بسم اللّه الرّحمن الرّحیم

دانشگاه صنعتی اصفهان ـ دانشکدهٔ مهندسی برق و کامپیوتر
(نیم‌سال تحصیلی ۴۰۰۱)

نظریهٔ زبان‌ها و ماشین‌ها

حسین فلسفین

دو مثال پایانی بحث کاهش‌پذیری

The following two decision problems are undecidable.
1. $A\varepsilon_{TM}$: Given a TM T, is $\varepsilon \in L(T)$?
2. ALL_{TM}: Given a TM T with input alphabet Σ, is $L(T) = \Sigma^*$?

Proof. *In each case, we will construct a reduction to the given problem from another decision problem that we already know is undecidable.*

1.$A_{TM} \leq A\varepsilon_{TM}$: *An instance of A_{TM} is a pair $\langle T, x \rangle$ containing a TM and a string. We must show how to construct for every pair like this a TM $F(T, x) = T_1$ (an instance of $A\varepsilon_{TM}$) such that for every pair (T, x), T accepts x if and only if T_1 accepts ε. We have to construct T_1 such that even though we don't know what the outcome will be when T processes x, the outcome when T_1 processes ε will be the same—or at least if the first outcome is acceptance, then the second will be, and if the first is not, then the second is not. This seems very confusing if you're thinking about what a TM might "normally" do on input ε, because it probably seems to have nothing to do with x! But keep in mind that as long as ε is the original input, then no matter what other string T_1 writes on the tape before starting a more "normal" computation, it's ε that will be accepted or not accepted. If we want T_1 to perform a computation on ε that has something to do with the computation of T on x, then we can simply make the computations identical, except that first we*

have to put x on the tape before the rest of the computation proceeds. That's the key to the reduction. We let $T_1 = Write(x)T$ the composite TM that first writes x on the tape, beginning in square 1, and then executes T. If T accepts x, then when T is called in the computation of T_1, it will process x as if it were the original input, and ε will be accepted; otherwise the second phase of T_1's computation will not end in acceptance, and ε will not be accepted. It may not be clear what T_1 is supposed to do if the original input is not ε. It doesn't matter, because whether or not the algorithm that comes up with T_1 is a correct reduction depends only on what T_1 does with the input ε.

2. $A_{\varepsilon TM} \leq ALL_{TM}$: Both problems have instances that are TMs. Starting with an arbitrary TM T, we must find another TM $F(T) = T_1$ such that if T accepts ε then T_1 accepts every string over its input alphabet, and if T doesn't accept ε then there is at least one string not accepted by T_1. It may be a little easier after the first reduction to see what to do here. This time, if T_1 starts with an input string x, it's x that is accepted if T_1 eventually reaches the accepting state, even if the computation resembles the one that T performs with input ε. We can define T_1 to be the composite Turing machine $Erase\ T$ where $Erase$ erases the input string and leaves the tape head on square 0. For every input, whether T_1 reaches q_{accept} depends only on whether T accepts ε. If T accepts ε, T_1 accepts everything, and otherwise T_1 accepts nothing.

For some infinite sets, no correspondence with $\mathbb{N}$ exists. These sets are simply too big. Such sets are called uncountable. The set of real numbers is an example of an uncountable set. A real number is one that has a decimal representation. The numbers $\pi = 3.1415926\ldots$ and $\sqrt{2} = 1.4142135\ldots$ are examples of real numbers. Let $\mathbb{R}$ be the set of real numbers. Cantor proved that $\mathbb{R}$ is uncountable. In doing so, he introduced the diagonalization method.

Theorem: $\mathbb{R}$ *is uncountable.*

Proof: *In order to show that* $\mathbb{R}$ *is uncountable, we show that no correspondence exists between* $\mathbb{N}$ *and* $\mathbb{R}$*. The proof is by contradiction. Suppose that a correspondence* f *existed between* $\mathbb{N}$ *and* $\mathbb{R}$*. Our job is to show that* f *fails to work as it should. For it to be a correspondence,* f *must pair all the members of* $\mathbb{N}$ *with all the members of* $\mathbb{R}$*. But we will find an* x *in* $\mathbb{R}$ *that is not paired with anything in* $\mathbb{N}$*, which will be our contradiction. The way we find this* x *is by actually constructing it. We choose each digit of* x *to make* x *different from one of the real numbers that is paired with an element of* $\mathbb{N}$*. In the end, we are sure that* x *is different from any real number that is paired. We can illustrate this idea by giving an example. Suppose that the correspondence* f *exists. Let* $f(1) = 3.14159\ldots$*,* $f(2) = 55.55555\ldots$*,* $f(3) = \ldots$*, and so on, just to make up some values for* f*. Then* f *pairs the number* 1 *with* $3.14159\ldots$*, the number* 2 *with* $55.55555\ldots$*, and so on.*

