بسم اللّه الرّحمن الرّحیم

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

نظریهٔ زبان‌ها و ماشین‌ها

حسین فلسفین

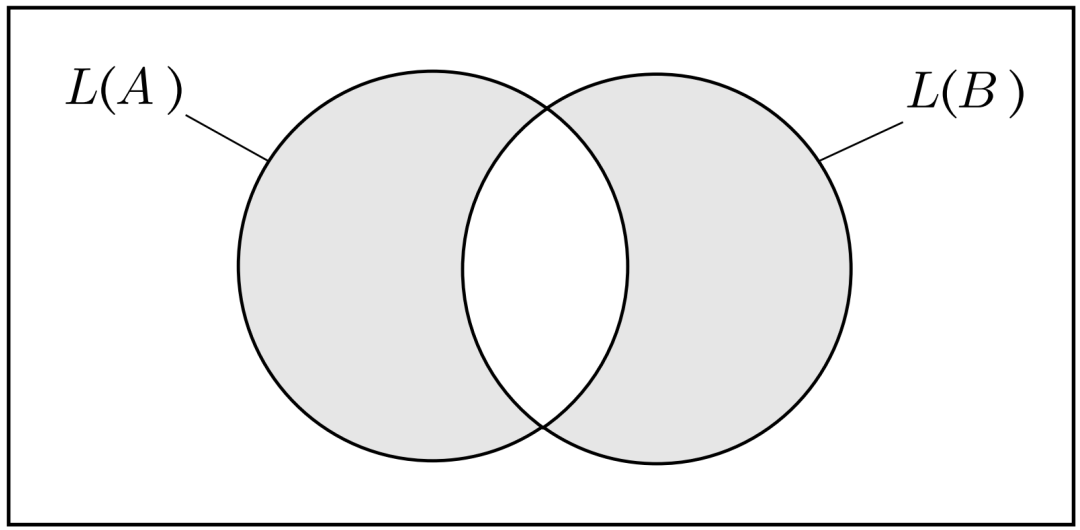
مثال‌هایی از مسائل حل‌پذیر (تصمیم‌پذیر)

Theorem: EQ_{DFA} ***is a decidable language.***

Proof: *We construct a new DFA* C *from* A *and* B*, where* C *accepts only those strings that are accepted by either* A *or* B ***but not by both****. Thus, if* A *and* B *recognize the same language,* C *will accept nothing. The language of* C *is*

$$L(C) = (L(A) \cap \overline{L(B)}) \cup (L(B) \cap \overline{L(A)}).$$

This expression is sometimes called the ***symmetric difference*** *of* $L(A)$ *and* $L(B)$ *and is illustrated in the following figure. The symmetric difference is useful here because* $L(C) = \varnothing$ *iff* $L(A) = L(B)$*. We can construct* C *from* A *and* B *with the constructions for proving the class of regular languages closed under complementation, union, and intersection.* ***These constructions are algorithms that can be carried out by TMs.*** *Once we have constructed* C*, we can use a decider for* E_{DFA} *to test whether* $L(C)$ *is empty. If it is empty,* $L(A)$ *and* $L(B)$ *must be equal.*



$F =$ "On input $\langle A, B \rangle$, where A and B are DFAs:

1. Construct DFA C as described.
2. Let T be a decider for E_{DFA}. Run T on input $\langle C \rangle$.
3. If T accepts, accept. If T rejects, reject."

آیا با روندی مشابه با روند فوق، می‌توان نشان داد EQ_{CFG} تصمیم‌پذیر است؟

Next, we consider the problem of determining whether two context-free grammars generate the same language. Let

$$EQ_{CFG} = \{\langle G, H\rangle | G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}.$$

The above theorem, gave an algorithm that decides the analogous language EQ_{DFA} for finite automata. We used the decision procedure for E_{DFA} to prove that EQ_{DFA} is decidable. Because E_{CFG} also is decidable, you might think that we can use a similar strategy to prove that EQ_{CFG} is decidable. But something is wrong with this idea! The class of context-free languages is not closed under complementation or intersection. In fact, EQ_{CFG} is not decidable.

