Chapter 9

Undecidability

(part b)

(2015/12/22)

Bibury, United Kingdom, 2000

Outline

9.0 Introduction (in part a)

9.1 A Language That Is Not Recursively Enumerable (in part a)

9.2 An Undecidable Problem That Is RE (in part a)

9.3 Undecidable Problems about TM's (in this part)

9.4 Post Correspondence Problem (in part c)

9.5 Other Undecidable Problems (in part c)

9.3 Undecidable Problems about Turing Machines

- **Concepts to be taught ---**
  - In this part, we will prove Rice's Theorem: any nontrivial property of TM's, which depends only on the language the TM accepts, is undecidable.
  - Also, we will investigate undecidable problems not involving TM's or their languages.

9.3.0 <u>Reviews</u>

- A review of proof of *undecidability* of the language $\overline{L}_u$ ---
  - $\overline{L}_u$ is the complement of the universal language L_u
  - The proof is based on a *reduction* from L_d to $\overline{L}_u$ as shown in Fig. 9.5 in Section 9.2.3, which is repeated here.

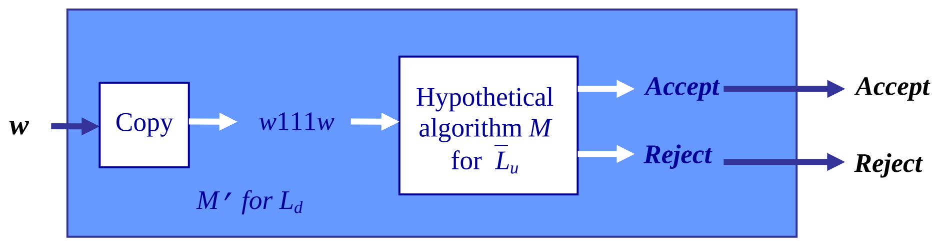

Fig. 9.5 A TM M' to accept L_d (repeated).

- A review of the technique of *problem reduction* ---
  - The basic idea of this technique is illustrated in Fig. 8.4 in Section 8.1.3, which is repeated here.

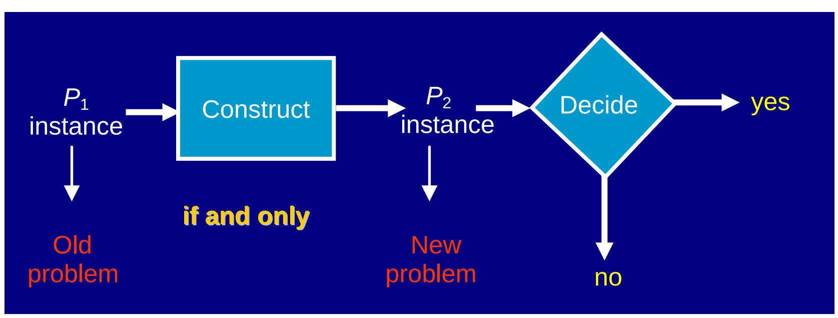

Fig. 8.4 An illustration of reducing one problem to another (repeated).

9.3.1 <u>Reductions</u>

- **Definition ---**

  If we have an *algorithm* to convert instances of a problem P_1 to instances of P_2 that have the same answer, then we say that P_1 *reduces* to P_2.

  - This technique of reduction may be used to prove that P_2 is at least as hard as P_1, like the following cases (proved later):

- if P_1 is not recursive, then P_2 cannot be recursive;
- if P_1 is non-RE, then P_2 cannot be RE.

■ The reduction technique may be visualized as Fig. 9.6.

♦ It is allowed that only a small fraction of P_2 is a target of the reduction (see the smaller circles in the right larger circle in Fig. 9.6).