The following table shows a few values of a hypothetical correspondence f between $\mathbb{N}$ and $\mathbb{R}$.

n	$f(n)$
1	$3.14159\ldots$
2	$55.55555\ldots$
3	$0.12345\ldots$
4	$0.50000\ldots$
$\vdots$	$\vdots$

We construct the desired x by giving its decimal representation. It is a number between 0 and 1, so all its significant digits are fractional digits following the decimal point. Our objective is to ensure that $x \neq f(n)$ for any n. To ensure that $x \neq f(1)$, we let the first digit of x be anything different from the first fractional digit 1 of $f(1) = 3.\underline{1}4159\ldots$. Arbitrarily, we let it be 4. To ensure that $x \neq f(2)$, we let the second digit of x be anything different from the

second fractional digit 5 *of* $f(2) = 55.5\underline{5}5555\ldots$. *Arbitrarily, we let it be* 6. *The third fractional digit of* $f(3) = 0.12\underline{3}45\ldots$ *is* 3, *so we let* x *be anything different—say,* 4. *Continuing in this way down the diagonal of the table for* f, *we obtain all the digits of* x, *as shown in the following table. We know that* x *is not* $f(n)$ *for any* n *because it differs from* $f(n)$ *in the* n*th fractional digit. (A slight problem arises because certain numbers, such as* $0.1999\ldots$ *and* $0.2000\ldots$, *are equal even though their decimal representations are different. We avoid this problem by never selecting the digits* 0 *or* 9 *when we construct* x.*)*

n	$f(n)$
1	$3.\underline{1}4159\ldots$
2	$55.5\underline{5}555\ldots$
3	$0.12\underline{3}45\ldots$
4	$0.500\underline{0}\ldots$
⋮	⋮