Example: *Let*

$$\textbf{ALL}_{\textbf{DFA}} = \{\langle A \rangle | A \text{ is a DFA and } L(A) = \Sigma^*\}.$$

Show that $\textbf{ALL}_{\textbf{DFA}}$ *is decidable.*

رویکرد اول برای اثبات:

M_1 = *"On input* $\langle A \rangle$*, where* A *is a DFA:*

1. *Create a DFA* B *that consists only of the initial state* q_0 *which is a goal state. For each symbol of the alphabet, there is a transition from* q_0 *to* q_0.
2. *Let* T *be a decider for* $\textbf{EQ}_{\textbf{DFA}}$. *Run* T *on* $\langle A, B \rangle$.
3. *If* T *accepts, accept; if* T *rejects, reject."*

رویکرد دوم برای اثبات:

M_2 = *"On input $\langle A \rangle$ where A is a DFA:*

1. Construct DFA B that recognizes $\overline{L(A)}$ by swapping accept and nonaccept states in A.

2. Let T be a decider for E_{DFA}. Run T on $\langle B \rangle$.

3. If T accepts, accept; if T rejects, reject."

Example: *Let*

$$\textit{INFINITE}_{\textit{DFA}} = \{\langle A \rangle | A \text{ is a DFA and } L(A) \text{ is an infinite language}\}.$$

Show that $\textit{INFINITE}_{\textit{DFA}}$ *is decidable.*
The following TM I *decides* $\textit{INFINITE}_{\textit{DFA}}$.

$I =$ "*On input* $\langle A \rangle$, *where* A *is a DFA:*

1. *Let* k *be the number of states of* A.
2. *Construct a DFA* D *that accepts all strings of length* k *or more.*
3. *Construct a DFA* M *such that* $L(M) = L(A) \cap L(D)$.
4. *Test* $L(M) = \varnothing$ *using a decider* T *for* $E_{\textit{DFA}}$.
5. *If* T *accepts, reject; if* T *rejects, accept.*"

This algorithm works because a DFA that accepts infinitely many strings must accept arbitrarily long strings. Therefore, this algorithm accepts such DFAs. Conversely, if the algorithm accepts a DFA, the DFA accepts some string of length k or more, where k is the number of states of the DFA. This string may be pumped in the manner of the pumping lemma for regular languages to obtain infinitely many accepted strings.

Example: *Is the decision problem. "Given a CFG G, and a string x, is $L(G) = \{x\}$?" decidable or undecidable? Give reasons for your answer.*

Solution: *The problem is decidable, and the following is a decision algorithm. First test x for membership in $L(G)$. If $x \notin L(G)$, then $L(G) \neq \{x\}$. If $x \in L(G)$, then $L(G) = \{x\}$ if and only if $L(G) \cap (\Sigma^* - \{x\}) = \varnothing$. We can test this last condition as follows. Construct a PDA M accepting $L(G)$. Then, construct a PDA M_1 accepting $L(M) \cap (\Sigma^* - \{x\})$, which is the intersection of $L(M)$ and a regular language. Construct a CFG G_1 generating the language $L(M_1)$. Finally, use a decider for E_{CFG} to determine whether $L(G_1) = \varnothing$.*

Example: *Let $A = \{\langle M \rangle | \ M$ is a DFA that doesn't accept any string containing an odd number of 1s$\}$. Show that A is decidable.*

Solution: *The following TM decides A.*

"On input $\langle M \rangle$:

1. *Construct a DFA O that accepts every string containing an odd number of 1s.*
2. *Construct a DFA B such that $L(B) = L(M) \cap L(O)$.*
3. *Test whether $L(B) = \varnothing$ using a decider T for E_{DFA}.*
4. *If T accepts, accept; if T rejects, reject."*

Example: *Let $\Sigma = \{0, 1\}$. Show that the problem of determining whether a CFG generates some string in 1^* is decidable. In other words, show that*

$$\{\langle G \rangle | G \text{ is a CFG over } \{0,1\} \text{ and } 1^* \cap L(G) \neq \varnothing\}$$

is a decidable language.

