Computability Crib Sheet

This is a “quick reference” sheet for the definitions, theorems and examples in computability that we have covered. When you meet a proposition, theorem, or claim below, try to recall why it is true!

1 Conventions

We work over alphabet Σ. (Think Σ = {0, 1}). A string is a member of Σ*, and ε is the empty string, which is the only string of length 0. A language is a subset of Σ*. The complement of a language L is \( L = \{ x \in \Sigma^* : x \not\in L \} = \Sigma^* - L \).

TMs can be encoded as strings, and can form the inputs to TMs. We adopt a convention under which every string in Σ* encodes some machine. (Specifically, we can adopt the convention that any string which does not look like an encoding of a valid machine encodes a fixed, always accepting TM.) The universal machine \( U \) takes as input \( \langle M, x \rangle \), where \( M \) is a TM and \( x \in \Sigma^* \), and simulates the computation of \( M \) on \( x \). That is, \( U(\langle M, x \rangle) \) accepts iff \( M(x) \) accepts, \( U(\langle M, x \rangle) \) rejects iff \( M(x) \) rejects, and \( M(x) \) halts with output \( y \) iff \( U(\langle M, x \rangle) \) halts with output \( y \).

2 Decidable, r.e. and co-r.e. languages

**Definition 2.1** We say that TM \( M \) decides language \( L \) if for all \( x \in \Sigma^* \) it is the case that

- If \( x \in L \) then \( M(x) \) accepts
- If \( x \not\in L \) then \( M(x) \) rejects.

We say that TM \( M \) recognizes language \( L \) if for all \( x \in \Sigma^* \) it is the case that

- If \( x \in L \) then \( M(x) \) accepts
- If \( x \not\in L \) then \( M(x) \) either rejects or loops.

Recall that “loops” is synonymous with “does not halt.” It is important to note that if TM \( M \) decides language \( L \) then \( M(x) \) halts for all \( x \in \Sigma^* \). On the other hand if \( M \) recognizes \( L \) then \( M(x) \) halts if \( x \in L \), but when \( x \not\in L \) it may or may not halt.

**Definition 2.2** A language \( L \) is decidable (also called recursive) if there exists a TM \( M \) that decides it. \( L \) is recognizable (also called recursively enumerable, abbreviated r.e.) if there exists a
TM $M$ which recognizes it. $L$ is **co-recognizable** (also called co-r.e.) if there exists a TM $M$ which recognizes $\overline{L}$, or, in other words, if $\overline{L}$ is r.e. 

These are classes of languages, for which we will use the following notations—

\[
\begin{align*}
D &= \{ L : L \text{ is decidable} \} \\
RE &= \{ L : L \text{ is r.e.} \} \\
coRE &= \{ L : L \text{ is co-r.e.} \}
\end{align*}
\]

Let us now look at some properties and alternative characterizations of these classes.

**Proposition 2.3 (Closure Properties)**

1. If $L$ is decidable then so is $\overline{L}$.
2. If $L_1, L_2$ are decidable then so is $L_1 \cup L_2$.
3. If $L_1, L_2$ are decidable then so is $L_1 \cap L_2$.
4. If $L_1, L_2$ are r.e. then so is $L_1 \cup L_2$.
5. If $L_1, L_2$ are r.e. then so is $L_1 \cap L_2$.

The first property can be said as: “The class of decidable languages is closed under complement.” The second can be said as: “The class of decidable languages is closed under union.” And so on. For this reason, these things are called “closure properties”.

If $L$ is decidable then it is both r.e. and co-r.e. Moreover, this property characterizes decidability:

**Proposition 2.4** $L$ is decidable if and only if $L$ is both r.e. and co-r.e. That is, $D = RE \cap coRE$.

**Definition 2.5** For any TM $M$ we let

\[
L(M) = \{ x \in \Sigma^* : M(x) \text{ accepts} \}.
\]

$L(M)$ is called the **language of $M$**.

