§8. The fall-out: part 1. Using the undecidability of the halting problem as a starting point, it is possible to demonstrate that many other naturally defined problems are undecidable. Before embarking on such a programme, we shall equip ourselves with an important tool.

§8.1. Reductions. Let L_1 and L_2 be languages over alphabets Σ_1 and Σ_2, respectively. A *reduction from* L_1 *to* L_2 is a function $f : \Sigma_1^* \to \Sigma_2^*$ such that:

(a) $x \in L_1 \iff f(x) \in L_2, \quad$ for all $x \in \Sigma_1^*$;

(b) there is a Turing machine transducer that computes f.

In other words the question 'is $x \in L_1$?' has the same answer as the question 'is $f(x) \in L_2$?'; moreover we have an algorithm for transforming the first question to the second one. If a reduction from L_1 to L_2 exists then we say that L_1 *is reducible to* L_2. Reductions play an important role in proofs of undecidability, which is clarified in the following result.

THEOREM 8.1 *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 recursive, then L_1 is also recursive.*

PROOF. Since L_2 is recursive, there is a Turing machine M_2 that accepts L_2 and halts on all inputs. Further, since L_1 is reducible to L_2, there is a function $f : \Sigma_1^* \to \Sigma_2^*$ satisfying conditions (a) and (b) above. We shall use these observations to construct a machine M_1 that accepts the language L_1 and halts on all inputs. It will follow that the language L_1 is recursive.

On input $x \in \Sigma_1^*$ the machine M_1 operates as follows. First, M_1 places a special end-of-tape marker on the leftmost square of its tape, shunts the input x right one place, and returns to the square immediately to the right of the marker. Then M_1 computes $f(x) \in \Sigma_2^*$ leaving the result on the tape; that this can be done is assured by condition (b) above. Finally, having returned the tape head to the square immediately to the right of the end-of-tape marker (having the marker means that we can avoid falling off the left end of the tape at this stage), M_1 behaves exactly like M_2. By condition (a), the machine M_1 accepts the language L_1. Further, since M_2 halts on all inputs, so also does M_1. $\square$

Note that the proof is just a Turing machine version of the following simple idea: first pre-process the input x by computing $f(x)$, then run the decision procedure for L_2 (with input $f(x)$).

An equivalent statement of Theorem 8.1 is that if L_1 is reducible to L_2, and L_1 is *not* recursive, then neither is L_2. It is generally in this form that we shall use the theorem. We know that L_{halt} is not recursive; we will extend our knowledge to show that other other languages are non-recursive by reducing L_{halt} to them. This will then give us more examples as starting points for reductions.

§8.2. The uniform halting problem. The *uniform halting problem* is as follows:

INSTANCE: A binary Turing machine M.

QUESTION: Does M halt on *all* inputs $x \in \{\mathtt{0},\mathtt{1}\}^*$?

Alternatively, we may view the uniform halting problem as the task of recognizing

$$L_{\text{uhalt}} = \{\langle M\rangle \mid M \text{ halts on all inputs } x \in \{\mathtt{0},\mathtt{1}\}^*\}.$$

It seems likely, given our experience with the halting problem itself, that the uniform halting problem is undecidable. However, since it is conceivable that determining whether a machine halts *in general* is easier than deciding whether it halts *on a particular input*, we really ought to prove that the uniform halting problem is undecidable.

THEOREM 8.2 *The language L_{uhalt} is not recursive.*

PROOF. We shall exhibit a reduction from L_{halt} to L_{uhalt}.

Consider a typical instance $\langle M\rangle\$x$ of the halting problem. (We assume that the instance has the appropriate format, i.e., that there is a single dollar symbol and the binary word to the left of the dollar symbol is a valid encoding of a Turing machine.) Recall that the halting problem asks the question: "Does M halt on input x?" We now construct a binary Turing machine M_x with the property that

$$M \text{ halts on input } x \iff M_x \text{ halts on all inputs.} \tag{1}$$

The implementation strategy is simple. The machine M_x compares its input $w \in \{\mathtt{0},\mathtt{1}\}^*$ with the word x: if $w \neq x$ then M_x immediately halts; if $w = x$ then M_x returns its head to the leftmost square of the tape and then behaves exactly like M.

In more detail, M_x is constructed from M as follows. Let $x = x_0x_1x_2\cdots x_{n-1}$, where each x_i is either $\mathtt{0}$ or $\mathtt{1}$. First, augment the states of M by adding $2n$ new states $q_0', q_1', \ldots, q_{n-1}'$, and $q_1'', q_2'', \ldots, q_n''$. Then extend the transition function to these new states by adding the right-sweeping quintuples $(q_0', x_0, q_1', x_0, R)$, $(q_1', x_1, q_2', x_1, R)$, $\ldots$, $(q_{n-2}', x_{n-2}, q_{n-1}', x_{n-1}, R)$, $(q_{n-1}', x_{n-1}, q_n'', x_{n-1}, R)$, and the left-sweeping quintuples $(q_n'', \mathfrak{b}, q_{n-1}'', \mathfrak{b}, L)$, $(q_{n-1}'', x_{n-1}, q_{n-2}'', x_{n-1}, L)$, $\ldots$, $(q_2'', x_2, q_1'', x_2, L)$, $(q_1'', x_1, q_I, x_1, L)$, where q_I is the initial state of M. Note that, on input w, the machine M_x halts and rejects if $w \neq x$, and operates exactly like M if $w = x$. Thus the behaviour of M_x satisfies (1). Note also that $\langle M_x\rangle$ (the encoding of M_x) is easy to compute given $\langle M\rangle$ (the encoding of M) and x.

