Chapter 9

Undecidability

(part a)

(2015/12/17)

Island Castle, Lithuania 2002

Outline

9.0 Introduction

9.1 A Language That Is Not Recursively Enumerable

9.2 An Undecidable Problem That Is RE

9.3 Undecidable Problems about TM's

9.4 Post Correspondence Problem

9.5 Other Undecidable Problems

9.0 Introduction

- **Concepts to be taught ---**
  - We will prove the undecidability of several problems formally:
    - Does a TM accept (the code for) itself as input?
    - Does a TM accept a certain input?
  - Actually all nontrivial problems about the language accepted by a TM are undecidable.

9.1 A Language That Is Not RE

- **Review of definition ---**
  - An algorithm is a procedure which always halts.
  - A language L is *recursive enumerable* (RE) if $L = L(M)$ for some TM M.
  - A language L is *recursive* if $L = L(M)$ for some TM M which always halts, regardless of whether or not it accepts (i.e., halts both when accepting and when rejecting).

- **Goal of this study ---**
  - Want to prove *undecidable* the language of pairs (M, w) where
    - M is a TM (suitably coded in binary) with alphabet {0, 1};
    - w is a string of 0's and 1's;
    - M accepts input w.

  - If this problem is undecidable, then those with general alphabets re also undecidable.

- **Steps to achieve the above goal in the 1st stage (the 2nd stage is in the next section) ---**
  - Coding the TM into a binary string.
  - Treating <u>any</u> binary string as a TM.
  - Regarding a *non-well-formed* string as a TM with no move.
  - Setting up a language L_d (called *diagonalization language*) consisting of all strings w such that the TM represented by w does *not* accept the input w.

- **Language L_d (called diagonalization language) ---**
  - $L_d = \{w | w$ not accepted by the TM represented by $w\}$
  - $w \in L_d$ means the illustration of Fig. 9.1.

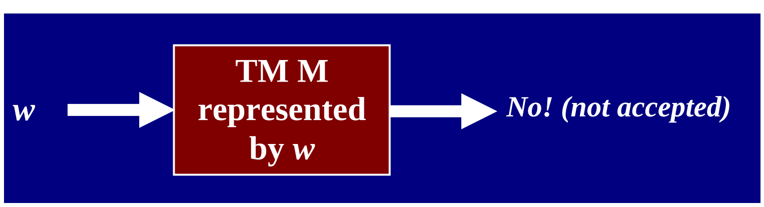

Fig. 8.1 A

- **Properties of the diagonalization language L_d ---**
  - *It can be proved that there is no TM which can accept L_d (later in this chapter) !!!*
  - Showing that no TM can accept a language is *stronger* than showing that it is undecidable (i.e., it has no algorithm or has <u>no TM that always halts</u>).
  - L_d plays the same role as the program H_2 in Section 8.1.2 of the last chapter (with a

self-contradiction property).

9.1.1 Enumerating the Binary Numbers

- We want to assign integers to binary strings in the following way:

  if w is a binary string, then regard $1w$ as *an* integer i, and call w the ith string, denoted as w_i.

  - For example, $(1w)_2 = i_{10} \Rightarrow w = i\text{th string} = w_i$.

9.1.2 Codes for the TM's

- We want to define the ith TM M_i after encoding the 7-tuple of the TM and assigning integers to the states, tape symbols, and directions L and R in the following way:
  - States are numbered as $q_1, q_2, \ldots, q_r$, with q_1 as the start state, and q_2 the only accepting state.
  - Tape symbols are numbered as $X_1, X_2, \ldots, X_s$, with $X_1 = 0$, $X_2 = 1$, $X_3 = B$.
  - L as D_1, and R as D_2.
  - Each transition rule $\delta(q_i, X_j) = (q_k, X_l, D_m)$ is represented by integers i, j, k, l, and m, and is coded as $C = 0^i10^j10^k10^l10^m$ with 1's as separators (a *unary number* representation).