Note that $L(M)$ is a r.e. language for every TM $M$, because $M$ itself recognizes $L(M)$. Conversely, from the definition of r.e. it is easy to see that if $A$ is a r.e. language then it equals $L(M)$ for some TM $M$. Hence we have the following:

**Proposition 2.6** Language $A$ is r.e. if and only if there is a TM $M$ such that $A = L(M)$.

R.e. languages can be characterized as those for which it is possible to provide proofs of membership. This looks ahead to $NP$.

**Definition 2.7** We say that TM $V$ **verifies** language $L$ if $V(x,y)$ halts for all inputs $x, y \in \Sigma^*$, and also for all $x \in \Sigma^*$—
(1) If \( x \in L \) then there exists \( y \in \Sigma^* \) such that \( V(x, y) \) accepts.

(2) If \( x \not\in L \) then for all \( y \in \Sigma^* \), \( V(x, y) \) rejects.

We say that \( L \) is verifiable if there exists a TM that verifies it.

**Proposition 2.8 (Verifier based, or proof theoretic, characterization of r.e. languages)**

A language \( L \) is r.e. if and only if it is verifiable.

### 3 Reductions and undecidability

We now move to undecidability. Let—

\[
H = \{ \langle M, x \rangle : M \text{ is a TM and } M(x) \text{ halts} \} \quad \text{— The halting problem}
\]

\[
BTH = \{ \langle M \rangle : M \text{ is a TM and } M(\epsilon) \text{ halts} \} \quad \text{— The blank tape halting problem.}
\]

It is easy to see that \( H \) is r.e. Its undecidability is shown by diagonalization. In summary—

**Theorem 3.1** \( H \) is r.e. but not decidable.

It follows that \( H \) is not co-r.e. and \( \overline{H} \) is not r.e. (Why? Use Proposition 2.4 to justify this claim.)

**Definition 3.2** A function \( f : \Sigma^* \to \Sigma^* \) is computable (or recursive) if there is a TM \( M \) such that \( M(x) = f(x) \) for all \( x \in \Sigma^* \).

That is, there is a TM which halts on all inputs \( x \in \Sigma^* \), and its output, when given input \( x \), is \( f(x) \).

**Definition 3.3** Let \( A, B \subseteq \Sigma^* \) be languages. A function \( f : \Sigma^* \to \Sigma^* \) is a mapping reduction (also called a many-one reduction) of \( A \) to \( B \) if it is computable and also for all \( x \in \Sigma^* \)—

(1) If \( x \in A \) then \( f(x) \in B \)

(2) If \( x \not\in A \) then \( f(x) \not\in B \).

We write \( A \leq_m B \) if there is a many-one reduction of \( A \) to \( B \).

The following is often useful:

**Proposition 3.4** If \( A \leq_m B \) then \( \overline{A} \leq_m \overline{B} \).

Many-one reducibility is considered because of the following nice properties. We see that “easiness inherits downwards” and “hardness inherits upwards.”

**Proposition 3.5** Suppose \( A \leq_m B \). Then—

(1) If \( B \) is decidable (resp. r.e., co-r.e.) then \( A \) is decidable (resp. r.e., co-r.e.)
(2) If $A$ is not decidable (resp. not r.e., not co-r.e.) then $B$ is not decidable (resp. not r.e., not co-r.e.).

The second part of the above proposition makes many-one reducibility a central tool for proving undecidability results. Namely, to show that a given language $B$ is not decidable it suffices to show $A \leq_m B$ where $A$ is a language we already know is non-decidable, such as $A = H$ or $A = \overline{H}$. Similarly to show that $B$ is not r.e. it suffices to show that $A \leq_m B$ for some $A$ which is not r.e., and similarly for co-r.e. In particular we have seen that—

**Proposition 3.6** $H \leq_m \text{BTH}$. 

Since BTH is easily seen to be r.e. we can apply Proposition 3.5 Part (2) to conclude that—

**Proposition 3.7** BTH is r.e. but not decidable.

Based on the above, BTH can be used to show undecidability results. This is often more convenient than using $H$.

