§9. The fall-out: part 2. In Note 7 we encountered a naturally defined language, namely L_{halt}, which is not recursive. In this note, we explore a little further, and discover languages that are not even recursively enumerable (r.e.). In a sense, a non-r.e. language is even 'harder' than L_{halt}. For although there is no Turing machine that accepts the language L_{halt} and *halts on all inputs*, there is at least a machine that accepts L_{halt} provided we do not expect a definite ruling on words outside L_{halt}. When a language is not r.e., even such a partial solution is denied.

§9.1. A language that is not recursively enumerable. Suppose $L \subseteq \Sigma^*$ is a language over Σ. The *complement of* L is the language $\overline{L} = \Sigma^* - L$; thus a word is in $\overline{L}$ if it is *not* in L, and vice versa. We open with some elementary facts about languages and their complements.

Theorem 9.1 *The complement of a recursive language is recursive.*

proof. Let L be any recursive language, and M a Turing machine that accepts L and halts on all inputs. We construct a Turing machine $\overline{M}$ that recognizes $\overline{L}$ and halts on all inputs.

All that is necessary is to interchange the roles of acceptance and rejection. Let Q be the state space of M, and let $q_F \in Q$ be the accepting state of M. To form $\overline{M}$ from M, augment Q by a single state $\overline{q}_F$, which becomes the new accepting state of $\overline{M}$. The old accepting state q_F now becomes a trap: after entering the state, $\overline{M}$ immediately halts without accepting. For all state/symbol pairs (q, s) on which the transition function of M is undefined, introduce a quintuple $(q, s, \overline{q}_F, s, R)$ into the transition function of $\overline{M}$. Observe that $\overline{M}$ accepts if M rejects, and vice versa. □

Theorem 9.2 *A language L is recursive if and only if both L and $\overline{L}$ are recursively enumerable.*

proof. The 'only if' part is immediate: by Theorem 9.1, both L and $\overline{L}$ are recursive and hence recursively enumerable.

For the 'if' part, let M and $\overline{M}$ be Turing machines that accept languages L and $\overline{L}$ (but which do not necessarily halt on all inputs). We construct a two-tape Turing machine $\widehat{M}$ that accepts L and halts on all inputs. In an initialization phase, $\widehat{M}$ shunts the input word x right one place, inserts a special end-of-tape symbol in the resulting gap, and copies the entire contents of tape 1 across to tape 2. Having returned both tape heads to the squares immediately to the right of the end-of-tape symbols, the simulation proper commences.

The machine $\widehat{M}$ simulates, move for move and concurrently, the computations of M and $\overline{M}$ on input x. The computation of M proceeds on tape 1, and that of $\overline{M}$ on tape 2. The presence of the end-of-tape markers enables $\widehat{M}$ to avoid crashing if the tape head of either machine falls off the end of the tape. Since the input x is either in L or $\overline{L}$, one of the simulated machines must eventually accept. At that point $\widehat{M}$ can pronounce on whether or not x is in the language L. $\square$

Since L_{halt} is r.e. but not recursive, it follows from Theorem 9.2 that the complement of L_{halt} is *not* r.e. Now consider the language

$$L_{\text{loop}} = \{\langle M\rangle\$x \mid M \text{ does } \textit{not} \text{ halt on input } x\}.$$

Note that the complement of L_{halt} is the union of L_{loop} and L_{dross}, where L_{dross} is the set of all badly formatted problem instances (i.e., all words in $\{\texttt{0},\texttt{1},\texttt{\$}\}^*$ that are not of the form $\langle M\rangle\$x$).

The language L_{loop} is our first example of a 'naturally defined' language that is not recursively enumerable.

COROLLARY 9.1 *The language L_{loop} is not recursively enumerable.*

PROOF. Given a Turing machine accepting the language L_{loop}, it would be straightforward to construct a Turing machine accepting the complement of L_{halt}. (The new machine merely adds a preprocessing phase that traps and accepts elements of L_{dross}.) But we know that the complement of L_{halt} is not r.e. $\square$

§9.2. The uniform halting problem (reprise). Recall that the uniform halting problem asks of a Turing machine whether it halts on all inputs. Intuitively, this seems a somewhat 'harder' problem than the question of whether a Turing machine halts on a specified input. In a sense this is the case: whereas L_{halt} is r.e., L_{uhalt} is not. However the fact the L_{uhalt} is not r.e. cannot be deduced from Theorem 9.2, since the complement of L_{uhalt} is not r.e. either. The language L_{uhalt} fails to be r.e. for a more interesting reason. First, let us observe an elementary fact about reductions.

THEOREM 9.3 *Suppose $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$ are languages. If L_1 is reducible to L_2, and L_2 is recursively enumerable, then L_1 is also recursively enumerable.*

PROOF. The proof exactly mirrors the proof of Theorem 8.1, and is left as a simple exercise. $\square$

We will often use this theorem in the contrapositive sense, i.e., if L_1 is reducible to L_2, and L_1 is *not* recursively enumerable, then L_2 is also not recursively enumerable.