  (Note that i, j, k, l, and m are all at least 1, so there will be no occurrence of two or more consecutive 1's in the code.)

- A complete code for a TM M consists of all the codes for the transitions, in a certain order, separated by pairs of 1's:

  $$\textbf{code of } \boldsymbol{M} = C_1 11 C_2 11 \ldots C_{n-1} 11 C_n$$

  where each Ci is the code for a transition.

- Code for (M, w) is that of M, followed by 111 and then w, i.e.,

  $$\textbf{code of } (\boldsymbol{M}, \boldsymbol{w}) = C_1 11 C_2 11 \ldots C_{n-1} 11 C_n 111 w$$

- **Example 9.1 ---**
- Given a TM $M = (\{q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, B\}, \delta, q_1, B, \{q_2\})$ where δ is such that

  $$\delta(q1, 1) = (q_3, 0, R),\ \delta(q_3, 0) = (q_1, 1, R),\ \delta(q_3, 1) = (q_2, 0, R),\ \delta(q_3, B) = (q_3, 1, L),$$

  then
  - the codes for the transition rules are

    $$0^1 10^2 10^3 10^1 10^2,\ 0^3 10^1 10^1 10^2 10^2,\ \ldots$$

  - and the code for M is

    $$0^1 10^2 10^3 10^1 10^2 \underline{11} 0^3 10^1 10^1 10^2 10^2 \underline{11} \ldots$$

9.1.3 The Diagonalization Language

- The previous coding of TM's allows a concrete notion of M_i, the ith TM, which is TM M whose code is w_i, the ith binary string.

- <u>If w_i is not a valid code of a TM</u>, we take M_i to be the TM with one state and *no* transition. That is, a TM with an invalid code w_i will halt immediately on any input.

- Therefore, we have a list of TM's in a certain order and there is a 1-to-1 mapping between TM's M_i and w_i.

- **Definition ---**

  The diagonalization language L_d is the set of strings w_i such that w_i is *not* in $L(Mi)$.

  - That is, L_d consists of all strings w such that the TM M, whose code is w, does not accept w when w is given as input.
  - By notations, we have $L_d = \{w_i \mid w_i \notin L(M_i)\}$.

- Why L_d is called a "diagonalization language"? ---
  - See Fig. 9.1 at first.

  - Meaning of the diagonal values:
    - "1" means that M_i accepts w_i;
    - "0" means that M_i does not accept w_i.

  - The *i*th row is the *characteristic vector* for language $L(M_i)$:
    - A "1" in the row indicates that the corresponding string is in the language.

		w_i						
		1	**2**	**3**	**4**	.	.	.
	1	**0**	**1**	**1**	**0**	.	.	.
	2	**1**	**1**	**0**	**0**	.	.	.
M_i	**3**	**0**	**0**	**1**	**1**	.	.	.
	4	**0**	**1**	**0**	**1**	.	.	.
	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	
	.	.	.	.	.	.	.	.

Fig. 9.1 Illustration of the diagonalization language.

  - To construct L_d, just complement the diagonal to collect the corresponding strings; all the resulting 1's in the diagonal means that the corresponding M_i does not accept w_i.
  - This is called the technique of *diagonalization*, useful for proving that "*Ld* is not RE."
  - Because the complement of the diagonal is 1000… in the previous figure, so L_d contains $w_1 = \varepsilon$ (from 1ε), but does not contain w_2, w_3, w_4, …

  - The *complemented* diagonal *disagrees in some column with every row* of the table of Fig. 9.1; <u>therefore it (any row) cannot be the characteristic vector of any TM!</u>
    (對每一列 row，至少在某一行 column 不會與原來的列一致！)

- ♦ A *non-zero* element at location (i, i) in the complemented diagonal means that M_i does not accept the corresponding w_i. (statement A)

9.1.4 Proof that L_d is not RE

- ■ **Theorem 9.2** ("Tragedy Theorem 悲劇定理") ---

  L_d is not an RE language. That is, no TM accepts L_d.