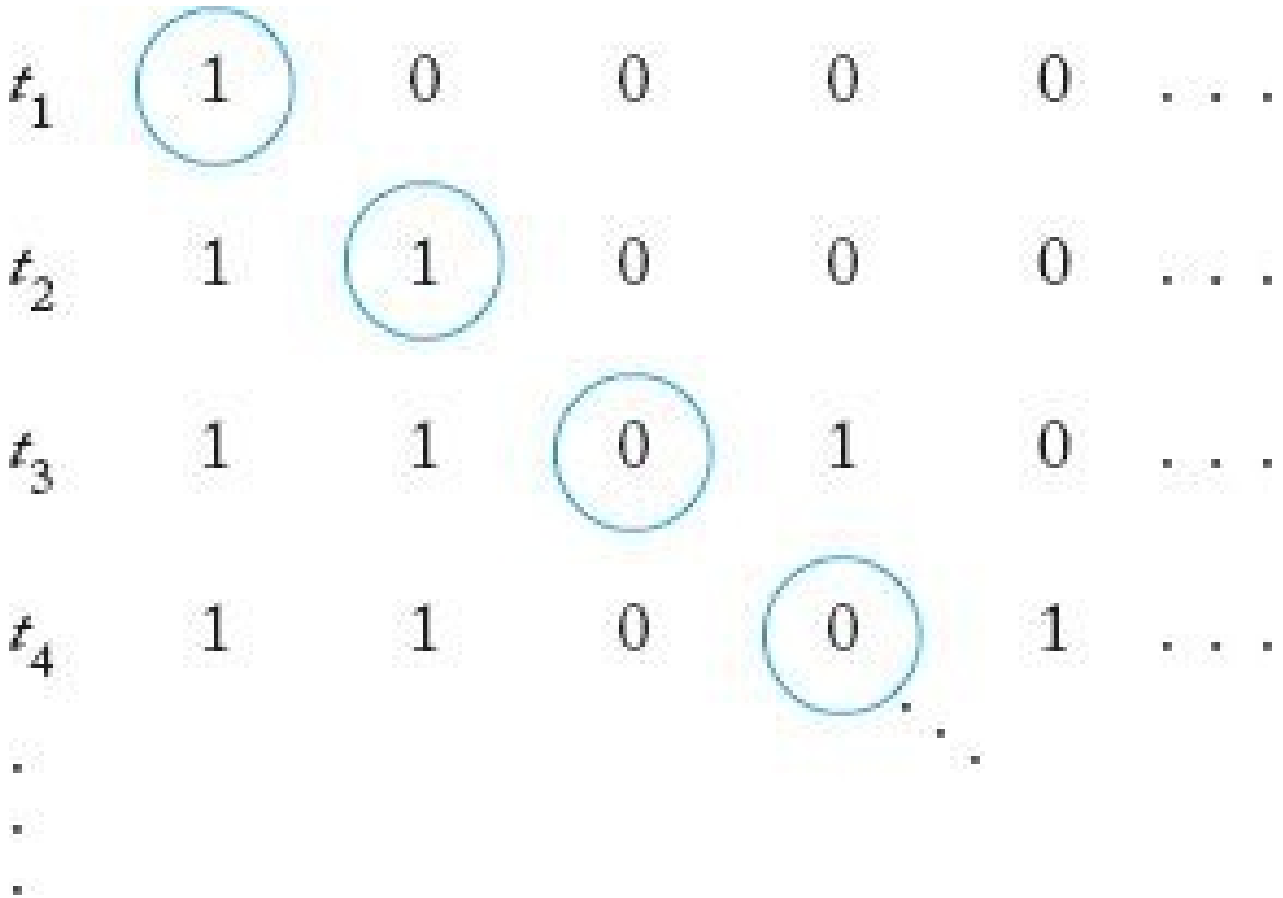
$x = 0.4641\ldots$

زبان‌هایی وجود دارند که حتی *Turing-Recognizable* هم نیستند

The preceding theorem has an important application to the theory of computation. It shows that some languages are not decidable or even Turing-recognizable, for the reason that there are uncountably many languages yet only countably many Turing machines. Because each Turing machine can recognize a single language and there are more languages than Turing machines, some languages are not recognized by any Turing machine. Such languages are not Turing-recognizable. We can establish the existence of languages that are not recursively enumerable in a variety of ways.

Theorem: *Let S be an infinite countable set. Then its power set 2^S is not countable.*

Proof: *Let $S = \{s_1, s_2, s_3, \ldots\}$. Then any element t of 2^S can be represented by a sequence of 0's and 1's, with a 1 in position i if and only if s_i is in t. For example, the set $\{s_2, s_3, s_6\}$ is represented by $01100100\ldots$, while $\{s_1, s_3, s_5, \ldots\}$ is represented by $10101\ldots$. Clearly, any element of 2^S can be represented by such a sequence, and any such sequence represents a unique element of 2^S. Suppose that 2^S were countable; then its elements could be written in some order, say $t_1, t_2, \ldots$, and we could enter these into a table, as shown in the following figure.*


t_1 1 0 0 0 0 . . .
t_2 1 1 0 0 0 . . .
t_3 1 1 0 1 0 . . .
t_4 1 1 0 0 1 . . .

In this table, take the elements in the main diagonal, and complement each entry, that is, replace 0 with 1, and vice versa. In the example in the above figure, the elements are $1100\ldots$, so we get $0011\ldots$ as the result. The new sequence along the diagonal represents some element of 2^S, say t_i for some i. But it cannot be t_1 because it differs from t_1 through s_1. For the same reason it cannot be t_2, t_3, or any other entry in the enumeration. This contradiction creates a logical impasse that can be removed only by throwing out the assumption that 2^S is countable.

This kind of argument, because it involves a manipulation of the diagonal elements of a table, is called diagonalization. The technique is attributed to the mathematician G. F. Cantor, who used it to demonstrate that the set of real numbers is not countable.

Languages That Are Not Recursively Enumerable

Fact: Some languages are not Turing-recognizable.