Solution: *We showed that if C is a context-free language and R is a regular language, then $C \cap R$ is context free. Therefore, $1^* \cap L(G)$ is context free. The following TM decides the language of this problem.*

"On input $\langle G \rangle$:

1. *Construct CFG H such that $L(H) = 1^* \cap L(G)$.*
2. *Test whether $L(H) = \varnothing$ using a decider R for E_{CFG}.*
3. *If R accepts, reject; if R rejects, accept."*

The language A_{DLBA}

Here, A_{DLBA} is the problem of determining whether an DLBA accepts its input. Even though A_{DLBA} is the same as the undecidable problem A_{TM} where the Turing machine is restricted to be a DLBA, we can show that A_{DLBA} is decidable. Let

$$A_{DLBA} = \{\langle M, w\rangle | M \text{ is a DLBA that accepts string } w\}.$$

Before proving the decidability of A_{DLBA}, we find the following lemma useful. It says that a DLBA can have only a limited number of configurations when a string of length n is the input.

Lemma: *Let M be a DLBA with q states and g symbols in the tape alphabet. There are exactly qng^n distinct configurations of M for a tape of length n.*

Proof: *Recall that a configuration of M is like a snapshot in the middle of its computation. A configuration consists of the state of the control, position of the head, and contents of the tape. Here, M has q states. The length of its tape is n, so the head can be in one of n positions, and g^n possible strings of tape symbols appear on the tape. The product of these three quantities is the total number of different configurations of M with a tape of length n.*

Theorem: A_{DLBA} **is decidable.**

Proof Idea: In order to decide whether DLBA M accepts input w, we simulate M on w. During the course of the simulation, if M halts and accepts or rejects, we accept or reject accordingly. The difficulty occurs if M loops on w. We need to be able to detect looping so that we can halt and reject. The idea for detecting when M is looping is that as M computes on w, it goes from configuration to configuration. If M ever repeats a configuration, it would go on to repeat this configuration over and over again and thus be in a loop. Because M is a DLBA, the amount of tape available to it is limited. By the above lemma, M can be in only a limited number of configurations on this amount of tape. Therefore, only a limited amount of time is available to M before it will enter some configuration that it has previously entered. Detecting that M is looping is possible by simulating M for the number of steps given by the above lemma. If M has not halted by then, it must be looping.

Proof: The algorithm that decides A_{DLBA} is as follows.

$L =$ "On input $\langle M, w \rangle$, where M is a DLBA and w is a string:
1. Simulate M on w for qng^n steps or until it halts.
2. If M has halted, accept if it has accepted and reject if it has rejected. If it has not halted, reject."

If M on w has not halted within qng^n steps, it must be repeating a configuration according to the above lemma and therefore looping. That is why our algorithm rejects in this instance.
The above theorem shows that DLBAs and TMs differ in one essential way: For DLBAs the acceptance problem is decidable, but for TMs it isn't. However, certain other problems involving DLBAs remain undecidable. One is the emptiness problem

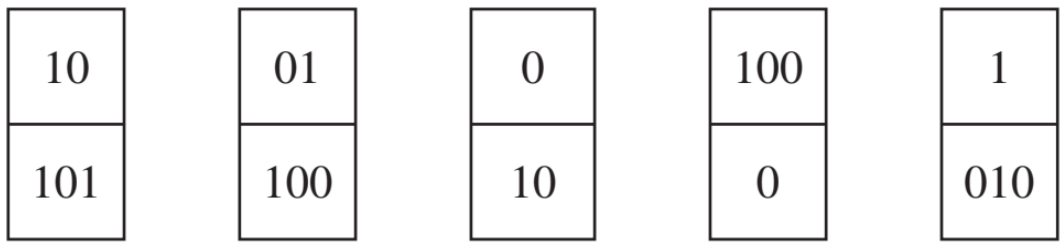
$$E_{DLBA} = \{\langle M \rangle | M \text{ is a DLBA where } L(M) = \varnothing\}.$$

مثال‌هایی از مسائل حل‌ناپذیر (تصمیم‌ناپذیر)

A Simple Undecidable Problem: Post Correspondence Problem (PCP)

The undecidability of the halting problem has many consequences of practical interest, particularly in the area of context-free languages. But in many instances it is cumbersome to work with the halting problem directly, and it is convenient to establish some intermediate results that bridge the gap between the halting problem and other problems. These intermediate results follow from the undecidability of the halting problem, but are more closely related to the problems we want to study and therefore make the arguments easier. One such intermediate result is the Post correspondence problem.