  Proof. Prove by contradiction.
  - ♦ Let L_d be $L(M)$ for some TM M. Then M is one of the TM's in the TM list mentioned previously, say M_i.
  - ♦ Now, check if the corresponding w_i is in L_d.
    - If $w_i \in L_d$, then M_i accepts w_i (because $L(M_i) = L_d$). But this is impossible because by definition of L_d, w_i cannot be accepted by M_i. Contradiction.
    - If $w_i \notin L_d$, then w_i is not accepted by M_i because $L_d = L(M_i)$. This in turns means $w_i \in L_d$ by the definition of L_d (see statement (A) above). Contradiction again.
  - ♦ Neither case holds, so the assumption that $L_d = L(M)$ for some TM M must be false. That is, no TM accepts L_d. Done.

9.2 An Undecidable Problem That Is RE

- ■ Concepts to be taught:
  - ♦ RE languages are accepted (recognized) by TM's.
  - ♦ RE languages may be grouped into two classes:
    - Class 1 (*recursive language*) --- each language L in this class has a TM (thought as an algorithm) which not only accepts strings of L, but also tells us what strings are not in L *by halting*.
    - Class 2 (*RE but not recursive*) --- each language L in this class has a TM (*not* thought as an algorithm) which accepts strings of L, but may not halt when a given input string is not in L.

9.2.1 Recursive Languages

- ■ **Definition ---**

  A language L is *recursive* if $L = L(M)$ for some TM M such that:

  (1) if $w \in L$, then M accepts (and therefore halts);
  (2) if $w \notin L$, then M *eventually halts*, although it *never* enters an accepting state (i.e., it "rejects").

  - ♦ A TM of this type corresponds to the formal notion of *algorithm*.

- ■ **Definition ---**

A given language L, *regarded as a problem*, is called *decidable* if L is a recursive language; and *undecidable* if not.

- ♦ The existence or nonexistence of an algorithm to solve a problem (i.e., the problem is decidable or undecidable) is *more important* than the existence or nonexistence of a TM to solve the problem.
- ♦ Conceptually, we have following equivalent statements:

decidable problem $\Leftrightarrow$ recursive language L $\Leftrightarrow$ there is a TM M which halts with L as input.

■ **Relationships among three classes of languages ---**

- ♦ See Fig. 9.2 for the relationships among the following three classes of languages:
  - • Recursive language
  - • Recursive enumerable language (RE language)
  - • Non-RE language

- ♦ In Fig. 9.2, only L_d is studied so far; others will be investigated in this chapter.
- ♦ All the relationships will be proved in this chapter, too.

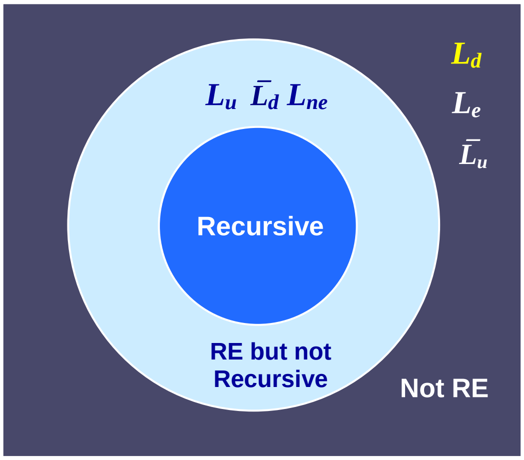

Fig. 9.2 Relationships among three classes of languages.

- ♦ L_u, the *universal language*, will soon be defined and proved *not* to be recursive, though it is an RE language.

9.2.2 Complements of Recursive and RE Languages

■ A powerful tool of proving is *language complementation*.

■ We will show that the recursive language is *closed* under complementation.

■ **Theorem 9.3 ---**

If a language L is recursive, so is $\bar{L}$.