Proof: The set of all Turing machines is countable. To show that the set of all languages is uncountable, we first observe that the set of all infinite binary sequences is uncountable. An infinite binary sequence is an unending sequence of 0s and 1s. Let $\mathcal{B}$ be the set of all infinite binary sequences. We can show that $\mathcal{B}$ is uncountable by using a proof by diagonalization similar to the one we used to show that $\mathbb{R}$ is uncountable. Let $\mathcal{L}$ be the set of all languages over alphabet Σ. We show that $\mathcal{L}$ is uncountable by giving a correspondence with $\mathcal{B}$, thus showing that the two sets are the same size. Let $\Sigma^* = \{s_1, s_2, s_3, \ldots\}$. Each language $A \in \mathcal{L}$ has a unique sequence in $\mathcal{B}$. The ith bit of that sequence is a 1 if $s_i \in A$ and is a 0 if $s_i \notin A$, which is called the characteristic sequence of A. For example, if A were the language of all strings starting with a 0 over the alphabet

$\{0, 1\}$, *its characteristic sequence* χ_A *would be*

$$\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\};$$

$$A = \{0, 00, 01, 000, 001, \ldots\};$$

$$\chi_A = 010110011 \cdots .$$

The function $f : \mathcal{L} \mapsto \mathcal{B}$, *where* $f(A)$ *equals the characteristic sequence of* A, *is one-to-one and onto, and hence is a correspondence. Therefore, as* $\mathcal{B}$ *is uncountable,* $\mathcal{L}$ *is uncountable as well. Thus we have shown that the set of all languages cannot be put into a correspondence with the set of all Turing machines. We conclude that some languages are not recognized by any Turing machine.*

A Turing-Unrecognizable Language

In the preceding sessions, we exhibited a language—namely, A_{TM}—that is undecidable. Now we exhibit a language that isn't even Turing-recognizable. Note that A_{TM} will not suffice for this purpose because we showed that A_{TM} is Turing-recognizable. We show that if both a language and its complement are Turing-recognizable, the language is decidable. Hence for any undecidable language, either it or its complement is not Turing-recognizable. Recall that the complement of a language is the language consisting of all strings that are not in the language. We say that a language is co-Turing-recognizable if it is the complement of a Turing-recognizable language.

Theorem: *If L is a recursive language over Σ, then its complement $\overline{L}$ is also recursive.*

Proof: *If T is a TM that decides L, the TM obtained from T by interchanging the two outputs decides $\overline{L}$. (Assume that L is recursive. Then there exists a decider for it. But this becomes a decider for $\overline{L}$ by simply complementing its conclusion. Therefore, $\overline{L}$ is recursive.)*

Theorem: ***A language is decidable*** *iff it is* ***Turing-recognizable*** *and* ***co-Turing-recognizable.*** *In other words, a language is decidable exactly when* ***both it and its complement are Turing-recognizable.***

Proof: *We have two directions to prove.* *First,* *if A is decidable, we can easily see that both A and its complement $\overline{A}$ are Turing-recognizable. Any decidable language is Turing-recognizable, and the complement of a decidable language also is decidable.* *For the other direction,* *if both A and $\overline{A}$ are Turing-recognizable, we let M_1 be the recognizer for A and M_2 be the recognizer for $\overline{A}$. The following Turing machine M is a decider for A.*

$M =$ "On input w:

1. Run both M_1 and M_2 on input w in parallel.
2. If M_1 accepts, accept; if M_2 accepts, reject."

Running the two machines in parallel means that M has two tapes, one for simulating M_1 and the other for simulating M_2. In this case, M takes turns simulating one step of each machine, which continues until one of them accepts. Now we show that M decides A. Every string w is either in A or $\overline{A}$. Therefore, either M_1 or M_2 must accept w. Because M halts whenever M_1 or M_2 accepts, M always halts and so it is a decider. Furthermore, it accepts all strings in A and rejects all strings not in A. So M is a decider for A, and thus A is decidable.

بیان دیگر برای قضیهٔ بالا

Theorem: *If L is a recursively enumerable language, and its complement $\overline{L}$ is also recursively enumerable, then L is recursive (and therefore $\overline{L}$ is recursive).*

Proof: *If T is a TM accepting L, and T_1 is another TM accepting $\overline{L}$, then here is an algorithm to decide L: For a string x, execute T and T_1 simultaneously on input x, until one halts. (One will eventually accept, because either $x \in L$ or $x \in \overline{L}$.) If the one that halts first is T, and it accepts, or the one that halts first is T_1, and it rejects, return 1; otherwise return 0.*

معرفی یک زبان که حتی *Turing-recognizable* هم نیست

Corollary: $\overline{A_{TM}}$ ***is not Turing-recognizable.***

Proof: *We know that* A_{TM} *is Turing-recognizable. If* $\overline{A_{TM}}$ *also were Turing-recognizable,* A_{TM} *would be decidable. We know that* A_{TM} *is not decidable, so* $\overline{A_{TM}}$ *must not be Turing-recognizable.*