**Recommended methodology:** Suppose we are given a language $L$. Then

1. If you want to show that $L$ is not r.e., show that $BTH \leq_m \overline{L}$.
2. If you want to show that $L$ is not co-r.e., show that $BTH \leq_m L$.
3. If you want to show that $L$ is undecidable, pick and show the appropriate one of the above two.\(^1\)

It is worth making sure you understand why these reductions do the job. We are using Part (2) of Proposition 3.5 with $A = \text{BTH}$. We know that $A$ is not co-r.e.. Thus in the first case above, the Proposition tells us that $\overline{L}$ is not co-r.e., or, equivalently, that $L$ is not r.e., as desired. In the second case, it tells us that $L$ is not co-r.e. again as desired. We will provide examples of how this is used later.

Finally, let us discuss Rice’s theorem. We did not talk about this in class so it is only for your interest.

Let $\mathcal{C} \subseteq \text{RE}$ be a subclass of the class of r.e. languages. (That is, $\mathcal{C}$ is a collection of languages, each of which is r.e.. Think of it as capturing some “property” of r.e. languages.) We let

$$L_\mathcal{C} = \{ \langle M \rangle : M \text{ is a TM and } L(M) \in \mathcal{C} \}$$

be the set of all TMs $M$ such that $L(M)$ (the language of $M$) is in the class $\mathcal{C}$. Notice that $L_{\emptyset} = \emptyset$ and $L_{\text{RE}} = \Sigma^*$, and these are decidable languages. Rice’s theorem says that these are the only possible choices for $\mathcal{C}$ for which $L_\mathcal{C}$ is decidable.

**Theorem 3.8 (Rice’s theorem)** Suppose $\mathcal{C} \subseteq \text{RE}$ is a subclass of the class of r.e. languages such that $\mathcal{C} \neq \emptyset$ and $\mathcal{C} \neq \text{RE}$. Then the language $L_\mathcal{C}$ is undecidable. 

\(^1\)Observe that it is important which choice you make. If $L$ happens to be co-r.e. you will NOT succeed if you try to show that $BTH \leq_m L$, because this would imply that $L$ is not co-r.e. You have to do the other one.
Rice’s theorem is often a simple way of seeing that a language is undecidable. For example, it is easily applied to see that the following are undecidable—

\[ L = \{ \langle M \rangle : M \text{ is a TM and } L(M) \text{ is finite} \} \]
\[ L = \{ \langle M \rangle : M \text{ is a TM and } \exists x \in \Sigma^* \text{ such that } M(x) \text{ halts} \} \]
\[ L = \{ \langle M \rangle : M \text{ is a TM and } L(M) \text{ is regular} \} \].

Why? Check by providing in each case a (non-trivial) class \( C \subseteq \text{RE} \) such that \( L = L_C \).

4 Examples

We exemplify the determination of the status of languages using the above methodology. The problem is that we will be given some language and have to put it in one of the following classes:

1. decidable
2. r.e. but not co-r.e.
3. co-r.e. but not r.e.
4. neither r.e. nor co-r.e.

4.1 Example 1

The language we consider is

\[ L_1 = \{ \langle M' \rangle : M' \text{ is a TM which loops on all inputs} \} \].

A useful first step is to write down the complement:

\[ \overline{L_1} = \{ \langle M' \rangle : M' \text{ is a TM and } \exists x \in \Sigma^* \text{ such that } M'(x) \text{ halts} \} \].

It may be worth expanding here on a technical point associated to the taking of complements above. Strictly speaking, the complement of \( L_1 \) is

\[ \begin{cases} w \in \Sigma^* : & \text{Either } w \text{ does not encode a TM, or} \\
& w = \langle M' \rangle \text{ where } M' \text{ is TM and } \exists x \in \Sigma^* \text{ such that } M'(x) \text{ halts} \end{cases} \].