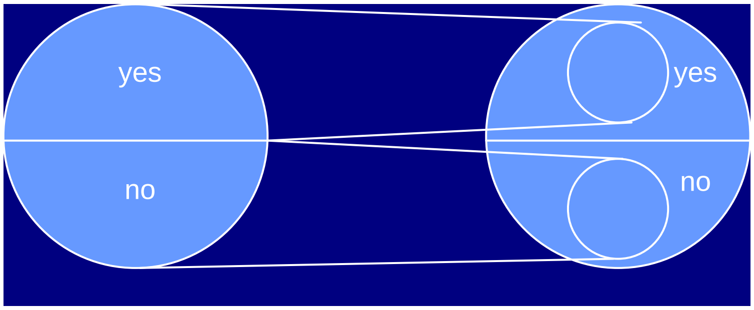

Fig. 9.6 Visualization of problem reduction (Fig. 9.7 in the textbook).

■ Alternative concepts of problem reduction ---

A reduction may be regarded as:

♦ a *TM* that takes an *instance* of P_1 on its tape and *halts* with an instance of P_2 on its tape; or

♦ a *computer program* that takes an instance of P_1 as input and produces an instance of P_2 as output; or

♦ an *algorithm* that takes an instance of P_1 as input and produces an instance of P_2 as output.

■ Theorem 9.7 ---

If there is a reduction from P_1 to P_2, then

(a) if P_1 is undecidable (not recursive), then so is P_2;
(b) if P_1 is non-RE, then so is P_2.

***Proof*.** Proof by contradiction.

■ *Proof of Part (a)* ---

♦ Suppose P_1 is undecidable.

♦ If P_2 is decidable, say, by the use of an algorithm A_2, then we combine the reduction R with A_2 to form an algorithm A_1 to decide P_1 in the following way (see Fig. 9.7).

(1) Given an *instance* w of P_1, apply the reduction algorithm R to w to get an instance of P_2, say x.

(2) Since P_2 is decidable, we apply A_2 to x.

(a) If A_2 says "yes" (i.e., x is in P_2), then the answer to w for P_1 is also "yes" (i.e., w is in P_1) because x is derived from w by R which is an algorithm.

(b) Similarly, if x is not in P_2, then w is not in P_1.

♦ In the above way, we have an algorithm A_1 to decide P_1 (A_1 is illustrated in Fig. 9.7)
♦ But this is impossible because we know that P_1 is undecidable. Contradiction!
♦ So, the assumption that P_2 is decidable is not correct. Done.

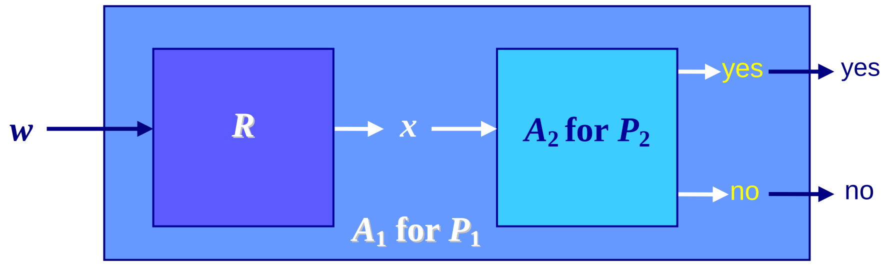

Fig. 9.7 An algorithm A_1 to implement P_1.

■ *Proof of Part (b)* ---
♦ Assume P_1 is non-RE, but P_2 is RE.
♦ So there exists a TM M_2 which will halt and says "yes" if its input is in P_2 (a language); and "no" if not.
♦ Now we combine the reduction algorithm R with M_2 to *construct* another TM M_1 in the following way (see Fig. 9.8).
(1) Given a *string* w in P_1, apply R to transform w to be a string x in P_2.
(2) If x is accepted by M_2, then let M_1 accept w (there is no need to check the case of "not accept")
♦ Now, if w is in P_1, then the corresponding x is in P_2 and accepted by M_2, and so w is accepted by M_1.
♦ That is, M_1 is the TM for P_1, or equivalently, P_1 is RE. Contradiction!
♦ So P_2 cannot be RE.

