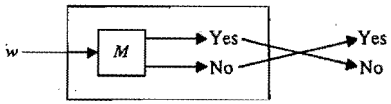


**Fig. 8.1** Construction showing that recursive languages are closed under complementation.

final state. Since one of these two events occurs, $M'$ is an algorithm. Clearly $L(M')$ is the complement of $L$ and thus the complement of $L$ is a recursive language. Figure 8.1 pictures the construction of $M'$ from $M$. □

**Theorem 8.2** The union of two recursive languages is recursive. The union of two recursively enumerable languages is recursively enumerable.

*Proof* Let $L_1$ and $L_2$ be recursive languages accepted by algorithms $M_1$ and $M_2$. We construct $M$, which first simulates $M_1$. If $M_1$ accepts, then $M$ accepts. If $M_1$ rejects, then $M$ simulates $M_2$ and accepts if and only if $M_2$ accepts. Since both $M_1$ and $M_2$ are algorithms, $M$ is guaranteed to halt. Clearly $M$ accepts $L_1 \cup L_2$.

For recursively enumerable languages the above construction does not work, since $M_1$ may not halt. Instead $M$ can simultaneously simulate $M_1$ and $M_2$ on separate tapes. If either accepts, then $M$ accepts. Figure 8.2 shows the two constructions of this theorem. □

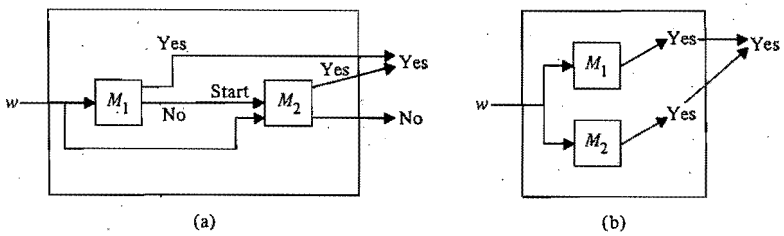


**Fig. 8.2** Construction for union.

**Theorem 8.3** If a language $L$ and its complement $\bar{L}$ are both recursively enumerable, then $L$ (and hence $\bar{L}$) is recursive.

*Proof* Let $M_1$ and $M_2$ accept $L$ and $\bar{L}$ respectively. Construct $M$ as in Fig. 8.3 to simulate simultaneously $M_1$ and $M_2$. $M$ accepts $w$ if $M_1$ accepts $w$ and rejects $w$ if $M_2$ accepts $w$. Since $w$ is in either $L$ or $\bar{L}$, we know that exactly one of $M_1$ or $M_2$ will accept. Thus $M$ will always say either "yes" or "no," but will never say both. Note that there is no *a priori* limit on how long it may take before $M_1$ or $M_2$ accepts, but it is certain that one or the other will do so. Since $M$ is an algorithm that accepts $L$, it follows that $L$ is recursive. □



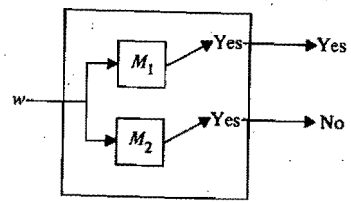


**Fig. 8.3** Construction for Theorem 8.3.

Theorems 8.1 and 8.3 have an important consequence. Let $L$ and $\bar{L}$ be a pair of complementary languages. Then either

1) both $L$ and $\bar{L}$ are recursive,
2) neither $L$ nor $\bar{L}$ is r.e., or
3) one of $L$ and $\bar{L}$ is r.e. but not recursive; the other is not r.e.

An important technique for showing a problem undecidable is to show by diagonalization that the complement of the language for that problem is not r.e. Thus case (2) or (3) above must apply. This technique is essential in proving our first problem undecidable. After that, various forms of reductions may be employed to show other problems undecidable.

## 8.3 UNIVERSAL TURING MACHINES AND AN UNDECIDABLE PROBLEM

We shall now use diagonalization to show a particular problem to be undecidable. The problem is: "Does Turing machine $M$ accept input $w$?" Here, both $M$ and $w$ are parameters of the problem. In formalizing the problem as a language we shall restrict $w$ to be over alphabet $\{0, 1\}$ and $M$ to have tape alphabet $\{0, 1, B\}$. As the restricted problem is undecidable, the more general problem is surely undecidable as well. We choose to work with the more restricted version to simplify the encoding of problem instances as strings.