However, the need to consider strings that don’t encode TMs is a technicality that changes nothing about the intrinsic properties of the problem in question, and we would rather avoid having to deal with it and simply view the complement as what we first wrote above. It is in order to do this rigorously that we adopted the convention, mentioned in Section 1 above, that \( \text{all} \) strings \( w \in \Sigma^* \) encode some TM. (We can make this true by saying that if \( w \) does not “really” encode a TM, we simply define it as the valid encoding of some fixed TM, like the always halting TM.) Now, the two definitions of the complement of \( L_1 \) above coincide. In the future, we will assume such convenient conventions have been adopted without mention. Let us now return to our problem.

Intuition:

This discusses the kind of thought process you might go through when you start addressing this problem. I want to clearly distinguish it from the actual solution because this is NOT something you should write in your solution. Below, we will see what you would actually write based on this reasoning.

Do you think \( L_1 \) is decidable? To answer this, we might first ask ourselves if we could write a program that given \( \langle M' \rangle \) would (halt and) accept if \( M'(x) \) loops for all \( x \), while (halting and) rejecting if \( M'(x) \) halts for some \( x \). Pretty much the only technique we could use in our program
is to have it run \( M' \) on different inputs, possibly using interleaving. If our program finds an \( x \) for which \( M'(x) \) halts, it may correctly reject and halt, but as long as it does not, we have no way to be sure that further search will not find one. So this method, at least, fails to decide \( L \). Based on this, we shall guess that \( L_1 \) is co-r.e. but not decidable.

It is important to realize that the above is a guess, not a proof. But intuitively, when the “obvious” decision procedure fails, it is likely none exists, and we should move to proving that.

Is \( L_1 \) r.e.? I think the easiest way to think about this is via Proposition 2.8. Can you design a verifier for \( L_1 \)? This amounts to asking whether someone who is all-powerful and knows whether or not a given \( \langle M' \rangle \) is in \( L_1 \) can convince you of this by coming up with a “proof” or “certificate” \( y \) for the membership of \( \langle M' \rangle \) in \( L_1 \). But it is hard to certify that a TM \( M' \) loops on all inputs. We guess that \( L_1 \) is not r.e..

Finally, is \( L_1 \) co-r.e.? This amounts to asking whether \( L_1 \) is r.e.. Again, use Proposition 2.8. Can you design a verifier for \( L_1 \)? This amounts to asking whether someone who is all-powerful and knows whether or not a given \( \langle M' \rangle \) is in \( L_1 \) can convince you of this by coming up with a “proof” or “certificate” \( y \) for the membership of \( \langle M' \rangle \) in \( L_1 \). This time you can see it is possible. The “prover” can provide \( x \) such that \( M'(x) \) halts. But that’s not enough; how do we check that the claim \( M'(x) \) halts is true? The prover must also provide the number of time steps this computation takes to halt.

So, our guess is that \( L_1 \) falls in category (3). Now we have to actually prove this. We will first prove that \( L_1 \) is co-r.e.. But we must also prove it is not r.e.. Referring to the above methodology, we must show that \( \text{BTH} \leq_m L_1 \).

Solution:

This is what you would actually be expected to write as your answer. (I stress again that what is above is not the solution; don’t show it on your writeup. Your writeup should be more along the lines of what follows.)

Claim 4.1 \( L_1 \) is co-r.e..

Proof: We need to show that \( \overline{L_1} \) is r.e.. We will use Proposition 2.8. Namely we provide a verifier \( V_1 \) that works as follows:

\[
V_1(\langle M' \rangle, y) \\
\text{Parse } y \text{ as a pair } (x, 1^t) \\
\text{Run } M'(x) \text{ for } t \text{ steps} \\
\text{If it halts accept else reject}
\]

Now we must check that this verifier meets the conditions of Proposition 2.8.

