $x$ reaches the accepting halting state $H_a$, then $M_1$ on $x$ halts, but $M_2$ on $x$ goes to an infinite loop. Otherwise, $M_1$ on $x$ goes to an infinite loop and $M_2$ on $x$ halts.

Hence, $L = \{x | M_1 \text{ on } x \text{ halts}\}$
$L' = \{x | M_2 \text{ on } x \text{ halts}\}$

Thus, $L$ and $L'$ are both Turing-Acceptable.

$\Leftarrow$
Assume $L$ and $L'$ are Turing-Acceptable. Then, by definition there exists DTM $M_1$ and $M_2$ such that
$L = \{x | M_1 \text{ halts on } x\}$
$L' = \{x | M_2 \text{ halts on } x\}$
Now we construct a new DTM $M$ such that on any input $x$, $M$ simulates $M_1$ on $x$ and $M_2$ on $x$ one step at a time, in a round-robin fashion (i.e., run $M_1$ on $x$ for one step, then run $M_2$ on $x$ for one step, then run $M_1$ on $x$ for another step, then run $M_2$ on $x$ for another step, etc.) This technique is called “time slicing” in Operating Systems, and “dove tailing” in recursion theory.

By assumption one of the computations of $M_1$ on $x$ and $M_2$ on $x$ will halt. If $M_1$ on $x$ halts, then let $M$ halt at the halting state $H_a$. If $M_2$ on $x$ halts, then let $M$ halt on the halting state $H_r$. Hence, $L$ is decidable.

However not all Turing-Acceptable languages are decidable. In other words, there are languages $L$ such that $L$ is Turing-Acceptable but $L'$ is not Turing-Acceptable. Such languages are called Semidecidable.

If a language is Turing-Acceptable, it’s also called recursively enumerable (r.e).

What this means is that we can enumerate all the elements in $L$ one by one by a special TM $M$ that takes no input, runs forever, and prints all the elements in $L$ on its output tape.

The reason why this is true is the following:

Assume that $L$ is Turing-Acceptable. Then there exists a DTM $M_L$ such that $L = \{x | M_L \text{ on } x \text{ halts}\}$. Construct a DTM $M$ which takes no input and does the following:

$n = 0$

Stage $N$: Lexicographically generates the $1^{st}$ $n$ strings over the alphabet of $L$: $s_0$, $s_1$, $s_2$, ..., $s_{n-1}$.

For each string $s_i$, $M$ simulates $M_L$ on input $s_i$ for $n$ steps. If $M_L$ on $s_i$ halts, then this means that $s_i \in L$, and $M$ prints $s_i$ on its output tape.

Set $n = n+1$, and repeat.

Since if $x \in L$, then $M_L$ on $x$ will halt, $x$ will be printed by $M$ at some stage $N$.

We observe that the machine $M$ we constructed actually prints every $x \in L$ infinitely many times. We can modify $M$ so that $M$ will print each $x \in L$ exactly once (Homework problem).
Universal Turing Machine
And
The Halting Problem

TM’s we’ve seen so far only perform a specific task. Look at the computers we have. We know that they can perform many different tasks. These computers take programs as inputs. This motivates us to look at Universal TM’s which take a pair \(<m, x>\) as input, where \(m\) is a TM and \(x\) is an input of \(m\), and simulates \(m\) on \(x\).

Every alphabet can be encoded using a binary alphabet. One common encoding scheme is ASCII.

Halting Problem: M on an Universal TM.

Q: On input \(<m, x>\), does \(M\) on \(x\) halt?
A: This problem is not decidable.

We first look at an easier halting problem:
\(K = \{<m>| M\text{ on input }<m>\text{ halts}\}\)

**Theorem**
\(K\) is Turing-Acceptable but \(K\) is not decidable.

The reason why \(K\) is Turing-Acceptable is that we can construct a TM \(M_k\) such that \(M_k\) on input \(<m>\) simulates \(M_k\) on \(<m_k>\).
\(M_k\) halts if \(M_k\) on \(<m>\) halts, thus:
\(K = \{<m>| M_k\text{ halts on }<m>\}\).
In other words \(K\) is accepted by \(M_k\).

To see why \(K\) is not decidable, it suffices to show that \(K'\) is not Turing-Acceptable.

Assume that \(k'\) is Turing-Acceptable (proof by contradiction), then there exists TM \(M_0\) such that \(K'\) is accepted by \(M_0\) (i.e., \(K' = \{x | M_0\text{ on }x\text{ halts}\}\)).

Now let’s look at a particular string \(x=<m_0>\). Then we have \(<m_0> \in K'\) iff \(M_0\) on \(<m_0>\) halts iff \(<m_0> \in K\).
This is a contradiction: Hence our assumption is wrong (i.e., \(K'\) is not Turing-Acceptable).