### Turing machine codes

To begin, we encode Turing machines with restricted alphabets as strings over $\{0, 1\}$. Let

$$M = (Q, \{0, 1\}, \{0, 1, B\}, \delta, q_1, B, \{q_2\})$$

be a Turing machine with input alphabet $\{0, 1\}$ and the blank as the only additional tape symbol. We further assume that $Q = \{q_1, q_2, \ldots, q_n\}$ is the set of states, and that $q_2$ is the only final state. Theorem 7.10 assures us that if $L \subseteq (0 + 1)^*$ is accepted by any TM, then it is accepted by one with alphabet $\{0, 1, B\}$. Also, there

is no need for more than one final state in any TM, since once it accepts it may as well halt.

It is convenient to call symbols 0, 1, and $B$ by the synonyms $X_1$, $X_2$, $X_3$, respectively. We also give directions $L$ and $R$ the synonyms $D_1$ and $D_2$, respectively. Then a generic move $\delta(q_i, X_j) = (q_k, X_\ell, D_m)$ is encoded by the binary string

$$0^i10^j10^k10^\ell10^m. \tag{8.1}$$

A binary code for Turing machine $M$ is

$$111\ \text{code}_1\ 11\ \text{code}_2\ 11 \cdots 11\ \text{code}_r\ 111, \tag{8.2}$$

where each $\text{code}_i$ is a string of the form (8.1), and each move of $M$ is encoded by one of the $\text{code}_i$'s. The moves need not be in any particular order, so each TM actually has many codes. Any such code for $M$ will be denoted $\langle M \rangle$.

Every binary string can be interpreted as the code for at most one TM; many binary strings are not the code of any TM. To see that decoding is unique, note that no string of the form (8.1) has two 1's in a row, so the $\text{code}_i$'s can be found directly. If a string fails to begin and end with exactly three 1's, has three 1's other than at the end, or has two pair of 1's with other than five blocks of 0's in between, then the string represents no TM.

The pair $M$ and $w$ is represented by a string of the form (8.2) followed by $w$. Any such string will be denoted $\langle M, w \rangle$.

---

**Example 8.1** Let $M = (\{q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, B\}, \delta, q_1, B, \{q_2\})$ have moves:

$$\delta(q_1, 1) = (q_3, 0, R),$$
$$\delta(q_3, 0) = (q_1, 1, R),$$
$$\delta(q_3, 1) = (q_2, 0, R),$$
$$\delta(q_3, B) = (q_3, 1, L).$$

Thus one string denoted by $\langle M, 1011 \rangle$ is

$$11101001000101001100010101010010011$$
$$00010010010100110001000100010010111011$$

Note that many different strings are also codes for the pair $\langle M, 1011 \rangle$, and any of these may be referred to by the notation $\langle M, 1011 \rangle$.

---

### A non-r.e. language

Suppose we have a list of $(0 + 1)^*$ in canonical order (see Section 7.7), where $w_i$ is the $i$th word, and $M_j$ is the TM whose code, as in (8.2) is the integer $j$ written in



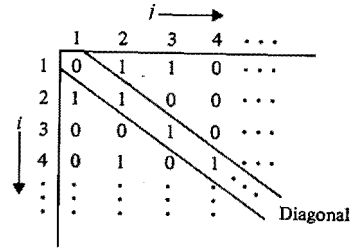


**Fig. 8.4** Hypothetical table indicating acceptance of words by TM's.

binary. Imagine an infinite table that tells for all $i$ and $j$ whether $w_i$ is in $L(M_j)$. Figure 8.4 suggests such a table;† 0 means $w_i$ is not in $L(M_j)$ and 1 means it is.

We construct a language $L_d$ by using the diagonal entries of the table to determine membership in $L_d$. To guarantee that no TM accepts $L_d$, we insist that $w_i$ is in $L_d$ if and only if the $(i, i)$ entry is 0, that is, if $M_i$ does not accept $w_i$. Suppose that some TM $M_j$ accepted $L_d$. Then we are faced with the following contradiction. If $w_j$ is in $L_d$, then the $(j, j)$ entry is 0, implying that $w_j$ is not in $L(M_j)$ and contradicting $L_d = L(M_j)$. On the other hand, if $w_j$ is not in $L_d$, then the $(j, j)$ entry is 1, implying that $w_j$ is in $L(M_j)$, which again contradicts $L_d = L(M_j)$. As $w_j$ is either in or not in $L_d$, we conclude that our assumption, $L_d = L(M_j)$, is false. Thus, no TM in the list accepts $L_d$, and by Theorem 7.10, no TM whatsoever accepts $L_d$.