First suppose \( \langle M' \rangle \in \overline{L_1} \). We want to show that \( \exists y \) such that \( V_1(\langle M' \rangle, y) \) accepts. By definition of \( \overline{L_1} \) there is an \( x \) such that \( M'(x) \) halts. Set \( y = (x, 1^t) \) where \( t \) is the number of steps taken by the computation \( M'(x) \) to halt. By construction of \( V_1 \) we can see that \( V_1(\langle M' \rangle, y) \) accepts. So we have shown that \( \exists y \) such that \( V_1(\langle M' \rangle, y) \) accepts. (Note that this argument does not require that we “find” or “compute” \( y \) in some constructive sense. We need only prove it exists.)
Now suppose \( \langle M' \rangle \not\in L_1 \). We want to show that \( \forall y \) it must be that \( V_1(\langle M' \rangle, y) \) rejects. Consider some arbitrary \( y = (x, 1^t) \). The computation \( M'(x) \) does not halt in \( t \) steps; indeed, we know that \( M'(x) \) loops because \( \langle M' \rangle \in L_1 \). So \( V_1(\langle M' \rangle, y) \) will reject as desired. 

Note: An alternative proof of this claim is to specify a TM \( M \) that recognizes \( \overline{L_1} \). It takes input \( \langle M' \rangle \) and runs \( M' \) on all inputs \( x \) in an interleaving fashion, accepting if \( M' \) halts on some input. The proof that \( L_1 \) is not r.e. follows from the following claim.

**Claim 4.2** \( \text{BTH} \leq_m \overline{L_1} \).

**Proof:** As per Definition 3.3 we need to specify a computable function \( f \) that takes as input the encoding \( \langle M \rangle \) of a TM \( M \) and outputs the encoding \( \langle M' \rangle \) of a TM \( M' \) such that the following is true:

1. If \( \langle M \rangle \in \text{BTH} \) then \( \langle M' \rangle \in \overline{L_1} \)
2. If \( \langle M \rangle \not\in \text{BTH} \) then \( \langle M' \rangle \not\in \overline{L_1} \).

Here is the specification of the TM \( M' \) associated to \( M \):

\[
M'(x) \\
\text{Run } M(\varepsilon) \\
\text{If this halts then halt}
\]

First we need to check that the function \( f \) defined by \( f(\langle M \rangle) = \langle M' \rangle \) is computable. This is true because we can write a program that given \( \langle M \rangle \) will output the \( \langle M' \rangle \). All this program has to do to produce \( \langle M' \rangle \) is put a small “wrapper” around the code of \( M \).

Now we need to check the two properties (1),(2) above.

Begin with (1). We have:

\[
\langle M \rangle \in \text{BTH} \quad \Rightarrow \quad M(\varepsilon) \text{ halts} \quad \text{(By def. of BTH)} \\
\quad \Rightarrow \quad M'(x) \text{ halts for all } x \quad \text{(By def. of } M' \text{ above)} \\
\quad \Rightarrow \quad M'(x) \text{ halts for some } x \\
\quad \Rightarrow \quad \langle M' \rangle \in \overline{L_1} \quad \text{(By def. of } \overline{L_1} \text{)}.
\]

Now for (2) we have:

\[
\langle M \rangle \not\in \text{BTH} \quad \Rightarrow \quad M(\varepsilon) \text{ loops} \quad \text{(By def. of BTH)} \\
\quad \Rightarrow \quad M'(x) \text{ loops for all } x \quad \text{(By def. of } M' \text{ above)} \\
\quad \Rightarrow \quad \langle M' \rangle \in L_1 \quad \text{(By def. of } L_1 \text{)} \\
\quad \Rightarrow \quad \langle M' \rangle \not\in \overline{L_1} \quad \text{(By def. of } \overline{L_1} \text{)}.
\]

This completes the proof. 

4.2 Example 2

The language we consider is

\[ L_2 = \{ \langle M' \rangle : M \text{ is a TM and } L(M') \text{ is decidable} \} . \]

A useful first step is to write down the complement:

\[ \overline{L}_2 = \{ \langle M' \rangle : M \text{ is a TM and } L(M') \text{ is undecidable} \} . \]

Recall that as per Definition 2.5,

\[ L(M') = \{ x \in \Sigma^* : M'(x) \text{ accepts} \} . \]

Intuition:

This discusses the kind of thought process you might go through when you start addressing this problem. I want to clearly distinguish it from the actual solution because this is NOT something you should write in your solution. Below, we will see what you would actually write based on this reasoning.

