1 Acceptable and Decidable Languages

Let $P$ be a program and $x$ be a string. What happens when we run $P$ on on input $x$. There are there possibilities. i) The program runs for ever, ii) The program outputs the word “ACCEPT” and halts, iii) The program does not output the word and “ACCEPT” and halts. Observe that these are the only three possibilities that can happen, and they are mutually exclusive.

A program $P$ accepts a string $x$, if $P$ on input $x$ halts and outputs the word “ACCEPT”, if $P$ on input $x$ halts and does not output the word “ACCEPT”. Given a program $P$, the language accepted by $P$ is

$$L(P) = \{x \mid P \text{ accepts } x\}.$$  

Given a program $P$ and a language $L$, we say $P$ accepts $L$, if $L = L(P)$. A language $L$ is Turing acceptable if there exists a program $P$ such that $P$ accepts $L$, i.e., $L = L(P)$. A language $L$ is Turing decidable if there exists a program that accepts $L$, and $P$ always halts.

If $L$ is Turing acceptable, then there exists a program $P$ such that: for any string $x$, if $x \in L$, then $P$ accepts $x$. If $x \notin L$, then there are two possibilities: 1) $P$ may halt and does not output “ACCEPT”, or 2) $P$ may run forever.

If $L$ is Turing decidable, then there exists a program $P$ such that: for any string $x$, if $x \in L$, then $P$ accepts $x$, else $P$ rejects $x$.

Useful bijections. Recall the following:

- There is a bijection from $\mathbb{N}$ to $\Sigma^*$.
- There is a bijection from $\mathbb{N}$ to $\mathbb{N} \times \mathbb{N}$.
- There is a bijection from $\mathbb{N}$ to $\mathbb{N} \times \Sigma^*$.
- There is a bijection from $\Sigma^*$ to $\Sigma^* \times \Sigma^*$.

For each of the above bijections, there is a program that computes that bijection. Think about such programs. We will will use those programs quite often. In the above, we can replace $\mathbb{N}$ with any set $S$ such that there is a bijection from $\mathbb{N}$ to $S$, and there is a program that computes that bijection.
Godel Numbers

Consider any program $P$. We can encode it as a string $w_P$ over $\{0, 1\}$. Note that any string in $\Sigma^*$ can be interpreted as a natural number. Thus the set of all programs is countable. This means there is a bijection from $\mathbb{N}$ to the set of all programs. Moreover, this bijection can be computed by a program. Fix your favorite such bijection, lets call it $g$. Now $g(0), g(1), \ldots$ are programs. For every program $P$ there is a number $e \in \mathbb{N}$ such that $g(e) = P$. We say that $e$ is the Godel number of $P$.

The function $g$ maps every natural number $e$ to a unique program. We denote this program with $P_e$. This enables us to view program as numbers and numbers as programs. Thus $P_1, P_2, \ldots$ is an enumeration of all programs.

Example’s of TA and TD Languages

A few examples of Turing Decidable languages.

\[
\{G \mid G \text{ is a graph with a Hamiltonian cycle}\}.
\]

\[
\{n \mid n \text{ is a prime}\}.
\]

\[
\{(e, x, t) \mid P_e \text{ accepts } x \text{ in } t \text{ steps}\}.
\]

Here is an example of a Turing acceptable language.

\[
\{\langle e, x \rangle \mid P_e \text{ accepts } x \}.
\]

Here is an algorithm for the above language.

1. Input $\langle e, x \rangle$.
2. Run $P_e$ on $x$.
3. If $P_e$ accepts $x$, then then output “ACCEPT”.
4. HALT.

If $\langle e, x \rangle$ belongs to the language, then $P_e$ accepts $x$. Thus, the above algorithm accepts in Step 3. If $\langle e, x \rangle$ does not belong to the language, then either $P_e$ rejects $x$ or $P_e$ runs for ever. In the former case, the condition in step 3 is not satisfied, thus the program reaches Step 4 and halts. In this case, the algorithm does not output “ACCEPT”. This the algorithm rejects $\langle e, x \rangle$. In the latter case the above algorithm runs forever. Thus the language is Turing acceptable.