Think of each of the five rectangles in the following figure as a domino, and assume that there is an unlimited supply of each of the five. The question is whether it is possible to make a horizontal line of one or more dominoes, with duplicates allowed, so that the string obtained by reading across the top halves matches the one obtained by reading across the bottom. In this example, the answer is yes. One way to do it is shown in the following figure.



(*a*)

10	1	01	0	100	100	0	100
101	010	100	10	0	0	10	0

(*b*)

☞ *In fact, the task is to make a list of the dominos (repetitions permitted) so that the string we get by reading off the symbols on the top is the same as the string of symbols on the bottom. This list is called a match.*

$$\left\{ \left[\frac{b}{ca}\right], \left[\frac{a}{ab}\right], \left[\frac{ca}{a}\right], \left[\frac{abc}{c}\right] \right\} \text{☞} \left[\frac{a}{ab}\right]\left[\frac{b}{ca}\right]\left[\frac{ca}{a}\right]\left[\frac{a}{ab}\right]\left[\frac{abc}{c}\right]$$

Reading off the top string we get $abcaaabc$*, which is the same as reading off the bottom.*

☞ *For some collections of dominos, finding a match may not be possible. For example, the collection* $\left\{ \left[\frac{abc}{ab}\right], \left[\frac{ca}{a}\right], \left[\frac{acc}{ba}\right] \right\}$ *cannot contain a match because every top string is longer than the corresponding bottom string. The Post Correspondence Problem is to determine whether a collection of dominos has a match. This problem is unsolvable by algorithms.*

Definition (Post Correspondence Problem): *An instance of PCP is a set* $\{(\alpha_1, \beta_1), (\alpha_2, \beta_2), \ldots, (\alpha_n, \beta_n)\}$ *of pairs, where* $n \geq 1$ *and the* α_i*'s and* β_i*'s are all* ***nonnull strings*** *over an alphabet* Σ*. The decision problem is this: Given an instance of this type, do there exist a positive integer* k *and a sequence of integers* $i_1, i_2, \ldots, i_k$*, with each* i_j *satisfying* $1 \leq i_j \leq n$*, satisfying*

$$\alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_k} = \beta_{i_1}\beta_{i_2}\cdots\beta_{i_k}.$$

Let PCP $= \{\langle P\rangle | P$ *is an* ***instance of*** *the Post Correspondence Problem* ***with a match***$\}$*. It can be proved that PCP is* ***undecidable.***

Undecidable Problems Involving Context-Free Languages

Here, we will look at an approach to obtaining undecidability results involving context-free grammars and CFLs. This approach uses the fact that PCP is undecidable. For an instance

$$\{(\alpha_1, \beta_1), (\alpha_2, \beta_2), \ldots, (\alpha_n, \beta_n)\}$$

of PCP, where the α_i's and β_i's are strings over Σ, we construct two context-free grammars G_α and G_β whose properties are related to those of the correspondence system. Let $c_1, c_2, \ldots, c_n$ be symbols that are not in Σ. G_α will be the CFG with start symbol S_α and the $2n$ productions

$$S_\alpha \to \alpha_i S_\alpha c_i | \alpha_i c_i \ (1 \leq i \leq n)$$

and G_β is constructed the same way from the strings β_i. Then for every string $c = c_{i_1} c_{i_2} \cdots c_{i_k}$, where $k \geq 1$, there is exactly one string x in $L(G_\alpha)$ and exactly one string y in $L(G_\beta)$ that is a string in Σ^* followed by c.

The string x is

$$\alpha_{i_k} \cdots \alpha_{i_2} \alpha_{i_1} c_{i_1} c_{i_2} \cdots c_{i_k},$$

and the string y is

$$\beta_{i_k} \cdots \beta_{i_2} \beta_{i_1} c_{i_1} c_{i_2} \cdots c_{i_k}.$$

Both x and y have unique derivations in their respective grammars, and both derivations are determined by c.