First, try to parse the question. It says that someone gives you \( \langle M' \rangle \) and asks you whether \( L(M') \) is decidable. Meaning, given \( \langle M' \rangle \), can you determine whether there is some TM, call it \( M^* \), that depends on \( M' \), and that decides the set \( L(M') \)? (We know there is one that recognizes this set, but is there one that decides it?)

Do you think \( L_2 \) is decidable? It is hard to see how one could write a program that given \( \langle M' \rangle \) would halt and accept if an \( M^* \) as above exists, while halting and rejecting if such an \( M^* \) does not exist. We guess that \( L_2 \) is not decidable. (This is not a proof, just a guess.)

Is \( L_2 \) r.e.? This amounts to asking: can you design a verifier for \( L_2 \)? Suppose an all-powerful “prover” knows whether or not a given \( \langle M' \rangle \) is in \( L_2 \). Can they convince you of this by coming up with a “proof” or “certificate” \( y \) for the membership of \( \langle M' \rangle \) in \( L_2 \)? They could provide \( \langle M^* \rangle \) where \( M^* \) is a TM that decides \( L(M') \). But how could a verifier check that \( M^* \) really decided \( L(M') \)? And what could a prover provide to prove this? It is hard to find any certificate. So we guess that \( L_2 \) is not r.e..

Finally, is \( L_2 \) co-r.e.? This amounts to asking whether \( \overline{L}_2 \) is r.e. Can you design a verifier for \( \overline{L}_2 \)? This amounts to asking whether our all-powerful “prover,” knowing whether or not a given \( \langle M' \rangle \) is in \( \overline{L}_2 \), can convince you of this by coming up with a “proof” or “certificate” \( y \) for the membership of \( \langle M' \rangle \) in \( \overline{L}_2 \). They need to convince us that \( L(M') \) is undecidable. It is hard to imagine any certificate for such a claim. We guess that \( \overline{L}_2 \) is not r.e..

So, our guess is that \( L_2 \) falls in category (4). Now we have to actually prove this. Referring to the above methodology, we must show two reductions.

Solution:

This is what you would actually be expected to write as your answer. (I stress again that what is above is not the solution; don’t show it on your writeup. Your writeup should be more along the lines of what follows.)

The proof that \( L_2 \) is not r.e. follows from the following claim.
Claim 4.3  $\text{BTH} \leq_m L_2$.

**Proof:** As per Definition 3.3 we need to specify a computable function $f$ that takes as input the encoding $\langle M \rangle$ of a TM $M$ and outputs the encoding $\langle M' \rangle$ of a TM $M'$, such that the following is true:

1. If $\langle M \rangle \in \text{BTH}$ then $\langle M' \rangle \in L_2$.
2. If $\langle M \rangle \notin \text{BTH}$ then $\langle M' \rangle \notin L_2$.

Here is the specification of the TM $M'$ associated to $M$. We write its input as $\langle N \rangle$ because we want to view this input as being a TM.

\[ M' (\langle N \rangle) \]
\[ \text{Run } M (\varepsilon) \]
\[ \text{If this halts then Run } N (\varepsilon) \]
\[ \text{If this halts then accept} \]

We can observe that it is possible to write a program that given the code $\langle M \rangle$ of $M$ spits out the code $\langle M' \rangle$ of $M'$. This means the function $f$ defined by $f(\langle M \rangle) = \langle M' \rangle$ is computable.

Now we need to check the two properties (1), (2) above.

