Decidability — Truth or Turing?

Richard W. DeVaul
Aesthetics and Computation Group, MIT Media Lab

June 10, 1998 Revision: 1.4 α

Abstract

This short paper covers the substance of a talk on Turing decidability presented to the Babbage group on Thursday, June 11, 1998.

Contents

1 Background for Decidability **2**
- 1.1 Turing Machines briefly defined . . . 2
- 1.2 Procedures and Functions . . . 2
- 1.3 Problems . . . 3
  - 1.3.1 Reducibility . . . 3
  - 1.3.2 Encoding and Algorithms . . . 3

2 Recursive and Recursively Enumerable Languages **3**
- 2.1 Recursively Enumerable Languages . . . 3
  - 2.1.1 Recursive Languages . . . 4
- 2.2 Are All Languages Recursively Enumerable? . . . 4

3 Decidability **4**
- 3.1 Trivial Decidability . . . 4
- 3.2 Decidable Doesn't Mean Solved . . . 4
- 3.3 Decidability Theorems . . . 5
  - 3.3.1 Theorem 1 . . . 5
  - 3.3.2 Theorem 2 . . . 5
  - 3.3.3 Theorem 3 . . . 5
- 3.4 Decidability analysis . . . 5
  - 3.4.1 Strategy for decidability . . . 5
  - 3.4.2 A non-r.e. language: L_d . . . 5
  - 3.4.3 A r.e, non-recursive language: L_u . . . 6
  - 3.4.4 Rice's Theorem . . . 6
  - 3.4.5 Post's Correspondence Problem . . . 6

1 Background for Decidability

1.1 Turing Machines briefly defined

A Turing machine is composed of an infinite tape bounded on the left, a read-write tape head, and a finite control.

A Turing Machine (TM) is described by the tuple[1]

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)\,, \tag{1}$$

where

Q is the finite set of states,

Γ is the the finite set of allowable tape symbols,

B is the blank symbol ($B \in \Gamma$).

Σ is the set of input symbols, $\Sigma \subset \Gamma, B \notin \Sigma$.

δ is the next move function, $\delta(Q \times \Gamma) \rightarrow Q \times \Gamma \times \{L, R\}$.

q_0 is the start state ($q_0 \in Q$).

F is the set of final states, $F \subseteq Q$.

For each symbol scanned, a TM:

1. changes state,
2. prints a symbol on tape cell, replacing what was written there before, and
3. moves the tape head left or right one cell.

1.2 Procedures and Functions

Procedures equivalent to TM's A procedure is equivalent to a TM, and all TM's can be encoded as finite binary strings [Hopcroft, Section 8.3]. This means that the set of TM's encoded in binary is a subset of the integers encoded in binary (not all binary integers code for a TM). For this reason, the members of the set of integers can be put in a one-to-one correspondence with the members of the set of TMs (procedures).

More functions than procedures The set of functions mapping integers into the set $0, 1$ is at least as large as the set of real numbers (consider $f(i) = \mathrm{mod}(i * j, 2)\ \forall\ j \in \mathbf{I}$). Since the set of integers cannot be put in one-to-one correspondence with the set of reals, there exist functions for which there are no corresponding procedures. This is important because it means that some functions can not be computed [Hopcroft, p. 147].

[1] Hopcroft and Ullman, Section 7.2, p. 148.

1.3 Problems

A problem is a yes-no question A problem is essentially a question for which we want an answer yes-no answer [Hopcroft p. 177], which can be thought of as a boolean function, the domain of which are the instances of the problem.

1.3.1 Reducibility

A reducible to B means A at least as hard as B Restricting our consideration to boolean problems has little impact on decidability, since many general problems are *reducible* to (at least as hard as) the yes-no problem. As an example, the yes-no problem of deciding the ambiguity of a context free grammar is called AMB. The general case of AMB called FIND (which produces a word with two or more parses) is reducible to AMB. Since no algorithm for AMB exists, no algorithm for FIND can exist [Hopcroft p. 178].

1.3.2 Encoding and Algorithms

Problem instances as strings, problem as language By restricting consideration to boolean problems, and encoding instances of the problem by strings over a finite alphabet, we can transform the question of whether there exists an algorithm for solving a problem to whether or not a particular language is recursive [Hopcroft p. 177]. (A problem can be represented as the language of instance strings)

2 Recursive and Recursively Enumerable Languages

2.1 Recursively Enumerable Languages

r.e. languages enumerable by TMs The set of recursively enumerable (r.e.) languages are precisely those languages which can be listed (enumerated) by a TM. Basically, if a TM can generate all the strings in a language, a TM can accept all the strings in a language.

Enumeration means listing only valid strings An important point is that enumeration involves listing *only those strings in the language* — there can be repetitions and the strings can be generated in any order, but no illegal strings ($w \notin L(M)$) can be generated in the process of generating the legal strings.