Theorem: These two problems are undecidable:
1. CFGNonemptyIntersection: Given two CFGs G_1 and G_2, is $L(G_1) \cap L(G_2)$ nonempty?
2. IsAmbiguous: Given a CFG G, is G ambiguous?

Proof: *We will reduce PCP both to CFGNonemptyIntersection and to IsAmbiguous, and the two reductions will be very similar. Starting with an arbitrary instance I of PCP, involving α_i's and β_i's as above, we construct the two context-free grammars G_α and G_β as we have described. For the first reduction, we let $F_1(I)$ be the instance (G_α, G_β) of CFGNonemptyIntersection, and for the second we let $F_2(I)$ be the grammar G constructed in the usual way to generate $L(G_\alpha) \cup L(G_\beta)$: The start symbol of G is S, and the productions include those of G_α, those of G_β, and the two additional ones $S \to S_\alpha | S_\beta$. If I is a yes-instance of PCP, then for some sequence of integers $i_1, i_2, \ldots, i_k$,*

$$\alpha_{i_k} \alpha_{i_{k-1}} \cdots \alpha_{i_1} = \beta_{i_k} \beta_{i_{k-1}} \cdots \beta_{i_1}$$

so that

$$\alpha_{i_k} \alpha_{i_{k-1}} \cdots \alpha_{i_1} c_{i_1} \cdots c_{i_k} = \beta_{i_k} \beta_{i_{k-1}} \cdots \beta_{i_1} c_{i_1} \cdots c_{i_k}$$

It follows that this string is an element of the intersection $L(G_\alpha) \cap L(G_\beta)$, and that it has two derivations in G, one starting with $S \Rightarrow S_\alpha$ and the other with $S \Rightarrow S_\beta$. Therefore, both $F_1(I)$ and $F_2(I)$ are yes-instances of their respective decision problems. On the other hand, if either $F_1(I)$ or $F_2(I)$ is a yes-instance (the two statements are equivalent), then for some $x \in \Sigma^*$ and some sequence of integers $i_1, i_2, \ldots, i_k$, there is a string $xc_{i_1}c_{i_2}\cdots c_{i_k}$ that is in the intersection $L(G_\alpha) \cap L(G_\beta)$ and can therefore be derived from either S_α or S_β. As we observed above, this implies that

$$\alpha_{i_k}\alpha_{i_{k-1}}\cdots\alpha_{i_1}c_{i_1}\cdots c_{i_k} = \beta_{i_k}\beta_{i_{k-1}}\cdots\beta_{i_1}c_{i_1}\cdots c_{i_k}$$

which implies that I is a yes-instance of PCP.

Example: *Show that the Post Correspondence Problem is decidable over the unary alphabet $\Sigma = \{a\}$. (Show that the special case of PCP in which the alphabet has only one symbol is decidable.)*

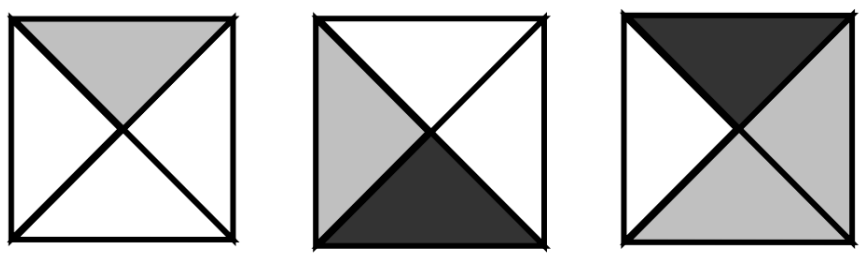
Solution: *Suppose*

$$(\alpha_1, \beta_1), (\alpha_2, \beta_2), \ldots, (\alpha_n, \beta_n)$$

is an instance of PCP in which the α_i's and β_i's are all strings over the alphabet $\{a\}$, and let $d_i = |\alpha_i| - |\beta_i|$. If $d_i = 0$ for some i, the instance is a yes-instance. If $d_i > 0$ for every i, or if $d_i < 0$ for every i, then the instance is a no-instance. And finally, if $d_i = p > 0$ and $d_j = q < 0$ for some i and j, then it is a yes-instance, because $\alpha_i^q \alpha_j^p = \beta_q^i \beta_p^j$.