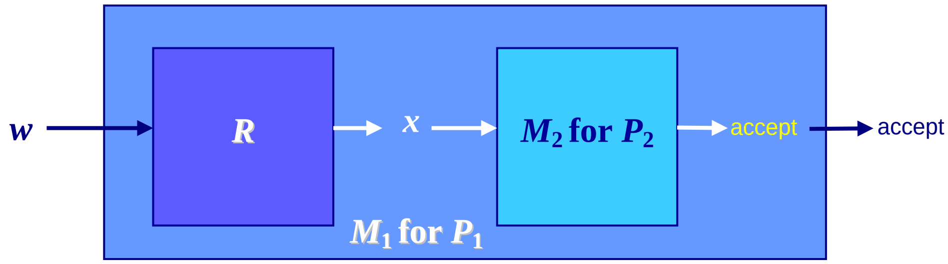

Fig. 9.8 A TM M_1 to implement P_1.

■ **A comment ---**

The diagram of Fig. 9.7 may be re-drawn as Fig. 9.9 for the case of proving Theorem 9.6 from the viewpoint of Fig. 8.4 to make it clear that Fig. 9.7 is indeed an example of problem reduction.

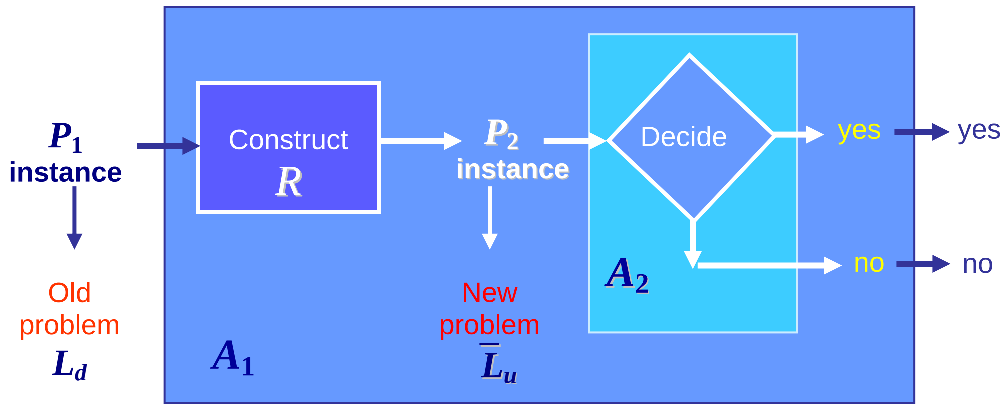

Fig. 9.9 An implementation of the algorithm of Fig. 9.7 from the viewpoint of problem reduction.

- **Another comment ---**

  Theorem 9.7 may be illustrated by Fig. 9.10 where each arrow $\rightarrow$ in the figure means "implies." By logic reasoning, in addition to the truth stated in the theorem, it is easy to see the validity of the reverse statement that if P_1 can be reduced to P_2 which is decidable, then P_1 is also decidable.

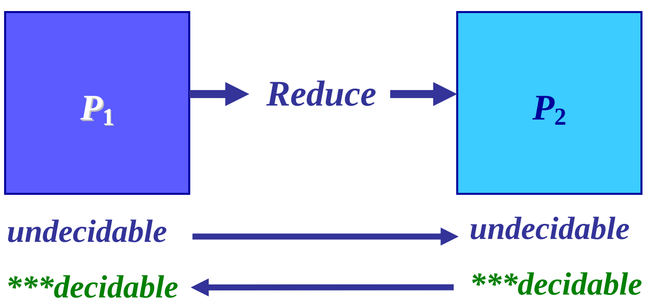

Fig. 9.10 An implementation of then.

9.3.2 <u>Turing Machines That Accepts the Empty Language</u>

- **Definitions ---**

  Regard the label M of a TM as its binary code. Define:

$$L_e = \{M \mid L(M) = \phi\}$$

which is the language of the codes of TM's which do not accept strings; and

$$L_{ne} = \{M \mid L(M) \neq \phi\}$$

which is the language of the codes of TM's, each accepting at least one string.