Some r.e. languages have no always-halting TM However, just because a TM can generate all the strings in a r.e. language does not mean that a TM can determine if a particular string is a member of the language. The class of r.e. languages include some languages for which it is impossible to mechanically determine membership, *i.e.*, if $L(M)$ is such a language, then any

TM recognizing $L(M)$ must fail to halt on some input not in $L(M)$ [Hopcroft, p. 150].

TMs always halt for w in $L(M)$ It is important to note that $L(M)$ will still halt for any w in $L(M)$ — that is the definition for the acceptance of a language by a TM.

2.1.1 Recursive Languages

Recursive languages = TMs that always halt A subset of r.e. languages is the recursive languages. A recursive language $L(M)$ which can be accepted by at least one TM guaranteed to halt on all its inputs, whether or not the input is a member of $L(M)$. This does not mean that all inputs are accepted, but rather that all inputs can be either accepted or rejected in a finite time. Such a TM will *always halt*.

2.2 Are All Languages Recursively Enumerable?

Some languages not r.e. No. All languages are not recursively enumerable. This is not surprising, since the set of recursively enumerable languages is the same as the set of languages accepted by TMs, which we have previous said is equivalent to the set of procedures. Just as some functions have no corresponding procedure, some languages cannot be accepted by a TM.

3 Decidability

problems with recursive languages decidable Since yes-no problems can be posed as languages, problems with a recursive languages are *decidable*. This is because for recursive languages there exist TMs which will say yes or no for all inputs.

3.1 Trivial Decidability

one string = recursive language One consequence of the previous statement is that all problems with a single instance are trivially decidable — a language with one string is always recursive.

3.2 Decidable Doesn't Mean Solved

***e.g.* Fermat's conjecture** Fermat's conjecture is decidable, since there is only one instance of the problem. There exists a TM that accepts any input and one that rejects any input – one of these is right but it is not obvious which one [Hopcroft P. 179].

decidable doesn't mean we know how to solve it The above example points out that just because a problem is decidable doesn't mean that we know which algorithm will decide it *correctly*, only that such an algorithm exists.

3.3 Decidability Theorems

3.3.1 Theorem 1

The complement of a recursive language is recursive Append a logical not to any algorithm, and it's still an algorithm [Hopcroft, Theorem 8.1].

3.3.2 Theorem 2

The union of two recursive languages is recursive Run the two algorithms sequentially or in parallel; If each will halt independently, the union will halt [Hopcroft, Theorem 8.2].

3.3.3 Theorem 3

If a language and it's complement are r.e, they are both recursive This is tricky, but a TM M accepting an r.e. language will always halt for $w \in L(M)$. Since w is always in either L or $\bar{L}$, either M or M' will always halt. But, by Theorem 1, the complement of a recursive language is recursive. Therefor, both L and $\bar{L}$ are recursive if both are r.e.

Implications of Theorem 3 This theorem is very important. Let L and $\bar{L}$ be complementary languages. Then one of three cases is true:

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

.

3.4 Decidability analysis

3.4.1 Strategy for decidability

first principles or reduction Decidability can be determined by the application of the previously mentioned decidability theorems, or reducing a problem to another problem of previously determined decidability.

3.4.2 A non-r.e. language: L_d

definition of w_i and M_j Given two integers i and j, word w_i is defined as the binary code for i and M_j as the TM who's code is the binary code for j. If j does not code for a valid TM, then it is considered to be a TM which accepts the empty set $(L(M) = \emptyset)$.

acceptance table Imagine i and j are arranged in an infinite table, where the entries in the table are boolean values indicating the acceptance of word w_i by TM M_j. The diagonal of this table ($i = j$) are the entries for which the binary code of w_i and M_j are the same. The language L_d is defined as those w_i for which the (i, i) table entry is 0 (false), *i.e* L_d contains only those strings which do not code for TMs accepting themselves.

The language L_d is not r.e. Suppose that some TM M_j accepts L_d. In that case, entry (j, j) must be zero by definition of L_d. This means that w_j is not in $L(M_j)$, which contradicts $L(M_j) = L_d$. Likewise, if entry (j, j) is one, then w_j is not in L_d, which contradicts $L(M_j) = L_d$. Therefor, L_d is not r.e.

3.4.3 A r.e, non-recursive language: L_u

The Universal Language The universal language L_u is defined as the set $\{< M, w > |M \text{ accepts } w\}$. This language is recursively enumerable, but not recursive. This is because a TM can be constructed which, when given $< M, w >$ will run M on w, and will accept if and only if $< M, w >$ is in L_u. The language is not recursive because no TM can decide if $< M, w >$ will ever halt (the classic halting problem).

3.4.4 Rice's Theorem

Any nontrivial property P of the r.e. languages is undecidable. This means that for r.e. sets you can't decide:

1. emptiness,
2. finiteness,
3. regularity,
4. context-freedom.

Rice's theorem applies to languages, not to TMs You can't say anything about r.e. languages, but you can still talk about TMs, *e.g.* does a TM have an even number of states?

3.4.5 Post's Correspondence Problem

Post's correspondence problem is an undecidable problem useful in categorizing other problems as undecidable.