We mentioned earlier that another term that is used for decidable is recursive, and another term that is used for semi-decidable is recursively enumerable. We now explain the enumerable part of recursively enumerable.

**Definition 2.4.** Let \( L \) be an infinite language on an alphabet \( \Sigma \). An enumeration algorithm for \( L \) is a computer program, \( M \), (written in your favorite language) which runs on an idealized computer and does the following:

- \( M \) takes no input.
- \( M \) never halts.
- As it runs \( M \) is progressively outputting a list of strings on \( \Sigma \).
- For all strings \( w \in \Sigma^* \), \( w \in L \) iff \( w \) is one of the strings which is eventually output from \( M \).

**Definition 2.5.** A language \( L \) is algorithmically enumerable iff there exists an enumeration algorithm for \( L \).

**Lemma 2.6.** Let \( L \) be an infinite language. Then \( L \) is semi-decidable iff \( L \) is algorithmically enumerable.

**Proof.** First assume that \( L \) is algorithmically enumerable. Let \( M \) be an enumeration algorithm for \( L \). We will now describe an algorithm \( M_1 \) which is a semi-acceptor for \( L \):

1. input(w);
2. start running \( M \);
3. if \( M \) ever outputs \( w \) then accept and halt;

If \( w \in L \) then \( M \) will eventually output \( w \) and so \( M_1 \) will eventually accept \( w \) and halt. If \( w \notin L \) then \( M \) will never output \( w \) and so \( M_1 \) will never halt. Thus \( M_1 \) is a semi-acceptor for \( L \).

Conversely, assume now that \( L \) is semi-decidable. Let \( M_1 \) be a semi-acceptor for \( L \). We will now describe an enumeration algorithm, \( M \), for \( L \). First we consider an enumeration of all of the strings in \( \Sigma^* \). There are many ways to do this and it doesn’t matter which one you use. For example if \( \Sigma = \{a, b\} \) you could use an enumeration like: \( w_0 = \lambda, w_1 = a, w_2 = b, w_3 = aa, w_4 = ab, w_5 = ba, w_6 = bb, w_7 = aab, w_8 = aba, w_9 = abb, w_{10} = bba, \ldots \)

Now here is the enumeration algorithm \( M \). Notice that the main structure of \( M \) is an infinite loop.

For \( i := 1 \) to infinity

For \( j := 0 \) to \( i \)

1. Run algorithm \( M_1 \) on input \( w_j \) for \( i \) steps;
2. If \( M_1 \) accepts \( w_j \) within \( i \) steps then output \( w_j \)

Let \( w \in \Sigma^* \). Then \( w = w_i \) for some \( i \). If \( w \in L \) then there is some \( j \) such that \( M_1 \) accepts \( w_i \) within \( j \) steps. Thus \( w = w_i \) will eventually be output by \( M \). If \( w \notin L \) then there is no such \( j \) and \( w \) will never be output by \( M \). Thus \( M \) is an enumeration algorithm for \( L \).
3 Closure Properties

Theorem 3.1. The family of decidable languages is closed under
(a) complimentation.
(b) union.
(c) intersection.

Proof. We proved (a) in Lemma 1.9.

We now prove (b). Let $L_1$ and $L_2$ be decidable. We will show that $L_1 \cup L_2$ is decidable. Let $M_1$ be an acceptor for $L_1$, and let $M_2$ be an acceptor for $L_2$. We will now describe an acceptor, $M$, for $L_1 \cup L_2$.

(1) input(w);
(2) run $M_1$ on input $w$;
(3) if $M_1$ accepts $w$ then accept and halt;
(4) run $M_2$ on input $w$;
(5) if $M_2$ accepts $w$ then accept and halt;
(6) reject and halt;

Since $M_1$ and $M_2$ always halt, $M$ always halts.

Part (c) is proved using parts (a) and (b) and DeMorgan’s Law, just as we did with the family of Regular Languages.}

Theorem 3.2. The family of semi-decidable languages is closed under
(a) union.
(b) intersection.

But not under complimentation.

Proof. Let $L_1$ and $L_2$ be semi-decidable. Let $M_1$ and $M_2$ be semi-acceptors for $L_1$ and $L_2$ respectively.

First we prove (a). We will now define a semi-acceptor, $M$, for $L_1 \cup L_2$. First consider the algorithm from the proof of part (b) of the previous theorem. It is interesting to see why this does not work here. Suppose $w$ is in $L_2$ but not in $L_1$. Then $w \in L_1 \cup L_2$ and so we are supposed to accept $w$. But since $w \notin L_1$ it is possible that $M_1$ does not halt on input $w$. But this means that the call to $M_1$ in step (2) of the algorithm may never return and so we never make it to step (4) of the algorithm. The solution to this problem is to run $M_1$ and $M_2$ simultaneously. As we have done before, we simulate parallel processing using time-slicing. Here is the algorithm which does work:

(1) input(w);
(2) run $M_1$ as a subroutine on the input $w$, but only for 1 step;
(3) if $M_1$ accepts $w$ in its first step then accept and halt;
(4) run $M_2$ as a subroutine on the input $w$, but only for 1 step;
(5) if $M_2$ accepts $w$ in its first step then accept and halt;
(6) run the next step of algorithm $M_1$ on the input $w$;
(7) if $M_1$ accepts $w$ in this next step then accept and halt;
(8) run the next step of algorithm $M_2$ on the input $w$;
(9) if $M_2$ accepts $w$ in this next step then accept and halt;
(10) goto (6);
If $w \in L_1$ or $w \in L_2$ then the test in step (7) or the test in step (9) will eventually evaluate to true and we will accept and halt.

Next we prove (b). We will now define a semi-acceptor, $M$, for $L_1 \cap L_2$. It is interesting to note that here we don’t need to use parallel processing. We can call $M_1$ and then $M_2$.

(1) input($w$);
(2) run $M_1$ on input $w$;
(3) if $M_1$ accepts $w$ then
    (3a) run $M_2$ on input $w$;
    (3b) if $M_2$ accepts $w$ then accept and halt;
(6) reject and halt;

If either $M_1$ or $M_2$ doesn’t halt then our algorithm will not halt. But this is ok, because if either $M_1$ or $M_2$ doesn’t halt then $w \notin L_1 \cap L_2$.

Finally we prove that the family of semi-decidable languages is not closed under complimentation. Well, actually we already know this. Let $L_1 = \{ w : w \in L(M_w) \}$, and let $L = \{ w : w \notin L(M_w) \}$. Then $L$ is the compliment of $L_1$. But as we saw in the previous section, $L_1$ is semi-decidable, and $L$ is not. \qed