***Proof*.**

- ♦ Prove by constructing a halting TM to accept $\overline{L}$.
- ♦ Let TM M accepts L, which always halts.
- ♦ Construct a new TM $\overline{M}$ for M as illustrated by Fig. 9.3.

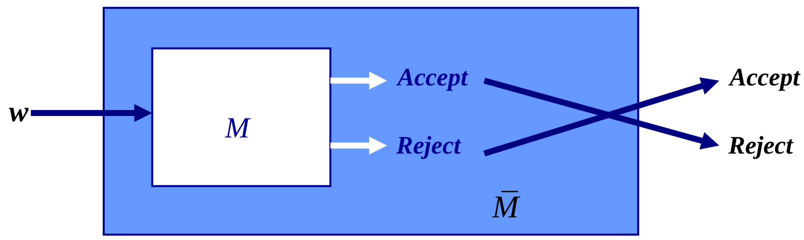

Fig. 9.3 A TM $\overline{M}$ to accept $\overline{L}$.

- The accepting states of M are made non-accepting with no transitions, i.e., in these states M will halt without accepting.
- $\overline{M}$ has a new accepting state r and no transition from r.
- For each combination of a non-accepting state of M and a tape symbol of M such that M has no transition (i.e., such that M halts without accepting), add a transition to the accepting state r.

- ♦ Then if M halts, so is $\overline{M}$, and $\overline{M}$ accepts strings not accepted by M.

■ **Theorem 9.4 ---**

If a language L and its complement are RE, then L is recursive, and so is $\overline{L}$.

***Proof*.**

- ♦ Easy by Fig. 9.4 where $L = L(M_1)$ and $\overline{L} = L(M_2)$ and the new machine is M.

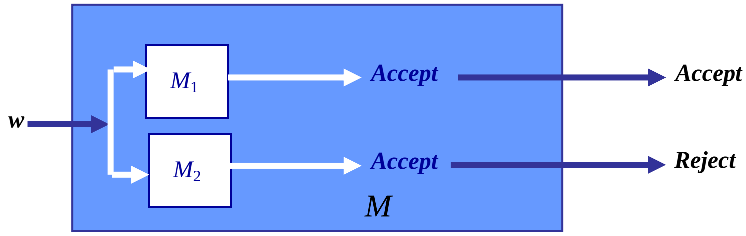

Fig. 9.4 A TM M to accept $\overline{L}$.

- ♦ If w is in L, then M_1 will eventually accept. If so, M will accept and halt.
- ♦ If w is not in L, then it is in $\overline{L}$, so that M_2 will eventually accept. If so, M will halt without accepting.
- ♦ Therefore, for all input w, M halts and $L(M) = L$, that is, L is recursive.
- ♦ By Theorem 9.3, $\overline{L}$ is recursive. Done.

- **Discussions ---**
  - Of the 9 possibilities of placing L and $\overline{L}$ in Fig. 9.2, only the following 4 are valid by Theorems 9.3 and 9.4:
    - Both L and $\overline{L}$ are recursive (both in the inner ring of Fig. 9.2).
    - Neither L nor $\overline{L}$ is RE. (both in the outer ring).
    - L is RE but not recursive, and $\overline{L}$ is not in RE (one in the middle ring, and the other in the outer ring), like L_u and $\overline{L}_u$ in Fig. 9.2.
    - $\overline{L}$ is RE but not recursive, and L is not in RE (a swap of above), like $\overline{L}_d$ and L_d, and L_{ne} and L_e in Fig. 9.2.

- **Example 9.5 ---**
  - L_d is not RE as shown before.
  - So, $\overline{L}_d$ cannot be recursive (not in the four cases above). (Otherwise, L_d is recursive by Theorem 9.4 and so is RE too by definition, contradiction!)
  - It can be shown that $\overline{L}_d$ is RE (omitted), just like the way we show the universal language L_u to be RE (shown next).
  - Note that $\overline{L}_d$ is the set of strings w_i such that the corresponding M_i accepts w_i.
  - In conclusion, $\overline{L}_d$ *is* RE but *not recursive* and so *undecidable*
  - That is, although there is a TM for $\overline{L}_d$, the problem defined by $\overline{L}_d$ *is undecidable* (i.e., there is no algorithm for it).
  - The universal language L_u has the *same* property, as proved later.