Show that the following languages are Turing acceptable.

\[
\{e \mid M_e \text{ accepts } e \}.
\]

\[
\{(p, n) \mid p \text{ is a polynomial in } n \text{ variables, and } p \text{ has integer solutions}\}.
\]

Later in the course, we show that these languages are not Turing decidable.

Consider the following language:

\[
L_{ne} = \{e \mid L(P_e) \neq \emptyset\}.
\]
A number $e$ belongs to $L_{ne}$ if and only if the program $P_e$ accepts at least one string. We will now show that the $L_{ne}$ is Turing acceptable. For this we have to exhibit an algorithm that accepts $L_{ne}$. Consider the following algorithm.

1. Input $e$.
2. $x = \lambda$.
3. Run $P_e$ on $x$.
4. If $P_e$ accepts $x$, then output “ACCEPT” and exit loop.
5. Else, $x = x + 1$, and GOTO Step 3.

At first glance, you may think that the above algorithm accepts $L_{ne}$. However, that is not the case. Think about it.

Here a correct algorithm.

1. Input $e$.
2. Set $n = 1$.
3. Run $P_e$ on first $n$ numbers: $n$ steps on each number.
4. if $P_e$ accepts any number within $n$ steps, then output “ACCEPT and exit loop.
5. Else, $n = n + 1$ and GOTO Step 3.

We claim that the above algorithm accepts a number $e$ if and only if $e \in L_{ne}$. Note that the above algorithm outputs the word ”ACCEPT” only when it discovers that $P_e$ accepted a string. Thus if $e \notin L_{ne}$, then $P_e$ does not accept any string. Thus the above algorithm never outputs “ACCEPT” on input $e$. Thus the above algorithm does not accept $e$.

Now assume that $e \in L_{ne}$. We have to show that the above algorithm accepts $L_{ne}$. If $e \in L_{ne}$, then there exist a number $m$ that $P_e$ accepts. Observe that if $P_e$ accepts $m$, then there is a finite number $t$ such that $P_e$ accepts $m$ in $t$ steps. Let $u = \max\{m, t\}$.

Observe that, in the above algorithm, the value of $n$ keeps on incrementing till it outputs “ACCEPT”. Thus when we run the above algorithm on input $e$, the value of $n$ must reach $u$ (unless the algorithm has already accepted. In that case we are done). When the value of $n$ reaches $u$, then the algorithm considers first $n = u$ strings, and runs $P_e$ on each of them for exactly $n = u$ steps. Since $m \leq u$, the algorithm runs $P_e$ on number $m$. Since $t \leq u$, the above algorithm runs $P_e$ on input $m$ for at $u \geq t$ steps. Since $P_e$ accepts $x$ in $t$ steps, the above algorithm discovers this, thus outputs “ACCEPT”. Thus the above algorithm accepts $e$ when $e \in L_{ne}$.

Here is an another algorithm that accepts $L_{ne}$. Let $g$ be a bijection from $\mathbb{N}$ to $\mathbb{N} \times \mathbb{N}$. Observe that there is a bijection that is computable by a program.

1. Input $e$.
2. Set $r = 0$.
3. Let $g(r) = (x, t)$. (Compute the value of $g$ on $r$ to obtain $x$ and $t$.)
4. Run $P_e$ on $x$ for $t$ steps. If $P_e$ accepts $x$ within $t$ steps, then output "ACCEPT" and exit loop.

5. $r = r + 1$. Goto Step 3.

We claim that the above algorithm also accepts the language $L_{ne}$. Suppose $e \in L_{ne}$. This implies that there is a number $x$ that is accepted by $P_e$. Thus, there is a finite natural number $t$ such that $P_e$ accepts $x$ within $t$ steps. There is a unique number $u$ for which $g(u) = \langle x, t \rangle$.