Tiling Problems

Consider a class of tiles called Wang tiles or Wang dominos. A Wang tile is a square that has been divided into four regions by drawing two diagonal lines, as shown in the following figure. Each region is colored with one of a fixed set of colors.

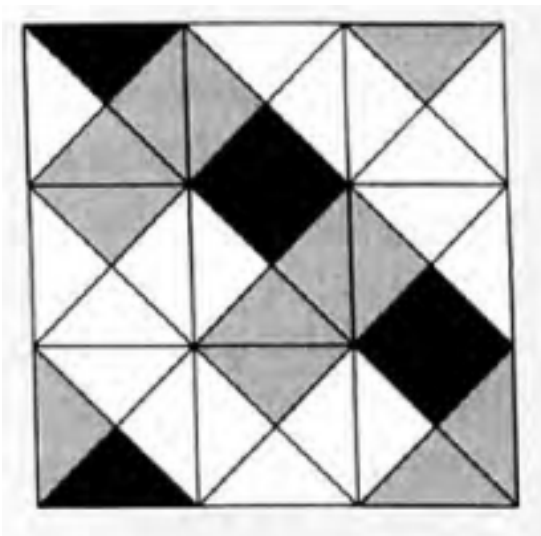


Now suppose that you are given a finite set of such tile designs, all of the same size, for example the set of three designs shown here. Further suppose that you have an infinite supply of each type of tile. Then we may ask whether or not it possible to tile an arbitrary surface in the plane with the available designs while adhering to the following rules:

1. *Each tile must be placed so that it is touching its neighbors on all four sides (if such neighbors exist). In other words, no gaps or overlaps are allowed.*
2. *When two tiles are placed so that they adjoin each other, the adjoining regions of the two tiles must be the same color.*
3. *No rotations or flips are allowed.*

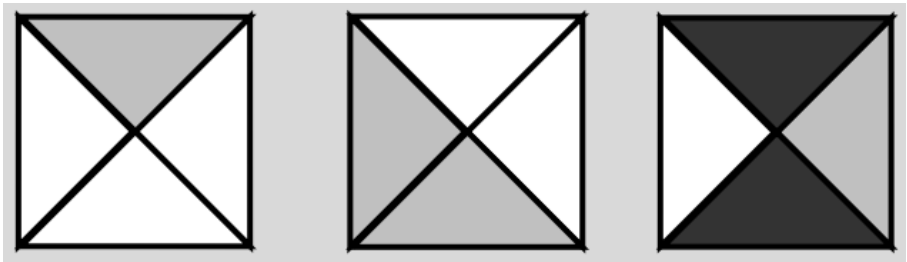
A Set of Tiles that Can Tile the Plane

Example: *The set of tiles shown above can be used to tile any surface in the plane. Here is a small piece of the pattern that can be built and then repeated as necessary since its right and left sides match, as do its top and bottom:*



A Set of Tiles that Cannot Tile the Plane

Example: *Now consider a new set of tiles:*



Only a small number of small regions can be tiled with this set. To see this, start with tile 1, add a tile below it and then try to extend the pattern sideways. Then start with tile 2 and show the same thing. Then observe that tile 3, the only one remaining, cannot be placed next to itself.

We can formulate the tiling problem, as we have just described it, as a language to be decided. To do that, we need to define a way to encode each instance of the tiling problem (i.e., a set of tile designs) as a string. We will represent each design as an ordered 4-tuple of values drawn from the set $\{G, W, B\}$. To describe a design, start in the top region and then go around the tile clockwise. So, for example, the first tile set we considered could be represented as: `(G W W W)(W W B G)(B G G W)`.

Now we can define: ***TILES*** $= \{\langle T \rangle$: *every finite surface on the plane can be tiled, according to the rules, with the tile set* $T\}$.