- **A review of results obtained so far ---**
  - ♦ L_d is not an RE language (Theorem 9.2).
  - ♦ It can be shown that $\overline{L}_d$ is RE (omitted; can be proved in the same way as we

show the universal language L_u to be RE).

- ♦ L_u is RE but not recursive (Theorem 9.6).
- ♦ $\overline{L}_u$ is not RE (by Theorem 9.4). (Reason: if $\overline{L}_u$ is RE, then by Theorem 9.4, L_u should be recursive; but this is not true according to Theorem 9.6.)
- ♦ These results are marked as yellow in Fig. 11.
- ♦ Now, we want to prove by reduction that L_e is non-RE, and L_{ne} is RE but not recursive (as Theorems 9.8, 9.9, 9.10).

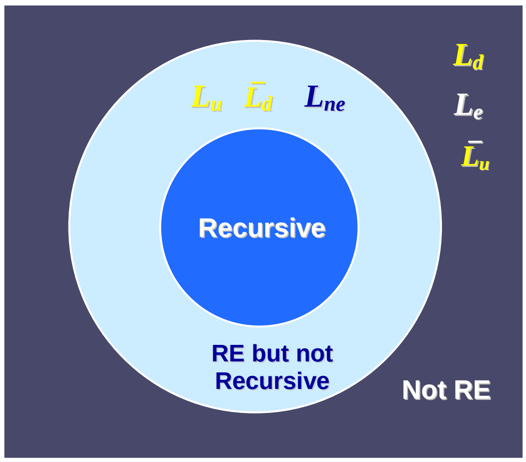

Fig. 9.11 Relationships among three classes of languages (Fig. 9.2 repeated).

■ **Theorem 9.8 ---**

L_{ne} is RE, but not recursive.

Proof.

■ *Proof of Part 1: proving "L_{ne} is RE"* ---

- ♦ We construct a nondeterministic TM M as shown in Fig. 12 (Fig. 9.8 in the textbook) which is described in detail as follows.
  - M takes as input a TM code, M_i.
  - Using *its nondeterministic capability*, M *guesses* an input w that M_i might accept.
  - M tests whether M_i accepts w by simulating the universal machine U that accepts L_u.
  - If M_i accepts w, then M accepts its own input M_i, too.

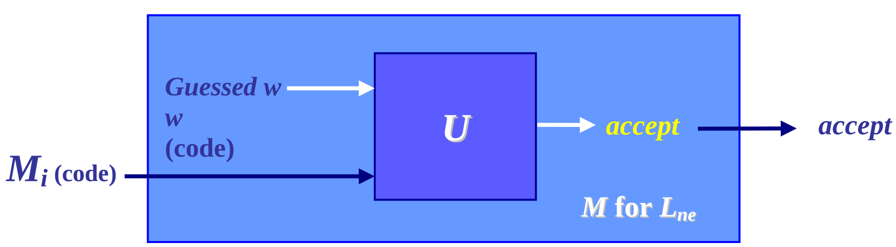

Fig. 9.12 A TM which accepts L_{ne} (Fig. 9.8 in the textbook).

- ♦ Obviously, with the code M_i as input, if the corresponding TM M_i accepts *even* one string, M will guess eventually that string (among others) and accepts M_i (using U).

- Conversely, if M_i does not accept any string (i.e., $L(M_i) = \phi$), then no guess of w will lead to acceptance by M_i and so M does not accept M_i (because of the property of U).
 (Note: U is the universal TM which accepts the universal language L_u, and each string in L_u is a pair (M_i, w) where M_i is a TM with the binary alphabet, and w is a binary string such that w is accepted by M_i.)
- The overall function of M with (the code of) M_i as input is: if M_i accepts w so that $L(M_i) \neq \phi$, then M accepts M_i.
- This means M is a TM for accepting the codes of TM's M_i with $L(M_i) \neq \phi$. That is, M is the TM for accepting L_{ne}. So, L_{ne} is RE.