Now, equivalence (1) may be rewritten

$$\langle M\rangle\$x \in L_{\text{halt}} \iff \langle M_x\rangle \in L_{\text{uhalt}},$$

whence it is clear that the function f that maps each word of the form $\langle M\rangle\$x$ to the word $\langle M_x\rangle$ is a reduction from L_{halt} to L_{uhalt}.[20] The result then follows from Theorem 8.1 together with the fact that L_{halt} is not recursive (see NOTE 7). $\square$

§8.3. The non-emptiness problem for r.e. languages. For any Turing machine M, let $L(M)$ denote the language accepted by M. The *non-emptiness problem for r.e. languages* is the following:

INSTANCE: A binary Turing machine M.

QUESTION: Is the language $L(M)$ non-empty?

As before, the problem can be recast as the task of recognizing an appropriately defined language, in this instance $L_{\text{ne}} = \{\langle M\rangle \mid L(M) \neq \emptyset\}$.

THEOREM 8.3 *The language L_{ne} is not recursive.*

PROOF. Again, we employ a reduction from the language L_{halt}. Consider a typical instance $\langle M\rangle\$x$ of the halting problem. We demonstrate how to construct a Turing machine M_x with the property that

$$M \text{ halts on input } x \iff L(M_x) \neq \emptyset. \tag{2}$$

First of all note that we can ensure that M does not fall off the left hand end of the tape (just construct an equivalent machine that avoids this behaviour). Now, construct a machine M' which is a very simple modification of M. For all state/symbol pairs (q, s) on which the transition function of M is undefined, introduce a new tuple (q, s, q_F, s, R) into the specification of the transition function of M'. Then M' behaves exactly like M until such time as M would reject; at that point M' accepts. Thus

$$M \text{ halts on input } x \iff M' \text{ accepts input } x.$$

We are now in a position to construct M_x itself. The machine M_x compares its input $w \in \{\mathtt{0}, \mathtt{1}\}^*$ with the word x: if $w \neq x$ then M_x immediately halts and rejects; if $w = x$ then M_x returns its head to the leftmost square of the tape and proceeds to behave exactly like M'. Note that M_x bears exactly the same relation to M' in this proof as its namesake did to M in the proof of Theorem 8.2. Therefore, exactly the same detailed construction can be employed. Note also that the machine M_x has property (2).

[20]For this statement to be entirely correct, one would need to extend f to badly formatted instances of the halting problem. This could be achieved by mapping each badly formatted instance to a binary word that is *not* a member of L_{uhalt}. This kind of unedifying technical detail can safely be swept under the carpet.

Let f be the function that maps each instance $\langle M \rangle \$ x$ of the halting problem to the instance $\langle M_x \rangle$ of the non-emptiness problem. Observe that (2) asserts that the function f is a reduction from L_{halt} to L_{ne}. The result now follows from Theorem 8.1 and the fact that L_{halt} is not recursive. $\square$

§8.4. Number theory: a simple first-order theory. Consider sentences formed from the following entities, according to 'appropriate syntactic rules', with brackets being used to resolve ambiguities.

(a) the constants 0 and 1;

(b) variables (which we shall denote by lower case roman letters);

(c) the binary arithmetic operators $+$ and $\times$;

(d) the relational operators $<$ and $=$;

(e) the logical connectives $\wedge$, $\vee$, and $\neg$;

(f) the quantifiers $\exists$ (there exists) and $\forall$ (for all).

(We shall not pause to say what the 'appropriate syntactic rules' are, but instead leave them to the imagination.) Each syntactically valid sentence can be interpreted as a statement about the natural numbers. As long as we stipulate that sentences should not contain free variables (i.e., every variable is bound by some quantifier), every sentence so interpreted will either be true or false.

Thus, for example, $\forall x\, \exists y\, [x < y]$ is the true assertion that for every natural number there is a greater natural number, whereas $\forall x\, \exists y\, [x = y + y]$ is the false assertion that every natural number is even. These are very simple examples, but even with very few symbols it is possible to make non-trivial assertions. For example, suppose we allow prime(x) as a macro for the predicate $\forall u\, \forall v\, [(u = 1) \vee (v = 1) \vee \neg(u \times v = x)]$, which asserts that x is a prime number. Then the sentence $\forall x\, \exists y\, [(x < y) \wedge \text{prime}(y)]$ asserts that there are an infinity of prime numbers, and the sentence $\forall x\, \exists y\, [(x < y) \wedge \text{prime}(y) \wedge \text{prime}(y + 1 + 1)]$ asserts that there are an infinity of 'prime pairs'. Note that the latter sentence is nothing more than a conjecture: it is not known whether the sentence is true or false!

Let L_{num} be the set of *true* sentences in the above 'theory of numbers'. Clearly, it would be of great interest to have an effective procedure that decides for any sentence in this 'theory of numbers' whether it is true or false. The final result of this note, a consequence of Kurt Gödel's theorem (1931) discussed in §1.7, denies the existence of such a procedure.

Theorem 8.4 *The language L_{num} is not recursive.*

Exercise Prove that there are infinitely many primes.