THEOREM 9.4 *The language L_{uhalt} is not recursively enumerable.*

PROOF. We shall exhibit a reduction from L_{loop} to L_{uhalt}; the result will then follow from Corollary 9.1 and Theorem 9.3.

Let $\langle M\rangle\$x$ be a typical instance of the 'looping problem'. We shall demonstrate how to construct an instance $\langle M_x\rangle$ of the uniform halting problem that satisfies the following condition:

$$M \text{ loops on input } x \iff M_x \text{ halts on all inputs.} \qquad (1)$$

In fact, we shall construct a *two-tape* machine M'_x that has the appropriate behaviour, and then use earlier results (Lemma 4.2 and Lemma 7.1) to transform M'_x into an equivalent *binary* Turing machine M_x satisfying (1).

On input $w \in \{0,1\}^*$ (which we recall is presented on tape 1), M'_x operates as follows.

(a) On tape 2, M'_x writes a special end-of-tape symbol followed by the word x. Since x is a *fixed* word, this procedure can be hard-wired into the finite control of M'_x, just as in the proof of Theorem 8.2.

(b) M'_x uses tape 1 as a binary counter, interpreting the initial contents of the tape as the initial value of the counter. On tape 2, M'_x performs a simulation of the computation of M on input x. On each step of the simulation, M'_x decrements the counter on tape 1. The presence of the end-of-tape symbol enables M'_x to trap any situation in which M halts.

(c) The halting criterion for M'_x is as follows. If the simulation of M is still in progress when the counter reaches zero, then M'_x halts (say, rejecting). If the simulation of M terminates before the counter reaches zero, then M'_x enters an infinite loop.

If M does *not* halt on input x then M'_x will halt after W steps of the simulation, where W is the natural number represented by the binary input word w. However, if M *does* halt on input x then M'_x will loop on all sufficiently large inputs: simply take w to be the binary representation of any number that is larger than the number of steps taken by M on input x.

Now transform the machine M'_x just constructed into an equivalent binary Turing machine M_x. It should be clear that the resulting machine M_x satisfies (1). Finally, observe that (1) asserts that the function mapping $\langle M\rangle\$x$ to $\langle M_x\rangle$ is a reduction from L_{loop} to L_{uhalt}. By Corollary 9.1 and Theorem 9.3, the language L_{uhalt} is not recursively enumerable. □

EXERCISE Show that the *complement* of L_{uhalt} is not recursively enumerable. [Hint: this is an easier proof than the one above. Refer to the proof of Theorem 8.2.]

§9.3. Proof systems for the uniform halting problem. The statement that L_{uhalt} is not r.e. is clearly stronger than the statement that L_{uhalt} is not recursive. The stronger statement has a consequence which we shall now investigate.

Suppose we have a system in which it is possible to conduct formal proofs about the behaviour of Turing machines. That is, we have a formal language for making assertions about Turing machines, and a collection of axioms and inference rules. We shall make only two assumptions about this system:

(a) The language is able to express assertions of the form 'machine M halts on all inputs', where M is an arbitrary binary Turing machine.

(b) Proofs are machine checkable. By this we mean that there exists a Turing machine M_{check} that meets the following specification. Let M be a binary Turing machine, and π be a word over the alphabet $\{\mathtt{0}, \mathtt{1}\}$. On input $\langle M\rangle\$\pi$, the machine M_{check} is required to accept if π encodes a valid proof of the assertion 'M halts on all inputs', and to halt without accepting otherwise.

With these assumptions, it is easy to show that the proof system cannot be both sound, and complete with respect to assertions of the form 'machine M halts on all inputs'. That is, it is either:

Unsound: There are machines that can be proved to halt on all inputs, which do not in reality halt on all inputs; or

Incomplete: There are machines that halt on all inputs, which cannot be proved to halt on all inputs (recall Gödel's result discussed in §1.7).

For suppose, to the contrary, that a proof system exists that is both sound and complete. Then we could construct a multi-tape Turing machine M_{uhalt} that accepts the language L_{uhalt}. On input $\langle M\rangle$, the machine M_{uhalt} operates as follows. On one tape, M_{uhalt} cycles through all binary words π, using M_{check} to determine whether π encodes a valid proof of the assertion 'M halts on all inputs'. If M_{uhalt} ever discovers such a proof then it immediately halts and accepts. To cycle through all proofs, M_{uhalt} simply enumerates all binary strings on one of its tapes, one by one, in order of increasing length. (For a particular length n, M_{uhalt} starts with the string $\mathtt{0}^n$ and increments it by one (in binary) to get the next string until the string $\mathtt{1}^n$ is reached.) The contents of this tape form a binary word that can be interpreted as the encoding of a putative proof π. On finding that a word π does not encode a valid proof of the assertion, M_{uhalt} generates the next string and tries again.

If the proof system is sound and complete, the machine M_{uhalt} will eventually find a valid proof precisely if M halts on all inputs. Thus M_{uhalt} accepts the language L_{uhalt}. But Theorem 9.4 denies the existence of such a machine.