■ *Proof of Part 2: proving "L_{ne} is not recursive"* ---

- Next, we want to prove L_{ne} is not recursive by reducing L_u to L_{ne}.
- We know
  - $L_u = \{(M, w) \mid w \in \{0, 1\}^*$ and $w \in L(M)\}$;
  - $L_{ne} = \{M \mid L(M) \neq \phi\}$.
- Reducing L_u to L_{ne} means transforming a code $y = (M, w) \in L_u$ to a code $z = M' \in L_{ne}$ such that $(M, w) \in L_u$ *if and only if* $M' \in L_{ne}$, which means: M accepts w *if and only if* $L(M') \neq \phi$ (i.e., M' accepts at least one string.)
- We want to prove this by reducing L_u to L_{ne} discussed previously. For this, we prove the reducibility at first: construct M' as shown in Fig. 9.13, which operates in the following way.
  (1) M' ignores its own input x and simulates M on input w (for details of this simulation using a universal TM U, see p. 396 in the textbook).
  (2) If M accepts w, then M' accepts any input x so that $L(M') \neq \phi$.
  (3) If M does not accept w, then M' will not accept any input x.
- Therefore, M accepts w if and only if $L(M') \neq \phi$ which means $(M, w) \in L_u$ *if and only if* $M' \in L_{ne}$.
- Accordingly, we can design a *reduction algorithm R* using a TM to transform the code (M, w) for M and the string w into the code for M' (the details not shown in the textbook, but the simulation mentioned above can convince you that you can do so).
- Therefore, by Theorem 9.7, since L_u is undecidable (i.e., not recursive), we conclude L_{ne} is not recursive, either. Done.

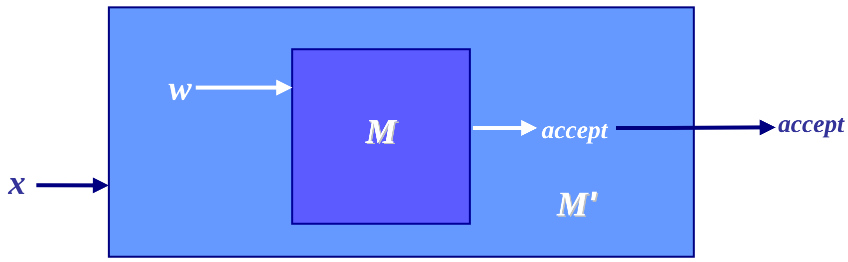

Fig. 9.13 A TM which accepts L_{ne} (Fig. 9.8 in the textbook).

■ *Proof of Part 3: another way to prove "L_{ne} is not recursive"* ---

- As a preliminary of proving Rice's Theorem later, we prove (by contradiction) below alternatively without using Theorem 9.7.
- Assume that L_{ne} is recursive. Then, there exists an algorithm B to decide if a given

input code M' can be accepted or not: if $L(M') \neq \phi$, then B accepts and halts; otherwise, "rejects" (does not accept but halts as well).

- Now, we develop an algorithm C using algorithms R mentioned above and B as illustrated in Fig. 9.14 which operates in the following way. (Note: R is a *reduction algorithm* using a TM to transform the code (M, w) for M and the string w into the code for M'.)
  (1) Algorithm C at first uses R to convert its input code (M, w) into code M' where M accepts w if and only if $L(M') \neq \phi$.
  (2) Then, B accepts the code M' if $L(M') \neq \phi$ and rejects it, otherwise.
  (3) Finally, we let C accepts if B accepts and vise versa.
  (4) The overall function of C is: with the code (M, w) as input, if M accepts w, then C accepts, and if not, then C rejects.
- That is, C is an algorithm for L_u, meaning that L_u is recursive. Contradiction.
- So, the assumption that L_{ne} is recursive is not true. Done.