9.2.3 The Universal Language

- **Definition ---**

  The universal language L_u is the set of binary strings, each of which encodes, in the notation of 9.1.2, a pair (M, w), where M is a TM with the binary alphabet, and w is a string in $(0 + 1)^*$, such that w is in $L(M)$.

- It can be shown that there is a TM U, called *universal Turing machine*, which accepts L_u, i.e., $L(U) = L_u$.
  - For the proof, see the textbook (pp. 387~389). Easy in concept.
  - *Essence of proof:*
    - Construct a multi-tape TM U to simulate M on w, so that U accepts (M, w) if and only M accepts w.
    - So L_u is an RE language.
  - U is in the TM list mentioned previously.

- **Theorem 9.6 ---**

  L_u is RE but not recursive.

Proof.
  - *To prove the second half ("not recursive") by contradiction.*
  - We know L_u is RE. Now suppose L_u is recursive.
  - By Theorem 9.3, $\overline{L}_u$ is also recursive.
  - So we can construct a TM M to accept $\overline{L}_u$, and then we also can construct a TM M' to accept L_d (shown next).

- ♦ But this is contradictory, because we have know L_d is not RE.
- ♦ Therefore, the assumption L_u is recursive is wrong. Done.
- ♦ <u>What is left is to show the construction of TM *M′* to accept L_d.</u>

- ♦ The construction of *M′* is illustrated in Fig. 9.5 (Fig. 9.6 in the textbook) and described in detail as follows, which is based on the concept of *problem reduction* mentioned in the last chapter --- *reduction* of L_d to $\overline{L}_u$.
- ♦ Let *M* be the TM such that $L(M) = \overline{L}_u$.
- ♦ As shown in Fig. 9.6, we modify *M* into *M′* such that it accepts L_d as follows.
  - * Note that each input into *M* is a pair (M_i, w_i) where M_i is the code of a TM M_i and w_i is a binary input string into M_i.
  - * Also, note that each input into *M′* is the code of a TM because *M′* accept L_d.

- • Given input binary string *w*, *M′* changes it to *w*111*w* which means the pair (*M″*, *w*), where *M″* is the TM encoded by *w*. This can be done by a TM called "copy" as shown in Fig. 9.6.
- • If *w* into *M′* is w_i representing M_i in the TM list, then the input to the hypothetical algorithm *M* for $\overline{L}_u$ is (M_i, w_i). Also, since *M* accepts $\overline{L}_u$, this means that *M* accepts if and only if M_i does not accept w_i. This in turn means *M′* accepts if and only if M_i does not accept w_i.
- • In other words, *M′* accepts *w* if and only if *w* is in L_d.
- • That is, we have TM *M′* which accepts L_d, but this is impossible because L_d *is not an RE language*. Contradiction!

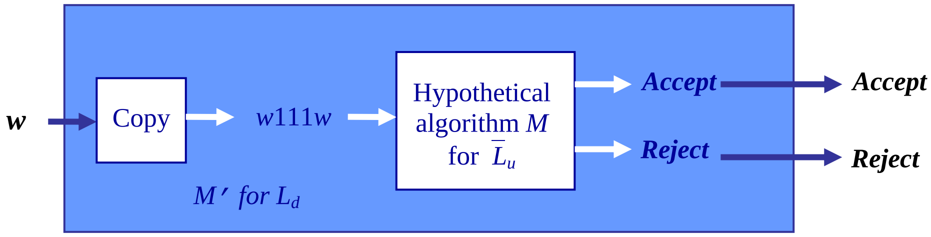

Fig. 9.5 A TM *M′* to accept L_d (Fig 9.6 in the textbook).