Theorem: *The language* ***TILES is not in decidable.*** *It is also* ***not Turing-recognizable.***

$REGULAR_{TM}$

Another interesting computational problem regarding Turing machines concerns determining whether a given Turing machine recognizes a language that also can be recognized by a simpler computational model. For example, we let $REGULAR_{TM}$ be the problem of determining whether a given Turing machine has an equivalent finite automaton. This problem is the same as determining whether the Turing machine recognizes a regular language. Let

$$REGULAR_{TM} = \{\langle M \rangle | M \text{ is a TM and } L(M) \text{ is a regular language}\}.$$

Theorem: $REGULAR_{TM}$ *is undecidable.*

Proof Idea: As usual for undecidability theorems, this proof is by reduction from A_{TM}. We assume that $REGULAR_{TM}$ is decidable by a TM R and use this assumption to construct a TM S that decides A_{TM}. Less obvious now is how to use R's ability to assist S in its task. Nonetheless, we can do so.

The idea is for S to take its input $\langle M, w \rangle$ and modify M so that the resulting TM recognizes a regular language if and only if M accepts w. We call the modified machine M_2. We design M_2 to recognize the non-regular language $\{0^n 1^n | n \geq 0\}$ if M does not accept w, and to recognize the regular language Σ^* if M accepts w. We must specify how S can construct such an M_2 from M and w. Here, M_2 works by automatically accepting all strings in $\{0^n 1^n | n \geq 0\}$. In addition, if M accepts w, M_2 accepts all other strings. Note that the TM M_2 is not constructed for the purposes of actually running it on some input—a common confusion. We construct M_2 only for the purpose of feeding its description into the decider for REGULAR_{TM} that we have assumed to exist. Once this decider returns its answer, we can use it to obtain the answer to whether M accepts w. Thus, we can decide A_{TM}, a contradiction.

Proof: *We let R be a TM that decides $\text{REGULAR}_{\text{TM}}$ and construct TM S to decide A_{TM}. Then S works in the following manner.*

S = "On input $\langle M, w \rangle$, where M is a TM and w is a string:

1. Construct the following TM M_2.

M_2 = "On input x:

1. If x has the form $0^n 1^n$, accept.

2. If x does not have this form, run M on input w and accept if M accepts w."

2. Run R on input $\langle M_2 \rangle$.

3. If R accepts, accept; if R rejects, reject."

Similarly, the problems of testing whether the language of a Turing machine is a context-free language, a decidable language, or even a finite language can be shown to be undecidable with similar proofs. ***In fact, a general result, called Rice's theorem, states that determining any property of the languages recognized by Turing machines is undecidable. We give Rice's theorem in Problem 5.28.***

A⋆ **5.28** **Rice's theorem.** Let P be any nontrivial property of the language of a Turing machine. Prove that the problem of determining whether a given Turing machine's language has property P is undecidable.

In more formal terms, let P be a language consisting of Turing machine descriptions where P fulfills two conditions. First, P is nontrivial—it contains some, but not all, TM descriptions. Second, P is a property of the TM's language—whenever $L(M_1) = L(M_2)$, we have $\langle M_1 \rangle \in P$ iff $\langle M_2 \rangle \in P$. Here, M_1 and M_2 are any TMs. Prove that P is an undecidable language.

یک جدول بسیار مهم از کتاب ریچ

Table 26.1 Comparing the classes of languages.

	Regular	*Context-Free*	*Context-Sensitive*	**D**	**SD**
Automaton	FSM	PDA	LBA		TM
Grammar(s)	Regular expressions	Context-free	Context-sensitive		Unrestricted
ND = D?	Yes	No	unknown		Yes
Closed under:					
Concatenation	Yes	Yes	Yes	Yes	Yes
Union	Yes	Yes	Yes	Yes	Yes
Kleene star	Yes	Yes	Yes	Yes	Yes
Complement	Yes	No	Yes	Yes	No
Intersection	Yes	No	Yes	Yes	Yes
∩ with Regular	Yes	Yes	Yes	Yes	Yes
Decidable:					
Membership	Yes	Yes	Yes		No
Emptiness	Yes	Yes	No		No
Finiteness	Yes	Yes	No		No
= Σ^*	Yes	No	No		No
Equivalence	Yes	No	No		No

تمرین:

Is the decision problem. "Given a CFG G and a regular language R, is $L(G) \subseteq R$?" decidable or undecidable? Give reasons for your answer.