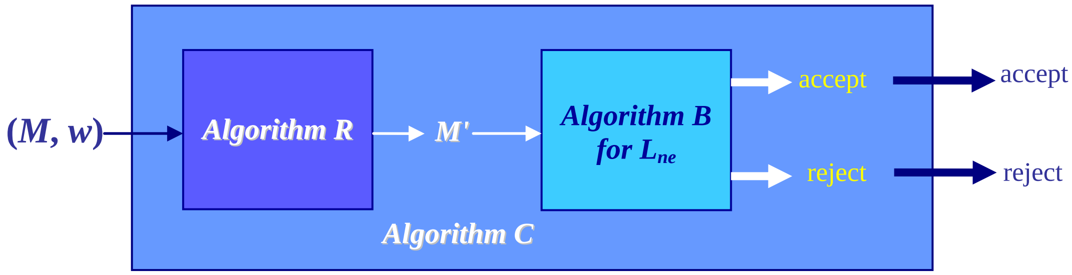

Fig. 9.14 Construction of an algorithm for L_u.

- **Theorem 9.10 ---**

  L_e is not RE.

Proof.

- Assume that L_e is RE. (Note that $L_e = \overline{L_{ne}}$).
- Then, since $L_{ne} = \overline{L_e}$ is RE, by Theorem 9.4 L_{ne} should be recursive. Contradiction!
- So L_e is not RE.

9.3.3 Rice's Theorem & Properties of RE Languages

- **Concepts ---**
  - We will prove that all nontrivial "properties" of the RE languages are undecidable.
  - A property of the RE languages is "the language is context-free."

- **Definition ---**

  A *property* of the RE languages is a *set* of RE languages.

  - So the property of *being context-free* is the set of CFL's.
  - A property is *trivial* if it is either *empty* or is the set of *all* RE languages; otherwise, *nontrivial*.
  - Examples --- the *empty property*, ϕ, is different from the property of *being an empty language*, $\{\phi\}$.

- **More concepts ---**
  - We cannot recognize a set of languages with languages themselves as the input to the recognizer (a TM usually) because a language, usually being infinite, cannot be written down as a finite-length string.
  - Instead, we recognize the codes of the TM's which accept these languages because the TM code itself is finite in length.
  - So, for a property $\mathcal{P}$ of the RE languages, we use $L_{\mathcal{P}}$ to denote the set of codes for the TM's M_i such that $L(M_i)$ is a language in $\mathcal{P}$.
  - When talking about the decidability of a property $\mathcal{P}$, we mean the decidability of the language $L_{\mathcal{P}}$.

- **Theorem 9.11 (Rice's Theorem) ---**

  Every nontrivial property $\mathcal{P}$ of the RE languages is undecidable.

Proof.

- *Case I: the empty language ϕ is not in $\mathcal{P}$---*
  - Assume at first that the empty language ϕ is *not* in $\mathcal{P}$.
  - Since $\mathcal{P}$ is nontrivial, there must be some nonempty language L in $\mathcal{P}$.
  - Let M_L be the TM accepting L.
  - We will reduce L_u to $L_{\mathcal{P}}$, thus proving that $L_{\mathcal{P}}$ is undecidable (according to Theorem 9.7).
  - Reduction of L_u to $L_{\mathcal{P}}$ means transforming a code $y = (M, w) \in L_u$ into a TM code $M' \in L_{\mathcal{P}}$ with the language of M' being $L \in \mathcal{P}$ as mentioned previously, such that $(M, w) \in L_u$ if and only if M' accepts L.
  - The algorithm A for this reduction may be designed to take a pair (M, w) and produce a TM M'.
  - The design of M' is shown in Fig. 9.15, which has the function:

    $L(M')$ is ϕ if M does not accept w, and $L(M') = L$ if M accepts w.

    (M' as shown in Fig. 9.15 is quite similar to M' shown in Fig. 9.13 except that an additional TM M_L is included).
  - This function is achieved in the following way.

