The alphabet of $M$ is $\{0, 1, B\}$, so each symbol of $M$'s tape can be held in one tape cell of $M_1$'s second tape. Observe that if we did not restrict the alphabet of $M_1$ we would have to use many cells of $M_1$'s tape to simulate one of $M$'s cells, but the simulation could be carried out with a little more work. The third tape holds the state of $M$, with $q_i$ represented by $0^i$. The behavior of $M_1$ is as follows:

1) Check the format of tape 1 to see if it has a prefix of the form $(8.2)$ and that there are no two codes that begin with $0^i 10^j$ for the same $i$ and $j$. Also check that if $0^i 10^j 10^k$ is a code, then $1 \leq i \leq 3$, $1 \leq j \leq 3$, and $1 \leq k \leq 2$. Tape 3 can be used as a scratch tape to facilitate the comparison of codes.

2) Initialize tape 2 to contain $w$, the portion of the input beyond the second block of three $1$'s. Initialize tape 3 to hold a single $0$, representing $q_1$. All three tape heads are positioned on the leftmost symbols. These symbols may be marked so the heads can find their way back.

3) If tape 3 holds $00$, the code for the final state, halt and accept.

4) Let $x_j$ be the symbol currently scanned by tape head 2 and let $0^i$ be the current contents of tape 3. Scan tape 1 from the left end to the second 111, looking for a substring beginning $10^i 10^j 10^k$. If no such string is found, halt and reject; $M$ has no next move and has not accepted. If such a code is found, let it be $0^i 10^j 10^k$. Then put $0^i$ on tape 3, print $x_j$ on the tape cell scanned by head 2 and move that head in direction $D_n$. Note that we have checked in (1) that $1 \leq i \leq 3$ and $1 \leq k \leq 2$. Go to step (3).

It is straightforward to check that $M_1$ accepts $(M, w)$ if and only if $M$ accepts $w$. It is also true that if $M$ runs forever on $w$, $M_1$ will run forever on $(M, w)$, and if $M$ halts on $w$ without accepting, $M_1$ does the same on $(M, w)$.

3.4 RICE'S THEOREM AND SOME MORE UNDECIDABLE PROBLEMS

We now have an example of an r.e. language that is not recursive. The associated problem "Does $M$ accept $w$?" is undecidable, and we can use this fact to show that other problems are undecidable. In this section we shall give several examples of undecidable problems concerning r.e. sets. In the next three sections we shall discuss some undecidable problems taken from outside the realm of TM's.

Example 8.2 Consider the problem: "Is $L(M) \neq \emptyset$?" Let $(M)$ denote a code for $M$ as in (8.2). Then define

$$L_{eq} = \{ (M) \mid L(M) \neq \emptyset \} \quad \text{and} \quad L_{ne} = \{ (M) \mid L(M) = \emptyset \}.$$  

Note that $L_{eq}$ and $L_{ne}$ are complements of one another, since every binary string denotes some TM; those with a bad format denote the TM with no moves. All these strings are in $L_e$. We claim that $L_{ne}$ is r.e. but not recursive and that $L_{eq}$ is not r.e.

We show that $L_{ne}$ is r.e. by constructing a TM $M$ to recognize codes of TMs that accept nonempty sets. Given input $(M)$, $M$ nondeterministically guesses a string $x$ accepted by $M_i$ and verifies that $M_i$ does indeed accept $x$ by simulating $M_i$ on input $x$. This step can also be carried out deterministically if we use the pair generator described in Section 7.7. For pair $(j, k)$ simulate $M_i$ on the $j$th binary string (in canonical order) for $k$ steps. If $M_i$ accepts, then $M$ accepts $(M_i)$.

Now we must show that $L_{eq}$ is not recursive. Suppose it were. Then we could construct an algorithm for $L_{eq}$, violating Theorem 8.5. Let $A$ be a hypothetical algorithm accepting $L_{eq}$. There is an algorithm $B$ that, given $(M, w)$, constructs a TM $M'$ that accepts $\emptyset$ if $M$ does not accept $w$ and accepts $(0 + 1)^*$ if $M$ accepts $w$. The plan of $M'$ is shown in Fig. 8.6. $M'$ ignores its input $x$ and instead simulates $M$ on input $w$, accepting if $M$ accepts.

Note that $M'$ is not $B$. Rather, $B$, is like a compiler that takes $(M, w)$ as "source program" and produces $M'$ as "object program." We have described what $B$ must do, but not how it does it. The construction of $B$ is simple. It takes $(M, w)$
and isolates $w$. Say $w = a_1 a_2 \cdots a_n$ is of length $n$. $B$ creates $n + 3$ states $q_1, q_2, \ldots, q_{n+3}$ with moves

$\delta(q_1, X) = (q_2, S, R)$ for any $X$ (print marker),

$\delta(q_i, X) = (q_{i+1}, a_{i-1}, R)$ for any $X$ and $2 \leq i \leq n + 1$ (print $w$),

$\delta(q_{n+2}, X) = (q_{n+2}, B, R)$ for $X \neq B$ (erase tape),

$\delta(q_{n+2}, B) = (q_{n+3}, B, L)$,

$\delta(q_{n+3}, X) = (q_{n+3}, X, L)$ for $X \neq S$ (find marker).