Begin with (1). We have:

\[
\langle M \rangle \in \text{BTH} \Rightarrow M (\varepsilon) \text{ halts} \quad \text{(By def. of BTH)}
\]
\[
\Rightarrow M' (\langle N \rangle) \text{ accepts iff } N (\varepsilon) \text{ halts} \quad \text{(By def. of } M' \text{ above)}
\]
\[
\Rightarrow M' (\langle N \rangle) \text{ accepts iff } \langle N \rangle \in \text{BTH} \quad \text{(By def. of BTH)}
\]
\[
\Rightarrow L (M') = \text{BTH} \quad \text{(By def. of } L(M') \text{)}
\]
\[
\Rightarrow L (M') \text{ is undecidable} \quad \text{(Because BTH is undecidable)}
\]
\[
\Rightarrow \langle M' \rangle \in L_2 \quad \text{(By def. of } L_2 \text{)}.
\]

Now for (2) we have:

\[
\langle M \rangle \notin \text{BTH} \Rightarrow M (\varepsilon) \text{ loops} \quad \text{(By def. of BTH)}
\]
\[
\Rightarrow M' (\langle N \rangle) \text{ loops for all TMs } N \quad \text{(By def. of } M' \text{ above)}
\]
\[
\Rightarrow L (M') = \emptyset \quad \text{(By def. of } L(M') \text{)}
\]
\[
\Rightarrow L (M') \text{ is decidable} \quad \text{(Because } \emptyset \text{ is decidable)}
\]
\[
\Rightarrow \langle M' \rangle \notin L_2 \quad \text{(By def. of } L_2 \text{)}.
\]

This completes the proof. \[\square\]

The proof that $L_2$ is not co-r.e. follows from the following claim.
Claim 4.4 BTH $\leq_m L_2$.

Proof: As per Definition 3.3 we need to specify a computable function $f$ that takes as input the encoding $\langle M \rangle$ of a TM $M$ and outputs the encoding $\langle M' \rangle$ of a TM $M'$, such that the following is true:

(1) If $\langle M \rangle \in \text{BTH}$ then $\langle M' \rangle \in L_2$
(2) If $\langle M \rangle \notin \text{BTH}$ then $\langle M' \rangle \notin L_2$.

Here is the specification of the TM $M'$ associated to $M$. We write its input as $\langle N \rangle$ because we want to view this input as being a TM.

\[ M'(\langle N \rangle) \]
\begin{itemize}
  \item Run $M(\varepsilon)$ and $N(\varepsilon)$ in parallel
  \item If one of them halts then accept
\end{itemize}

We can observe that it is possible to write a program that given the code $\langle M \rangle$ of $M$ spits out the code $\langle M' \rangle$ of $M'$. This means the function $f$ defined by $f(\langle M \rangle) = \langle M' \rangle$ is computable.

Now we need to check the two properties (1),(2) above.

Begin with (1). We have:

\[ \langle M \rangle \in \text{BTH} \Rightarrow M(\varepsilon) \text{ halts } \quad \text{(By def. of BTH)} \]
\[ \Rightarrow M'(\langle N \rangle) \text{ accepts for all TMs } N \quad \text{(By def. of } M' \text{ above)} \]
\[ \Rightarrow L(M') = \Sigma^* \quad \text{(By def. of } L(M') \text{)} \]
\[ \Rightarrow L(M') \text{ is decidable } \quad \text{(Because } \Sigma^* \text{ is decidable)} \]
\[ \Rightarrow \langle M' \rangle \in L_2 \quad \text{(By def. of } L_2 \text{)} . \]

Now for (2) we have:

\[ \langle M \rangle \notin \text{BTH} \Rightarrow M(\varepsilon) \text{ loops } \quad \text{(By def. of BTH)} \]
\[ \Rightarrow M'(\langle N \rangle) \text{ accepts iff } N(\varepsilon) \text{ halts } \quad \text{(By def. of } M' \text{ above)} \]
\[ \Rightarrow M'(\langle N \rangle) \text{ accepts iff } \langle N \rangle \in \text{BTH} \quad \text{(By def. of BTH)} \]
\[ \Rightarrow L(M') = \text{BTH} \quad \text{(By def. of } L(M') \text{)} \]
\[ \Rightarrow L(M') \text{ is undecidable } \quad \text{(Because BTH is undecidable)} \]
\[ \Rightarrow \langle M' \rangle \notin L_2 \quad \text{(By def. of } L_2 \text{)} . \]

This completes the proof.