    (1) M' simulates M on input w (for details of this simulation using a universal TM U, see p. 398 in the textbook).

    (2) If M accepts w, then M' begins simulating M_L on its own input x to accept the language L. (Since L is in $\mathcal{P}$, the code for M' is in $L_{\mathcal{P}}$.)

    (3) If M does not accept w, then M' does nothing, and never accepts its own input x, so $L(M') = \phi$. (Since we assume ϕ is not in property $\mathcal{P}$, that means the code for M' is not in $L_{\mathcal{P}}$.)
  - The overall function of M' is: M accepts w if and only if M' accepts L.
  - It is observed that the reduction of constructing M' from M and w can be carried out by an algorithm (named previously as A) (as said in the textbook).
  - By the "theorem of reduction" (Theorem 9.7), since L_u is undecidable, we conclude that $L_{\mathcal{P}}$ is undecidable, or equivalently, $\mathcal{P}$ is undecidable.

- *Case II: the empty language ϕ is in $\mathcal{P}$---*

- Now, we have to deal with the other case where ϕ is in $\mathcal{P}$.
- If so, we consider $\overline{\mathcal{P}}$ which does not contain ϕ.
- A similar proof to that above may be applied to show that $\overline{\mathcal{P}}$ (or equivalently $L_{\overline{\mathcal{P}}}$) is undecidable.
- Since every TM accepts an RE language, $\overline{L}_{\mathcal{P}}$, the set of (the codes for) TM's that do not accept a language in $\mathcal{P}$ is *the same as* $L_{\overline{\mathcal{P}}}$, the set of (the codes for) TM's that accept a language in $\overline{\mathcal{P}}$.
- That is, $L_{\overline{\mathcal{P}}} = \overline{L}_{\mathcal{P}}$ so that $\overline{L}_{\mathcal{P}}$ is undecidable.
- Now, suppose $L_{\mathcal{P}}$ is decidable in this case (i.e., $\phi \in \mathcal{P}$).
- Then, by Theorem 9.3, $\overline{L}_{\mathcal{P}}$ is decidable, too. Contradictory to the just-proved fact that $\overline{L}_{\mathcal{P}}$ is undecidable.

■ Therefore, for either case, $L_{\mathcal{P}}$ is undecidable. Done.

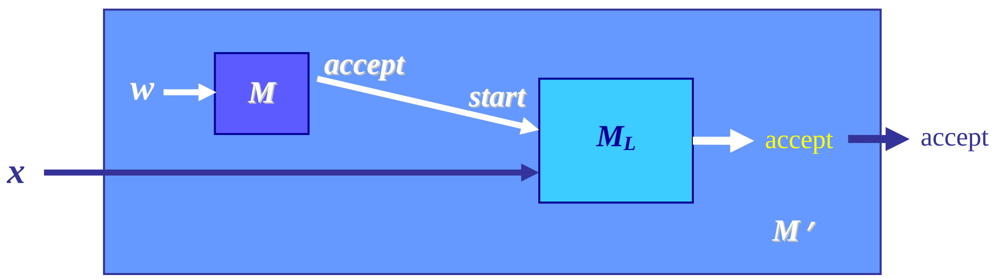

Figure 9.15: Construction of M' for proving Rice's Theorem (Fig. 9.10 in the textbook).

9.3.4 Problems about TM Specifications

■ All problems about TM's that involve only the language that the TM accepts are undecidable according to Rice's theorem.

■ The following are undecidable accordingly (except the first which is based on other theorems):
- whether the language accepted by a TM is empty (from Theorems 9.9 and 9.3);
- whether the language accepted by a TM is finite;
- whether the language accepted by a TM is a regular language;
- whether the language accepted by a TM is a context-free language.

■ Problems about TM's other than their languages are not related to Rice's Theorem.

■ **Example 9.12 ---**

This example shows some problems which can be decided.

- It is decidable whether a TM has five states.
- It is also decidable whether there exists some input such that the TM makes at least five moves.
- See the textbook for the details of the proofs.