We have thus proved

**Lemma 8.1** $L_d$ is not r.e.

### The universal language

Define $L_u$, the "universal language," to be $\{\langle M, w \rangle \mid M \text{ accepts } w\}$. We call $L_u$ "universal" since the question of whether any particular string $w$ in $(0 + 1)^*$ is accepted by any particular Turing machine $M$ is equivalent to the question of whether $\langle M', w \rangle$ is in $L_u$, where $M'$ is the TM with tape alphabet $\{0, 1, B\}$ equivalent to $M$ constructed as in Theorem 7.10.

**Theorem 8.4** $L_u$ is recursively enumerable.

*Proof* We shall exhibit a three-tape TM $M_1$ accepting $L_u$. The first tape of $M_1$ is the input tape, and the input head on that tape is used to look up moves of the TM $M$ when given code $\langle M, w \rangle$ as input. Note that the moves of $M$ are found between the first two blocks of three 1's. The second tape of $M_1$ will simulate the tape of $M$.

† Actually as all low-numbered Turing machines accept the empty set, the correct portion of the table shown has all 0's.

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 $M$, 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 that it has a prefix of the form (8.2) and that there are no two codes that begin with $0^i10^j1$ for the same $i$ and $j$. Also check that if $0^i10^j10^k10^\ell10^m$ is a code, then $1 \leq j \leq 3$, $1 \leq \ell \leq 3$, and $1 \leq m \leq 2$. 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 $110^i10^j1$. 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 $0^i10^j10^k10^\ell10^m$. Then put $0^k$ on tape 3, print $X_\ell$ on the tape cell scanned by head 2 and move that head in direction $D_m$. Note that we have checked in (1) that $1 \leq \ell \leq 3$ and $1 \leq m \leq 2$. Go to step (3).

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

The existence of $M_1$ is sufficient to prove Theorem 8.4. However, by Theorems 7.2 and 7.10, we can find a TM with one semi-infinite tape and alphabet $\{0, 1, B\}$ accepting $L_u$. We call this particular TM $M_u$, the *universal Turing machine*, since it does the work of any TM with input alphabet $\{0, 1\}$.

By Lemma 8.1, the diagonal language $L_d$ is not r.e., and hence not recursive. Thus by Theorem 8.1, $\bar{L}_d$ is not recursive. Note that $\bar{L}_d = \{w_i \mid M_i \text{ accepts } w_i\}$. We can prove the universal language $L_u = \{\langle M, w\rangle \mid M \text{ accepts } w\}$ not to be recursive by reducing $\bar{L}_d$ to $L_u$. Thus $L_u$ is an example of a language that is r.e. but not recursive. In fact, $\bar{L}_d$ is another example of such a language.

**Theorem 8.5** $L_u$ is not recursive.

*Proof* Suppose $A$ were an algorithm recognizing $L_u$. Then we could recognize $\bar{L}_d$ as follows. Given string $w$ in $(\mathbf{0} + \mathbf{1})^*$, determine by an easy calculation the value of $i$ such that $w = w_i$. Integer $i$ in binary is the code for some TM $M_i$. Feed $\langle M_i, w_i\rangle$ to algorithm $A$ and accept $w$ if and only if $M_i$ accepts $w_i$. The construction is shown in Fig. 8.5. It is easy to check that the constructed algorithm accepts $w$ if



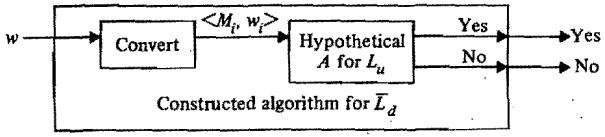


**Fig. 8.5** Reduction of $\bar{L}_d$ to $L_u$.

and only if $w = w_i$ and $w_i$ is in $L(M_i)$. Thus we have an algorithm for $\bar{L}_d$. Since no such algorithm exists, we know our assumption, that algorithm $A$ for $L_u$ exists, is false. Hence $L_u$ is r.e. but not recursive. □

## 8.4 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 $\langle M\rangle$ denote a code for $M$ as in (8.2). Then define

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

Note that $L_e$ 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 $L_e$ is not r.e.

We show that $L_{ne}$ is r.e. by constructing a TM $M$ to recognize codes of TM's that accept nonempty sets. Given input $\langle M_i\rangle$, $M$ 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 $\langle M_i\rangle$.

Now we must show that $L_e$ is not recursive. Suppose it were. Then we could construct an algorithm for $L_u$, violating Theorem 8.5. Let $A$ be a hypothetical algorithm accepting $L_e$. There is an algorithm $B$ that, given $\langle M, w\rangle$, constructs a TM $M'$ that accepts $\emptyset$ if $M$ does not accept $w$ and accepts $(\mathbf{0} + \mathbf{1})^*$ 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 $\langle M, w\rangle$ 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 $\langle M, w\rangle$