Observe that the value of $r$ in the above algorithm keeps on incrementing till the program halts. Thus the value of $r$ reaches $u$ at some point of time (unless the algorithm has accepted, which is a good case for us). When $r = u$, the $g(r) = \langle x, t \rangle$. Thus the algorithm runs $P_e$ on input $x$ for $t$ steps. Since $P_e$ accepts $x$ within $t$ steps, the above algorithm outputs "ACCEPT". Thus the above algorithm accepts $e$ when $e \in L_{ne}$. It is easy to see that when $e \notin L_{ne}$, the above algorithm does not output "ACCEPT".

2 Closure Properties

Let $A$ and $B$ be two Turing decidable languages. Then $A \cup B$ is also Turing decidable. We now formally prove this.

Since $A$ is Turing decidable, there exists a program $P$ such that $P$ always halts and accepts $A$. Since $B$ is Turing decidable, there is a program $Q$ such that $Q$ always halts and accepts $B$. Consider the following algorithm.

1. Input $x$.
2. Run $P$ on $x$.
3. If $P$ accepts $x$, then output "ACCEPT."
4. else, then run $Q$ on $x$.
5. If $Q$ accepts $x$, then output "ACCEPT."

We claim that the above algorithm always halts and accepts $A \cup B$. Suppose $x \in A \cup B$. Then $x$ is in $A$ or in $B$. First consider the case $x$ is in $A$. In this case, $P$ accepts $x$. Thus the above algorithm accepts in Step 3. Now consider the case $x$ is not $A$ but in $B$. Since $x$ is not $A$, $P$ on input $x$ must reject $x$. Thus the above algorithm reaches Step 4, and runs $Q$ on $x$. Since $x \in B$, $Q$ accepts $x$. Thus the above algorithm accepts.

Suppose $x \notin A \cup B$. Then $x \notin A$ and $x \notin B$. Thus both $P$ and $Q$ reject $x$. This means both $P$ and $Q$ will halt. Thus the above algorithm halts and does not output "ACCEPT".

We can show that if $A$ and $B$ are Turing acceptable, then $A \cup B$ is also Turing acceptable. To do this, we need a slightly different idea, as the above idea does not work. Think Why.

Since $A$ is Turing acceptable, there exists a program $P$ such that $P$ that accepts $A$. Since $B$ is Turing acceptable, there is a program $Q$ that accepts $B$. Consider the following algorithm.

1. Input $x$.
2. Run $P$ on input $x$ and $Q$ on input $x$ in parallel.
3. If one of the programs accept, stop the other program and ACCEPT.

Suppose \( x \in A \cup B \). Then \( x \) is in at least one of \( A \) or \( B \). So at least one of the programs \( P \) and \( Q \) accept \( x \). Thus the above algorithm reaches Step 3, and in this step the it accepts.

Suppose \( x \notin A \cup B \). Then \( x \) is neither in \( A \) nor in \( B \). Thus neither \( P \) nor \( Q \) accept \( x \). If one of these program run forever, then the above algorithm also runs forever. If both of them reject \( x \), then the above algorithm halts and does not output accept. Thus \( A \cup B \) is Turing acceptable.

We can show that if \( A \) is Turing decidable, then \( \overline{A} \) is also Turing decidable. Try to prove. What about Turing acceptable languages? If \( A \) is Turing acceptable, then is \( \overline{A} \) Turing acceptable?

Finally we conclude this section by showing that if \( A \) and \( \overline{A} \) are Turing Acceptable, then \( A \) is Turing decidable. If \( A \) is Turing Acceptable, then there is a program \( P \) that accepts \( A \). Since \( \overline{A} \) is also Turing Acceptable, there is a program \( Q \) that accepts \( \overline{A} \). Consider the following program: On input \( x \), run \( P(x) \) and \( Q(x) \) in parallel. If \( P(x) \) accepts, then stop the simulation of \( Q \), output “ACCEPT” and stop. If \( Q(x) \) accepts, then stop the simulation of \( P \), output “REJECT” and stop. It is easy to that is program accepts \( A \) and always halts.