Having produced the code for these moves, $B$ then adds $n + 3$ to the indices of the states of $M$ and includes the move

$\delta(q_{n+3}, S) = (q_{n+4}, S, R)$ /* start up $M$ */

and all the moves of $M$ in its generated TM. The resulting TM has an extra tape symbol $S$, but by Theorem 7.10 we may construct $M'$ with tape alphabet $\{0, 1, B\}$, and we may surely make $q_2$ the accepting state. This step completes the algorithm $B$, and its output is the desired $M'$ of Fig. 8.6.

Now suppose algorithm $A$ accepting $L_a$ exists. Then we construct an algorithm $C$ for $L_a$ as in Fig. 8.7. If $M$ accepts $w$, then $L(M') \neq \emptyset$; so $A$ says "no" and $C$ says "yes." If $M$ does not accept $w$, then $L(M') = \emptyset$, $A$ says "yes," and $C$ says "no." As $C$ does not exist by Theorem 8.5, $A$ cannot exist. Thus, $L_a$ is not recursive. If $L_{ne}$ were recursive, $L_a$ would be also by Theorem 8.1. Thus $L_{ne}$ is r.e. but not recursive. If $L_{ne}$ were r.e., then $L_{ne}$ and $L_{ne}$ would be recursive by Theorem 8.3. Thus $L_{ne}$ is not r.e.

**Example 8.3** Consider the language

$L_e = \langle M \rangle \mid L(M)$ is recursive

and

$L_{ne} = \langle M \rangle \mid L(M)$ is not recursive.

Note that $L_e$ is not $\langle M \rangle \mid M$ halts on all inputs], although it includes the latter language. A TM $M$ could accept a recursive language even though $M$ itself might loop forever on some words not in $L(M)$; some other TM equivalent to $M$ must always halt, however. We claim neither $L_e$ nor $L_{ne}$ is r.e.

Suppose $L_e$ were r.e. Then we could construct a TM for $L_{ne}$ which we know does not exist. Let $M'$ be a TM accepting $L_e$. We may construct an algorithm $A$ that takes $\langle M, w \rangle$ as input and produces as output a TM $M'$ such that

$L(M') = \begin{cases} \emptyset & \text{if } M \text{ does not accept } w, \\ L_a & \text{if } M \text{ accepts } w. \end{cases}$

Note that $L_a$ is not recursive, so $M'$ accepts a recursive language if and only if $M$ does not accept $w$. The plan of $M'$ is shown in Fig. 8.8. As in the previous example, we have described the output of $A$. We leave the construction of $A$ to the reader.
accepting $L$. Suppose $\mathcal{S}$ were decidable. Then there exists an algorithm $M'$ accepting $L_{M'}$. We use $M_L$ and $M_{\mathcal{S}}$ to construct an algorithm for $L_w$ as follows. First construct an algorithm $A$ that takes $\langle M, w \rangle$ as input and produces $\langle M' \rangle$ as output, where $L(M')$ is in $\mathcal{S}$ if and only if $M$ accepts $w$ ($\langle M, w \rangle$ is in $L_w$).

The design of $M'$ is shown in Fig. 8.11. First $M'$ ignores its input and simulates $M$ on $w$. If $M$ does not accept $w$, then $M'$ does not accept $x$. If $M$ accepts $w$, then $M'$ simulates $M_L$ on $x$, accepting $x$ if and only if $M_L$ accepts $x$. Thus $M'$ either accepts $\mathcal{S}$ or $L$ depending on whether $M$ accepts $w$.

![Fig. 8.11 M' used in Rice's theorem.](image)

We may use the hypothetical $M_{\mathcal{S}}$ to determine if $L(M')$ is in $\mathcal{S}$. Since $L(M')$ is in $\mathcal{S}$ if and only if $\langle M, w \rangle$ is in $L_w$, we have an algorithm for recognizing $L_{M'}$, a contradiction. Thus $\mathcal{S}$ must be undecidable. Note how this proof generalizes Example 8.2.

Theorem 8.6 has a great variety of consequences, some of which are summarized in the following corollary.

**Corollary** The following properties of r.e. sets are not decidable:

1. emptiness,
2. finiteness,
3. regularity,
4. context-freedom.

### Rice's Theorem for recursively enumerable index sets

The condition under which a set $L_{\mathcal{S}}$ is r.e. is far more complicated. We shall show that $L_{\mathcal{S}}$ is r.e. if and only if $\mathcal{S}$ satisfies the following three conditions.

1. If $L$ is in $\mathcal{S}$ and $L \subseteq E$, for some r.e. $E$, then $L$ is in $\mathcal{S}$ (the containment property).
2. If $L$ is an infinite language in $\mathcal{S}$, then there is a finite subset of $L$ in $\mathcal{S}$.
3. The set of finite languages in $\mathcal{S}$ is enumerable, in the sense that there is a Turing machine that generates the (possibly) infinite string code$_1\#$code$_2\#\ldots$, where code$_i$ is a code for the $i$th finite language in $\mathcal{S}$ (in